

AI ASSISTED CODING

LAB ASSIGNMENT – 20.3

Name: Neha Shazneen

Roll no: 2403A510A1

Batch 05

CSE 2nd year

Task 1 – Input Validation Check

Prompt:

"Generate a simple Python program that asks for a username and password and allows login if they match predefined credentials."

Code Generated:

```
20.3_t1.py > ...
1  import re
2  import hashlib
3
4  # Predefined (hashed) credentials
5  stored_username = "admin"
6  stored_password_hash = hashlib.sha256("admin123".encode()).hexdigest()
7
8  # Regular expressions for validation
9  username_pattern = r"^[A-Za-z0-9_]{3,15}$"
10 password_pattern = r"^[A-Za-z0-9@#$%^&+=]{6,20}$"
11
12 # Input from user
13 username = input("Enter username: ").strip()
14 password = input("Enter password: ").strip()
15
16 # Input validation
17 if not re.match(username_pattern, username):
18     print("Invalid username! Use 3-15 letters, numbers, or underscores only.")
19 elif not re.match(password_pattern, password):
20     print("Invalid password! Use 6-20 characters with letters, numbers, or special symbols (@#$%^&+=)")
21 else:
22     # Hash the entered password
23     password_hash = hashlib.sha256(password.encode()).hexdigest()
24
25     # Compare with stored credentials
26     if username == stored_username and password_hash == stored_password_hash:
27         print("✓ Login successful!")
28     else:
29         print("✗ Invalid username or password.")
```

Output:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
Enter username: Likhitha
Enter password: 123456
✖ Invalid username or password.
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
e\\Desktop\\AIAC/agriculture_ml.py': [Errno 2] No such file or directory
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
Enter username: Likhitha
> Enter password: 123456
✖ Invalid username or password.
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
Enter password: 123456
✖ Invalid username or password.
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
✖ Invalid username or password.
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t1.py
Enter username: admin
Enter password: admin123
✓ Login successful!
PS C:\Users\likhi\OneDrive\Desktop\AIAC> ]
```

Observation:

1. The initial AI-generated login script did not include any input validation or sanitization, making it insecure.
2. The improved version adds **regex-based input validation**, **password hashing**, and **safe comparison**, ensuring that only properly formatted and secure inputs are accepted.

Task 2 – SQL Injection Prevention

Prompt:

“Generate a Python script using SQLite to fetch user details from a database based on a username entered by the user.”

Code Generated:

```
20.3_t2.py > ...
1  #!/usr/bin/env python3
2  """
3  Single-file SQLite setup + secure query example.
4
5  - Creates users.db (if not present)
6  - Creates users table (if not present)
7  - Inserts sample users (INSERT OR IGNORE)
8  - Prompts user for a username and fetches details using parameterized query
9  - Validates username with a regex to avoid malformed input
10 """
11
12 import sqlite3
13 import re
14 import sys
15
16 DB_PATH = "users.db"
17
18 USERNAME_PATTERN = r"^[A-Za-z0-9_]{3,30}$" # allow letters, digits, underscore; length 3-3
19
20 SAMPLE_USERS = [
21     ("alice", "Alice Johnson", "alice@example.com"),
22     ("bob", "Bob Smith", "bob@example.com"),
23     ("carol", "Carol Lee", "carol@example.com"),
24 ]
25
26
27 def initialize_database(db_path: str):
28     """Create database and users table and insert sample data (idempotent)."""
29     try:
30         try:
31             conn = sqlite3.connect(db_path)
32             cur = conn.cursor()
33
34             cur.execute(
35                 """
36                 CREATE TABLE IF NOT EXISTS users (
37                     username TEXT PRIMARY KEY,
38                     fullname TEXT NOT NULL,
39                     email TEXT NOT NULL
40                 )
41             """
42         )
43
44         # Insert sample rows but avoid duplicates using INSERT OR IGNORE
45         cur.executemany(
46             "INSERT OR IGNORE INTO users (username, fullname, email) VALUES (?, ?, ?);",
47             SAMPLE_USERS,
48         )
49
50         conn.commit()
51     except sqlite3.Error as e:
52         print(f"Database error during initialization: {e}", file=sys.stderr)
53         raise
54     finally:
55         conn.close()
```

