Example: Secure Coding Review of a Python Web Application

1. Selected Application

- **Language**: Python
- **Framework**: Flask

2. Code

```
from flask import Flask, request

import sqlite3


app = Flask(__name__)


@app.route('/login', methods=['POST'])

def login():

    username = request.form['username']

    password = request.form['password']


    conn = sqlite3.connect('users.db')

    cursor = conn.cursor()

    cursor.execute(f"SELECT * FROM users WHERE username='{username}' AND password='{password}'")

    user = cursor.fetchone()

    conn.close()


    if user:

        return "Login successful"

    else:

        return "Invalid credentials"


if __name__ == '__main__':

    app.run(debug=True)
```

Code Review Findings

- **Vulnerability 1**: **SQL Injection Risk**
    - **Issue**: The application directly interpolates user inputs (`username` and `password`) into the SQL query.
    - **Impact**: This could allow an attacker to execute arbitrary SQL commands.
- **Vulnerability 2**: **Insecure Password Storage**
    - **Issue**: Passwords are stored in plain text in the database.
    - **Impact**: If the database is compromised, all user passwords are exposed.

4. Recommendations

- **For SQL Injection**:
    - Use parameterized queries to prevent SQL injection.
    - **Refactored Code Example**

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

For Password Storage:

- Implement password hashing using libraries like `bcrypt`.
- **Refactored Code Example** (for storing passwords)

```
from werkzeug.security import generate_password_hash, check_password_hash


hashed_password = generate_password_hash(password)

# Store `hashed_password` in the database instead of plain text.
```

5. Document Findings and Remediation Steps

- **Findings Report**:
    - **SQL Injection Risk**: Change the SQL query to use parameterized statements.
    - **Insecure Password Storage**: Use hashed passwords and implement secure authentication practices.
- **Remediation Steps**:
    1. Update the SQL query in the `login` function.
    2. Implement password hashing when creating user accounts.
    3. Conduct regular code reviews to identify similar issues in the future.

Conclusion

This example illustrates how to conduct a secure coding review, identify vulnerabilities, and provide actionable recommendations. If you need further details or additional examples, feel free to ask!