CODE ALPHA CYBER SECURITY INTERNSHIP

TASK 3 – SECURE CODING REVIEW

By - Yaman Dahiya

Language: Python

Application: Web application for managing user profiles.

Code Review:

```
from flask import Flask, request, jsonify
import sqlite3
app = Flask(__name__)
DB_FILE = 'user_profiles.db'
def connect_db():
    return sqlite3.connect(DB_FILE)
def init_db():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                    id INTEGER PRIMARY KEY,
                    username TEXT UNIQUE NOT NULL,
                    password TEXT NOT NULL,
                    email TEXT UNIQUE NOT NULL
                )''')
    conn.commit()
    conn.close()
@app.route('/register', methods=['POST'])
```

```
def register():
    data = request.get_json()
    username = data['username']
    password = data['password']
    email = data['email']
    conn = connect_db()
    cursor = conn.cursor()
    try:
        cursor.execute("INSERT INTO users (username, password, email)
VALUES (?, ?, ?)", (username, password, email))
        conn.commit()
        conn.close()
        return jsonify({'message': 'User registered successfully'})
    except sqlite3.IntegrityError:
        conn.close()
        return jsonify({'error': 'Username or email already exists'})
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data['username']
    password = data['password']
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE username = ? AND
password = ?", (username, password))
    user = cursor.fetchone()
    conn.close()
    if user:
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'error': 'Invalid username or password'})
if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Security Vulnerabilities and Recommendations:

1. SQL Injection Vulnerability:

   Vulnerability: The code directly concatenates user inputs into SQL queries, making it susceptible to SQL injection attacks.
   Recommendation: Use parameterized queries (as seen in some parts of the code) consistently to prevent SQL injection. Parameterized queries ensure that user input is treated as data, not as part of the query structure.

2. Storing Passwords in Plain Text:

   Vulnerability: Passwords are stored in plain text in the database, which is a major security risk if the database is compromised.
   Recommendation: Hash passwords before storing them in the database. Use strong and industry-standard hashing algorithms like bcrypt. This ensures that even if the database is compromised, passwords cannot be easily retrieved.

3. Lack of Input Validation:

   Vulnerability: The code doesn't perform sufficient input validation. It assumes that the incoming JSON data contains all required fields.
   Recommendation: Implement robust input validation to ensure that the required fields are present and that they meet expected formats. For instance, validate email addresses, enforce password complexity requirements, and ensure the absence of unexpected characters that could lead to injection attacks.

4. Exposing Sensitive Information:

   Vulnerability: The application returns specific error messages like "Username or email already exists" or "Invalid username or password" which can aid attackers in understanding the system and targeting attacks more effectively.
   Recommendation: Provide generic error messages to users in case of authentication failures or registration issues. Log detailed error messages on the server side for debugging purposes, but return only generic error messages to users to avoid leaking sensitive information.

5. No Transport Layer Security (TLS):

   Vulnerability: The application does not use HTTPS, which means that data transmitted between the client and the server is not encrypted, exposing it to potential interception and manipulation.
   Recommendation: Implement HTTPS by obtaining an SSL certificate and configuring the web server to use it. This ensures that all data exchanged between the client and the server is encrypted, enhancing confidentiality and integrity.

By addressing these vulnerabilities and adopting secure coding practices, the application can significantly improve its security posture and mitigate potential risks associated with user data manipulation and unauthorized access.