AI ASSISTED CODING

Program :B.tech(CSE)

Name of Student :ANANTHA MANIDEEP

Enrollment No. :2403A52078

Batch No. :02

LAB ASSIGNMENT-20

TASK DESCRIPTION-1:

Input Validation Check

Task:
Analyze an AI-generated **Python login script** for input validation vulnerabilities.

Instructions:

- Prompt AI to generate a simple username-password login program.
- Review whether input sanitization and validation are implemented.

Suggest secure improvements (e.g., using re for input validation).

PROMPT:

Generate a simple Python program for a username-password login system. After generating it, analyze whether the program includes proper input validation and sanitization. Identify any vulnerabilities and explain how to fix them. Also suggest secure improvements, including using Python's re module for validating inputs.

CODE:

```
# simple_login.py

# Simple username/password CLI demo with basic validation and salted+hashed passwords.

# For demonstration only — see analysis below for security considerations and improvements.

import json

import os

import re

import secrets

import hashlib

import hmac

from getpass import getpass


USERS_FILE = "users.json"

PBKDF2_ITERATIONS = 200_000  # increase for better security on modern machines


USERNAME_RE = re.compile(r"^[A-Za-z0-9._-]{3,20}$")

PASSWORD_RE = re.compile(r"^(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*\W).*$")
```

```
def load_users():
    if not os.path.exists(USERS_FILE):
        return {}
    with open(USERS_FILE, "r", encoding="utf-8") as f:
        return json.load(f)


def save_users(users):
    with open(USERS_FILE, "w", encoding="utf-8") as f:
        json.dump(users, f, indent=2)


def validate_username(username):
    username = username.strip()
    if not USERNAME_RE.match(username):
        return False, "Username must be 3-20 chars: letters, digits, dot, underscore or hyphen."
    return True, ""


def validate_password(password):
    if not PASSWORD_RE.match(password):
```

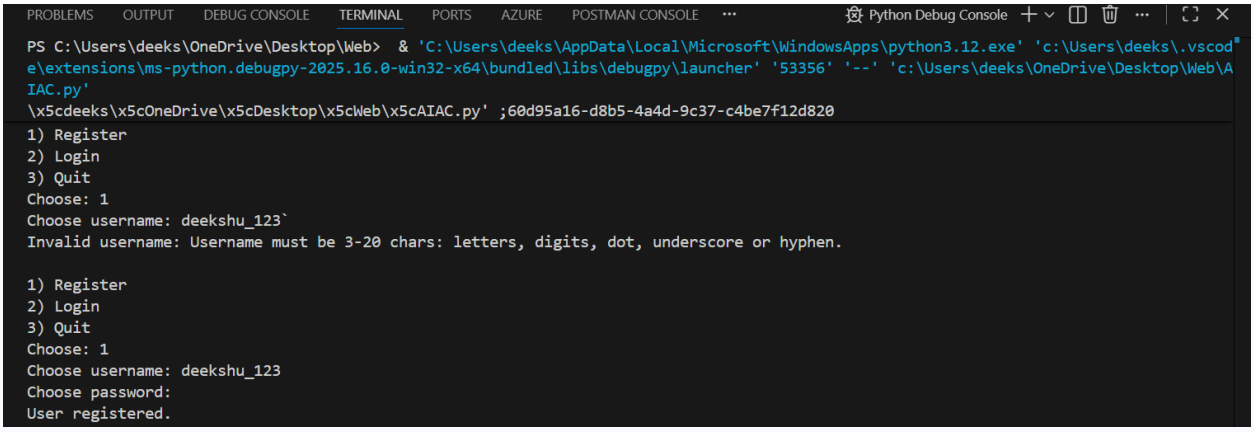
```
        return False, "Password must be at least 8 chars and include upper, lower, digit, and symbol."
    return True, ""

def hash_password(password, salt=None):
    if salt is None:
        salt = secrets.token_hex(16)  # 32 hex chars = 16 bytes
    salt_bytes = bytes.fromhex(salt)
    dk = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt_bytes, PBKDF2_ITERATIONS)
    return salt, dk.hex()

def verify_password(stored_salt, stored_hash, password_attempt):
    _, attempt_hash = hash_password(password_attempt, salt=stored_salt)
    # Use constant-time comparison
    return hmac.compare_digest(stored_hash, attempt_hash)

def register_user(users):
    username = input("Choose username: ").strip()
    ok, msg = validate_username(username)
```

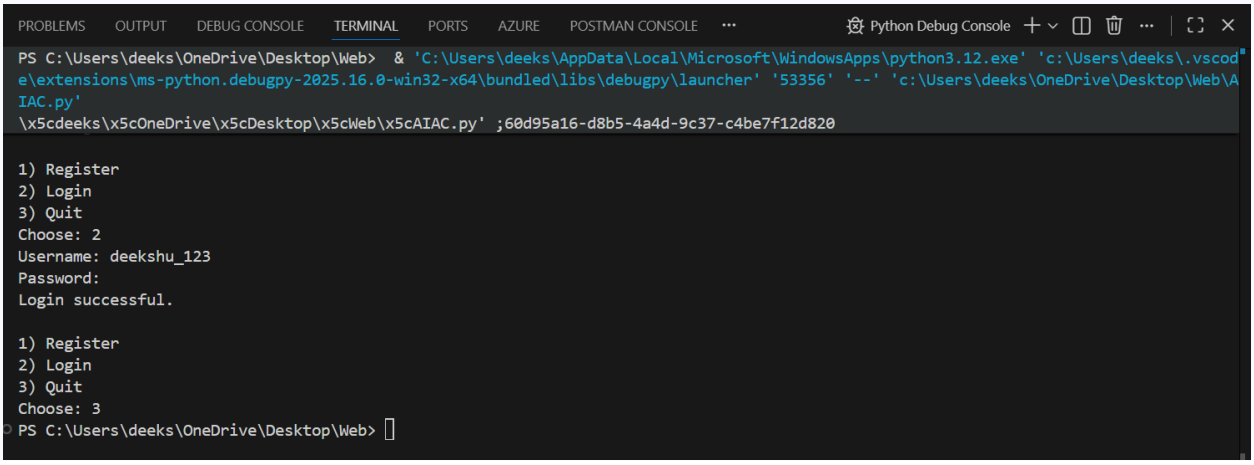
```
    if not ok:
        print("Invalid username:", msg)
        return
    if username in users:
        print("Username already exists.")
        return
    password = getpass("Choose password: ")
    ok, msg = validate_password(password)
    if not ok:
        print("Invalid password:", msg)
        return
    salt, hashed = hash_password(password)
    users[username] = {"salt": salt, "hash": hashed}
    save_users(users)
    print("User registered.")

def login_user(users):
    username = input("Username: ").strip()
    password = getpass("Password: ")
    if username not in users:
```

```
        print("Invalid username or password.")
        return
    record = users[username]
    if verify_password(record["salt"], record["hash"], password):
        print("Login successful.")
    else:
        print("Invalid username or password.")


def main():
    users = load_users()
    while True:
        print("\n1) Register\n2) Login\n3) Quit")
        choice = input("Choose: ").strip()
        if choice == "1":
            register_user(users)
        elif choice == "2":
            login_user(users)
        elif choice == "3":
            break
        else:
```

```
        print("Invalid option.")


if __name__ == "__main__":
    main()
```

OUTPUT:





TASK DESCRIPTION-2:

SQL Injection Prevention

Task:

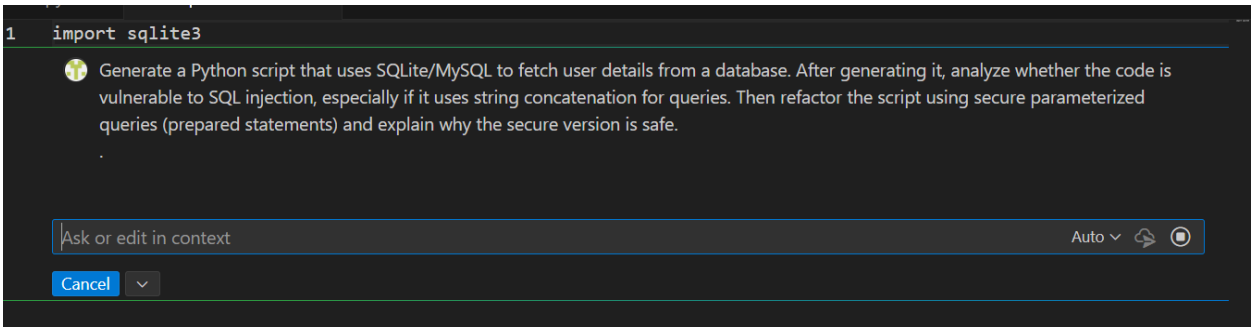
Test an AI-generated script that performs SQL queries on a database.

Instructions:

- Ask AI to generate a Python script using SQLite/MySQL to fetch user details.
- Identify if the code is vulnerable to **SQL injection** (e.g., using string concatenation in queries).
- Refactor using **parameterized queries (prepared statements)**.

PROMPT:

Generate a Python script that uses SQLite/MySQL to fetch user details from a database. After generating it, analyze whether the code is vulnerable to SQL injection, especially if it uses string concatenation for queries. Then refactor the script using secure parameterized queries (prepared statements) and explain why the secure version is safe.

