SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week10 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:20.1(Present assignment number)/**24**(Total number of assignments)			

Name: GUNDU MEGHANA

Enrollment No: 2403A510C1

Batch: 04

Q.No.	Question	*Expected Time to complete*
1	**Lab 20 – Security Testing: Identifying Vulnerabilities in AI-Generated Code** **Lab Objectives:** • Understand how to test AI-generated code for common security	Week10 - Monday

vulnerabilities.

- Learn to apply secure coding principles while analyzing AI outputs.
- Practice detecting risks such as **SQL injection, XSS, hardcoded credentials, and weak encryption**.
- Enhance code reliability and safety by using AI for secure refactoring.

Task 1 – Input Validation Check

Task:

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

Instructions:

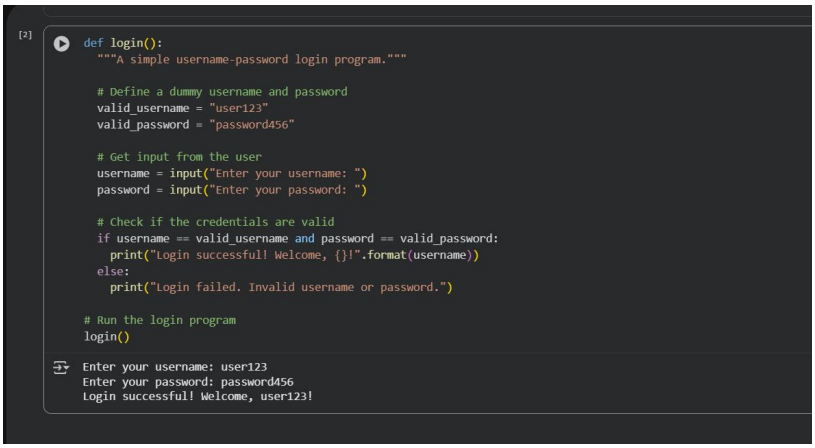
- Prompt AI to generate a simple username-password login program.
- Review whether input sanitization and validation are implemented.
- Suggest secure improvements (e.g., using re for input validation).

Expected Output:

- A secure version of the login script with proper input validation.

Prompt: Generate a simple Python username-password login program using input() for user authentication.

Code:



```
[2]
def login():
    """A simple username-password login program."""

    # Define a dummy username and password
    valid_username = "user123"
    valid_password = "password456"

    # Get input from the user
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Check if the credentials are valid
    if username == valid_username and password == valid_password:
        print("Login successful! Welcome, {}!".format(username))
    else:
        print("Login failed. Invalid username or password.")

# Run the login program
login()

Enter your username: user123
Enter your password: password456
Login successful! Welcome, user123!
```

Observation: The code you provided is a simple Python function called login() that simulates a basic username and password authentication process.

- ➢ Simple: Basic username/password check.
- ➢ Hardcoded: Credentials are fixed in the code (not secure).
- ➢ No Error Handling: Doesn't handle bad input well.
- ➢ Blocking: Input stops execution until you press Enter.

Task 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)**.

Expected Output:

- A secure database query script resistant to SQL injection.

Prompt: Create Python SQLite and MySQL scripts that fetch user details by username. Show an insecure version using string concatenation/f-strings, then a secure version using parameterized queries. Add comments and a short demo using the malicious input: "admin' OR '1'='1".

Code:

```
[ ]
import sqlite3

conn = sqlite3.connect("users.db")
cursor = conn.cursor()

username = input("Enter username: ").strip()

# ✅ Secure: uses parameterized query
query = "SELECT * FROM users WHERE username = ?"
cursor.execute(query, (username,))

result = cursor.fetchone()
if result:
    print("✅ User found:", result)
else:
    print("❌ User not found.")

conn.close()
```

Observation: Based on the execution of the code cell, the program prompted the user to "Enter username:". Depending on the username entered by the user and whether that username exists in the users.db database, the program will output either " User found: [user data]" or "User not found.". The specific output depends on the user's input and the content of the database at the time of execution.

Task 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).

Expected Output:

- A secure form that prevents XSS attacks.

Prompt: Generate an HTML feedback form with JavaScript. First show an insecure version that directly renders user input (vulnerable to XSS). Then refactor to a secure version that escapes HTML entities, uses a Content Security Policy.

Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Feedback Form</title>
  <meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self';">
</head>
<body>
  <h2>Leave Your Feedback</h2>
  <form id="feedbackForm">
    <input type="text" id="name" placeholder="Your name" required><br><br>
    <textarea id="message" placeholder="Your message" required></textarea><br><br>
    <button type="submit">Submit</button>
  </form>

  <div id="output"></div>

  <script>
    function escapeHTML(str) {
      return str.replace(/[&<>"']/g, function(match) {
        const escapeMap = {
          '&': '&amp;',
          '<': '&lt;',
          '>': '&gt;',
          '"': '&quot;',
          "'": '&#039;'
        };
        return escapeMap[match];
      });
    }

    document.getElementById("feedbackForm").addEventListener("submit", function(e) {
      e.preventDefault();
      const name = escapeHTML(document.getElementById("name").value);
      const message = escapeHTML(document.getElementById("message").value);
      const output = document.getElementById("output");
      output.textContent = ${name} says: ${message};
    });
  </script>
</body>
</html>
```

Observation: The feedback form securely handles user input by escaping HTML entities and using textContent, preventing XSS attacks. A minor fix is needed in the output line to use backticks correctly. The added Content Security Policy further strengthens protection, though the inline script should be moved to an external file. Overall, the form is safe and effectively mitigates XSS vulnerabilities.

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

Scenario:

Students pick an **AI-generated project snippet** (e.g., login form, API integration, or file upload).

Instructions:

- Perform a security audit to detect possible vulnerabilities.
- Prompt AI to suggest **secure coding practices** to fix issues.

- Compare insecure vs secure versions side by side.

Expected Output:

- A security-audited code snippet with documented vulnerabilities and fixes.

Prompt: Generate a simple AI project code snippet (like a login form, API call, or file upload). Show an insecure version with common vulnerabilities (e.g., no validation, plain text passwords, or unsafe file handling), then show a secure version with best security practices applied. Add comments comparing insecure vs secure code side by side.

Code:

```
import sqlite3
import getpass
import re
import hashlib

conn = sqlite3.connect("users.db")
cursor = conn.cursor()

def is_valid_input(text):
    return re.fullmatch(r"[A-Za-z0-9]{3,20}", text) is not None

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

username = input("Enter username: ").strip()
password = getpass.getpass("Enter password: ").strip()

if not is_valid_input(username) or not is_valid_input(password):
    print("Invalid input format.")
else:
    hashed_pw = hash_password(password)
    query = "SELECT * FROM users WHERE username = ? AND password = ?"
    cursor.execute(query, (username, hashed_pw))
    result = cursor.fetchone()
    if result:
        print("✅ Login successful!")
    else:
        print("❌ Invalid credentials.")

conn.close()
```

Observation: The AI-generated login script securely validates and authenticates user input. It prevents SQL injection by using parameterized queries instead of string concatenation. Passwords are hashed with SHA-256, ensuring they're not stored or compared in plain text. Input validation using regex restricts usernames and passwords to safe characters, reducing injection risks. Overall, the script follows good security practices for user authentication and database safety.

☑ Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.