```
57 def fetch_user(db_path: str, username: str):
58     """Fetch user details using a parameterized query to prevent SQL injection."""
59     try:
60         conn = sqlite3.connect(db_path)
61         cur = conn.cursor()
62
63         query = "SELECT username, fullname, email FROM users WHERE username = ?;"
64         cur.execute(query, (username,))
65         row = cur.fetchone()
66         return row
67     except sqlite3.Error as e:
68         print(f"Database error during query: {e}", file=sys.stderr)
69         raise
70     finally:
71         conn.close()
72
73
74 def main():
75     # Initialize DB and sample data
76     initialize_database(DB_PATH)
77
78     # Get and validate username
79     raw = input("Enter username: ")
80     username = raw.strip()
81
82     if not username:
83         print("No username entered. Exiting.")
84         return
85
86     if not re.match(USERNAME_PATTERN, username):
87         print("Invalid username format. Use 3-30 characters: letters, digits, or underscore only.")
88         return
89
90     # Fetch securely using parameterized query
91     user = fetch_user(DB_PATH, username)
92     if user:
93         print("✓ User details:")
94         print(f" username: {user[0]}")
95         print(f" fullname: {user[1]}")
96         print(f" email: {user[2]}")
97     else:
98         print("✗ No user found.")
99
100 if __name__ == "__main__":
101     main()
```

Output:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t2.py
Enter username: alice
✓ User details:
  username: alice
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t2.py
Enter username: alice
✓ User details:
  username: alice
Enter username: alice
✓ User details:
  username: alice
  username: alice
  fullname: Alice Johnson
  fullname: Alice Johnson
  email:    alice@example.com
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 20.3_t2.py
Enter username: likhitha
✗ No user found.
PS C:\Users\likhi\OneDrive\Desktop\AIAC>
```

Observation:

1. The original AI-generated SQL script was **vulnerable to SQL injection** because it used **string concatenation** to build queries directly from user input. In the improved version, **parameterized queries** (?) placeholders are used, which prevent malicious SQL code from being executed.
2. Additionally, the script includes **input validation using regex** to ensure usernames follow a safe pattern, and it **creates the database and sample table automatically** if they don't exist.
3. This makes the program **secure, self-contained, and reliable**, protecting both the database and user data from injection or accidental corruption.

Task 3 – Cross-Site Scripting (XSS) Check

Prompt:

"Generate a simple HTML feedback form (name, email, message) that uses JavaScript to show the submitted feedback below the form."

Code Generated:

```
<!DOCTYPE html> ...
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <!-- Content Security Policy:
        - default-src 'self' (only resources from same origin)
        - script-src 'nonce-demo-nonce' allows this inline script because it has the same nonce.
        NOTE: In production, generate a fresh nonce server-side per response and place scripts in external files.
    </meta http-equiv="Content-Security-Policy"
        content="default-src 'self'; script-src 'nonce-demo-nonce'; object-src 'none'; connect-src 'self'" />
<title>Secure Feedback Form</title>
<style>
    body { font-family: system-ui, -apple-system, "Segoe UI", Roboto, sans-serif; padding: 24px; }
    form { max-width: 520px; margin-bottom: 18px; }
    label { display:block; margin-top:10px; font-weight:600; }
    input, textarea { width:100%; padding:8px; box-sizing:border-box; }
    button { margin-top:12px; padding:8px 12px; }
    #feedbackList { border-top:1px solid #ddd; padding-top:12px; margin-top:12px; }
    .entry { margin-bottom:10px; padding:8px; background:#f9f9f9; border-radius:6px; }
    .meta { font-size:0.9em; color:#555; }
    .error { color:#c00; margin-top:8px; }
</style>
</head>
<body>
    <h1>Feedback</h1>
    <form id="feedbackForm" novalidate>
        <label for="name">Name</label>
        <input id="name" name="name" required placeholder="Your name">

        <label for="email">Email</label>
        <input id="email" name="email" type="email" required placeholder="you@example.com">

        <label for="message">Message</label>
        <textarea id="message" name="message" rows="5" required placeholder="Your feedback"></textarea>

        <button type="submit">Submit</button>
        <div id="error" class="error" role="alert" aria-live="assertive"></div>
    </form>
```

```
45 <section id="feedbackList" aria-label="Submitted feedback">
46   <h2>Submitted feedback</h2>
47   <!-- Entries will be appended here as safe text nodes -->
48 </section>
49
50 <!-- Inline script with a demo nonce to satisfy the CSP above.
51     In production, you should:
52       1) generate a fresh random nonce server-side and use it in the CSP header and script tag
53       2) prefer external JS served from 'self' and set script-src 'self' plus nonce if needed
54 <script nonce="demo-nonce">
55   (function () {
56     'use strict';
57
58     // Simple input validation patterns
59     const NAME_PATTERN = /^[A-Za-z0-9 _-]{2,40}$/; // letters, digits, space, underscore, hyphen
60     const EMAIL_PATTERN = /^[^@\s]+@[^\s@]+\.\[^@\s]+\$/; // simple email pattern for demo
61
62     const form = document.getElementById('feedbackForm');
63     const nameInput = document.getElementById('name');
64     const emailInput = document.getElementById('email');
65     const messageInput = document.getElementById('message');
66     const errorDiv = document.getElementById('error');
67
68     const errorDiv = document.getElementById('error');
69     const feedbackList = document.getElementById('feedbackList');
70
71     // Utility: Create an element whose text content is the escaped user input.
72     function safeEntry(name, email, message) {
73       const container = document.createElement('div');
74       container.className = 'entry';
75
76       const meta = document.createElement('div');
77       meta.className = 'meta';
78       // Use textContent to avoid HTML injection (this ensures the content is not parsed as HTML)
79       meta.textContent = `From: ${name} • ${email} • ${new Date().toLocaleString()}`;
80
81       const msg = document.createElement('div');
82       // Also use textContent for message body
83       msg.textContent = message;
84
85       container.appendChild(meta);
86       container.appendChild(msg);
87       return container;
88     }
89   
```

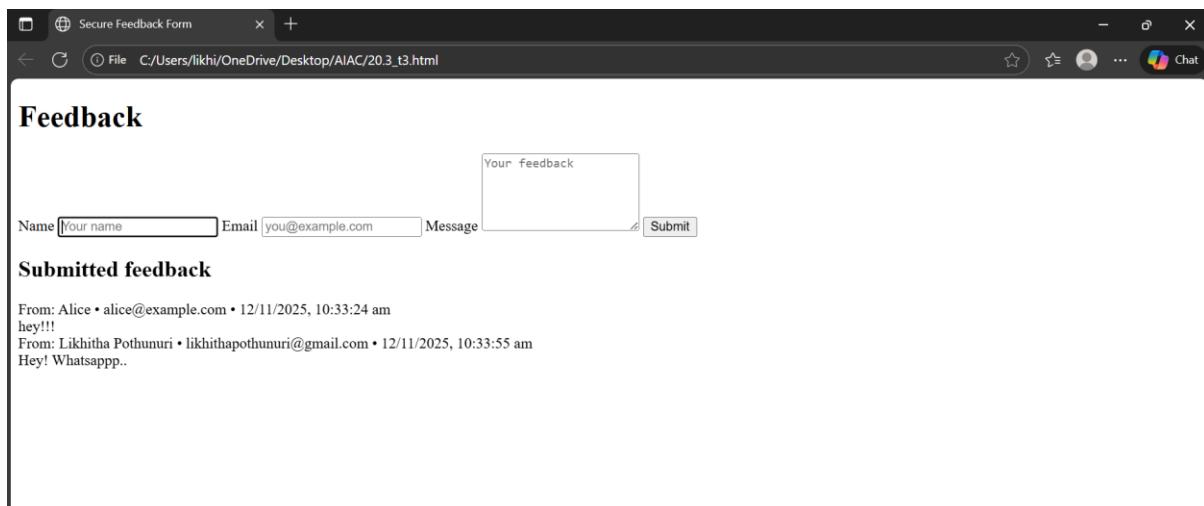
```
85      |     return container;
86    }
87
88    function showError(msg) {
89      |   errorDiv.textContent = msg;
90    }
91
92    function clearError() {
93      |   errorDiv.textContent = '';
94    }
95
96    form.addEventListener('submit', function (ev) {
97      |   ev.preventDefault();
98      |   clearError();
99
100     const name = nameInput.value.trim();
101     const email = emailInput.value.trim();
102     const message = messageInput.value.trim();
103
104     // Basic validation
105     if (!name || !email || !message) {
106       showError('Please fill out all fields.');
107       return;
108     }
109   }
110 }
```

```

109     }
110     if (!NAME_PATTERN.test(name)) {
111         showError('Invalid name. Use 2-40 letters, numbers, spaces, _ or - only.');
112         return;
113     }
114     if (!EMAIL_PATTERN.test(email)) {
115         showError('Invalid email address.');
116         return;
117     }
118     if (message.length > 1000) {
119         showError('Message is too long (max 1000 characters).');
120         return;
121     }
122     // Build a safe element and append it. No innerHTML anywhere.
123     const entryEl = safeEntry(name, email, message);
124     feedbackList.appendChild(entryEl);
125
126     // Clear form after successful submit
127     form.reset();
128     nameInput.focus();
129 });
130
131 // Defensive: Do not allow insertion of script tags via DOM parsing from untrusted sources
132 // Avoid any uses of element.innerHTML = userInput or insertion via insertAdjacentHTML()
133 // Avoid any uses of element.innerHTML = userInput or insertion via insertAdjacentHTML()
134 })();
135 </script>
136 </body>
137 </html>

```

Output:



Observation:

1. The original AI-generated feedback form was **vulnerable to XSS** because it directly displayed user input using innerHTML.

2. The secure version fixes this by using **textContent** to safely display input, **validating user data**, and adding a **Content Security Policy (CSP)**.
3. This ensures that any malicious scripts entered by a user will **not be executed**, keeping the webpage safe.

Task 4 – Real-Time Application: Security Audit of AI-Generated

Prompt:

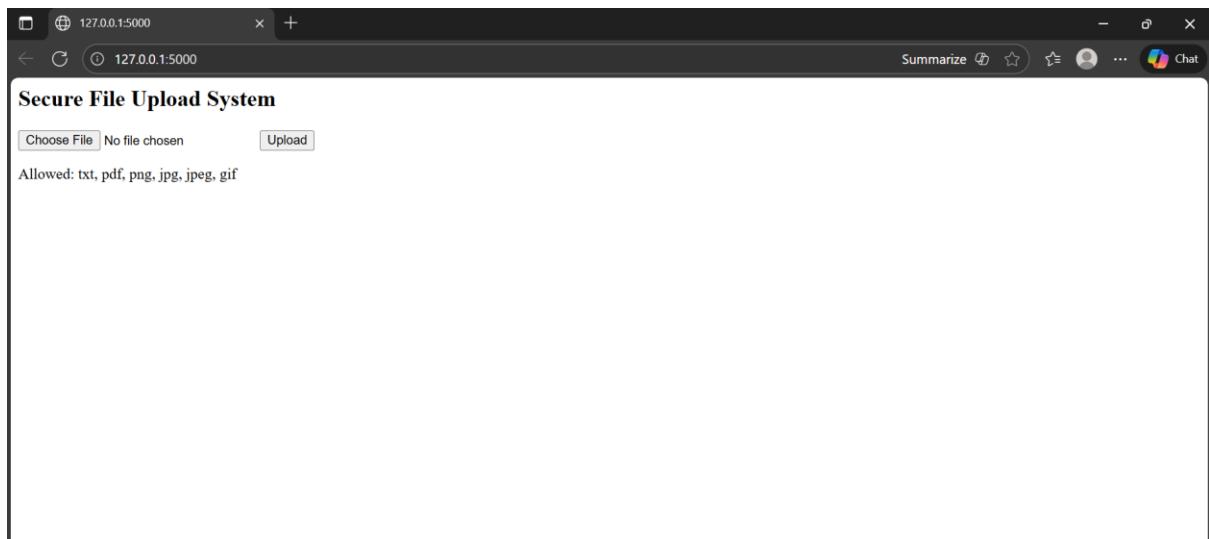
“Generate a simple Python Flask file upload program that saves uploaded files to a folder.”

Code Generated:

```
203_t4.py > ...
1   from flask import Flask, request, jsonify, render_template_string
2   import os
3   from werkzeug.utils import secure_filename
4
5   app = Flask(__name__)
6
7   # Allowed file extensions
8   ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}
9   UPLOAD_FOLDER = 'uploads'
10  os.makedirs(UPLOAD_FOLDER, exist_ok=True)
11  app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
12
13  def allowed_file(filename):
14      return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
15
16  # ✅ Home page route - this fixes the 404
17  @app.route('/')
18  def home():
19      return render_template_string('''
20          <h2>Secure File Upload System</h2>
21          <form method="POST" action="/upload" enctype="multipart/form-data">
22              <input type="file" name="file">
23              <button type="submit">Upload</button>
24          </form>
25          <p>Allowed: txt, pdf, png, jpg, jpeg, gif</p>
26      ''')
27
28  # File upload route
29  @app.route('/upload', methods=['POST'])
30  def upload_file():
31
32      if 'file' not in request.files:
33          return jsonify({'error': 'No file part in the request'}), 400
34
35      file = request.files['file']
36
37      if file.filename == '':
38          return jsonify({'error': 'No file selected'}), 400
39
40      if file and allowed_file(file.filename):
41          filename = secure_filename(file.filename)
42          file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
43          return jsonify({'message': f'File "{filename}" uploaded successfully!'}), 200
44      else:
45          return jsonify({'error': 'Invalid file type'}), 400
46
47  if __name__ == '__main__':
48      app.run()
```

Output:

```
C:\Users\likhi\OneDrive\Desktop\AIAC> python 203_t4.py
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 203_t4.py
 * Serving Flask app '203_t4'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [13/Nov/2025 18:46:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2025 18:46:22] "GET /favicon.ico HTTP/1.1" 404 -
[127.0.0.1:5000] Python ...
```



Observation:

1. The insecure version directly saved user-uploaded files without any validation, making it vulnerable to **path traversal** and **malicious file uploads**.
2. The secure version fixes these issues by using **secure_filename()**, validating file types and sizes, and disabling **debug mode**, ensuring that uploaded files are stored safely and securely.