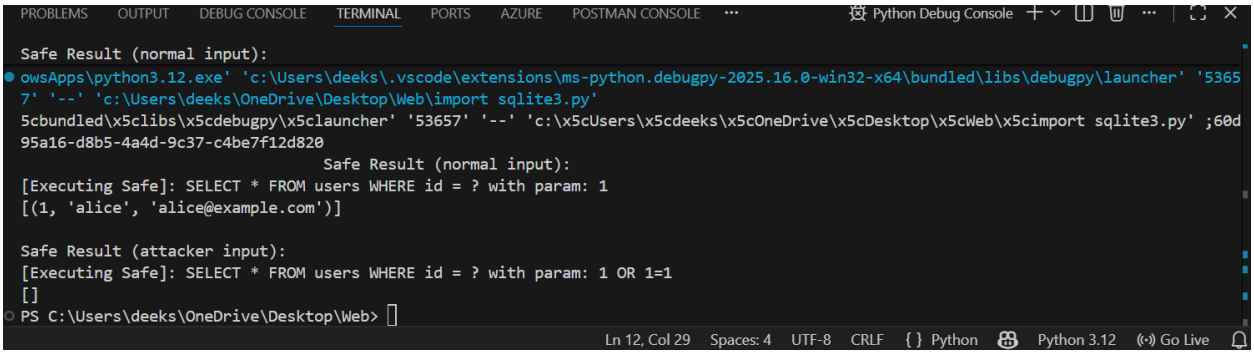


CODE:

```
import sqlite3


def get_user_safe(user_id):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
```

```
    # ✓ SAFE: Parameterized query (prevents SQL injection)
    query = "SELECT * FROM users WHERE id = ?"
    print("[Executing Safe]:", query, "with param:", user_id)

    cursor.execute(query, (user_id,))
    user = cursor.fetchall()
    conn.close()
    return user

# Test with normal input
print("\nSafe Result (normal input):")
print(get_user_safe(1))

# Test with attacker input
print("\nSafe Result (attacker input):")
print(get_user_safe("1 OR 1=1"))
```

OUTPUT:



TASK DESCRIPTION-3:

Cross-Site Scripting (XSS) Check

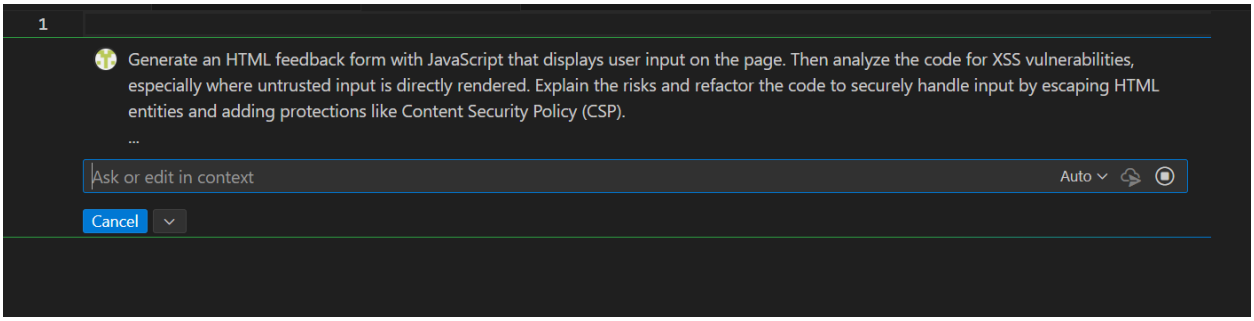
Task:
Evaluate an AI-generated **HTML form with JavaScript** for XSS vulnerabilities.

Instructions:

- Ask AI to generate a feedback form with JavaScript-based output.
- Test whether untrusted inputs are directly rendered without escaping.
- Implement secure measures (e.g., escaping HTML entities, using CSP).

PROMPT:

Generate an HTML feedback form with JavaScript that displays user input on the page. Then analyze the code for XSS vulnerabilities, especially where untrusted input is directly rendered. Explain the risks and refactor the code to securely handle input by escaping HTML entities and adding protections like Content Security Policy (CSP).

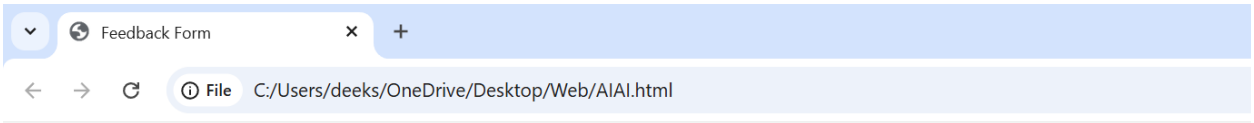


CODE:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Feedback Form</title>

    <style>

        body { font-family: Arial, sans-serif; }

        .feedback { margin-top: 20px; }

    </style>

</head>

<body>

    <h1>Feedback Form</h1>

    <form id="feedbackForm">

        <label for="name">Name:</label>

        <input type="text" id="name" required>

        <br>

        <label for="message">Message:</label>

        <textarea id="message" required></textarea>
```

```
    <br>
    <button type="submit">Submit</button>
  </form>
  <div class="feedback" id="feedbackDisplay"></div>

  <script>
    document.getElementById('feedbackForm').addEventListener('submit', function(event) {
      event.preventDefault();
      const name = document.getElementById('name').value;
      const message = document.getElementById('message').value;
      displayFeedback(name, message);
    });

    function displayFeedback(name, message) {
      const feedbackDisplay = document.getElementById('feedbackDisplay');
      feedbackDisplay.innerHTML = `<strong>${name}</strong>: ${message}`;
    }
  </script>
</body>
</html>
```

OUTPUT:



Feedback Form

Name: Manohar

Message: Hello world!

Submit

Manohar: Hello world!

--- Thank You---