Secure Coding Review Report

1. Programming Language and Application

Language: Python

Application: Simple login script using SQLite.

Purpose: To check login credentials against a local database.

2. Code Review: Original Insecure Code

```
import sqlite3


def login(username, password):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
    cursor.execute(query)
    result = cursor.fetchone()
    if result:
        print("Login successful!")
    else:
        print("Invalid credentials")
    conn.close()
```

3. Identified Vulnerabilities

- SQL Injection: User input directly in SQL query.
- Plaintext Passwords: No hashing used.
- No Input Validation: Risk of injection or crashes.
- No Logging: No trace of login activity.

4. Tools Used

Manual Code Review

Static Analyzer Suggested: Bandit (for Python)

Secure Coding Review Report

5. Recommendations and Best Practices

- Use parameterized SQL queries to prevent injection.
- Store hashed passwords using bcrypt.
- