TASK =3

Secure Coding Review Report

1. Selected Language and Application

Language: Python

Application: Simple User Login API (using Flask framework)

2. Code Review to Identify Security Vulnerabilities

Vulnerability	Location (File / Line No)	Risk Level	Description
SQL Injection	app.py → Line 25	High	Using string concatenation to build SQL query instead of parameterized query
Hardcoded Credentials	config.py → Line 5	Medium	Database username and password are hardcoded in the source code
Missing Input Validation	app.py → Line 18	Medium	No input validation or sanitization on user input (username, password)
Insecure Error Message	app.py → Line 30	Low	Detailed error messages are returned to the client (could leak sensitive info)

3. Tools Used for Review

• Manual Code Inspection

• Static Analysis Tool: Bandit (for Python security analysis)

Command used:
bandit -r .

4. Recommendations and Best Practices

Vulnerability	Fix Recommendation
SQL Injection	Use parameterized queries with placeholders instead of string concatenation
Hardcoded Credentials	Move sensitive credentials to environment variables (.env) or use config files outside source control
Missing Input Validation	Always validate and sanitize all user inputs using libraries like WTForms or Cerberus
Insecure Error Message	Show generic error messages to users, and log detailed errors internally

5. Remediation Steps for Safer Code

Example Code Fixes:

Before (SQL Injection prone)

```
query = "SELECT * FROM users WHERE username = '" + username + "'
AND password = '" + password + """
```

After (Safe with parameterized query)

```
cursor.execute("SELECT * FROM users WHERE username = %s AND
password = %s", (username, password))
```

Before (Hardcoded credentials)

```
DB_USERNAME = "admin"
DB_PASSWORD = "password123"
```

After (Using Environment Variables)

```
import os
DB_USERNAME = os.getenv("DB_USERNAME")
DB_PASSWORD = os.getenv("DB_PASSWORD")
```

Before (No input validation)

```
username = request.form['username']
```

After (With validation)

```
from wtforms import Form, StringField, validators

class LoginForm(Form):
  username = StringField('Username', [validators.InputRequired(),
validators.Length(min=3, max=25)])
```

Before (Insecure error message)

```
except Exception as e:
  return str(e)
```

After (Safe error handling)

```
except Exception as e:
  app.logger.error(f"Error: {e}")
  return "Internal Server Error", 500
```

---

6. Summary

• Total Vulnerabilities Found: 4

• Fix Recommendations Provided

• Sample Secure Coding Best Practices Followed

• Remediation Code Samples Attached

Conclusion: After applying the above remediation steps, the application will become safer against common web attacks like SQL Injection, Credential leakage, and Input Validation issues.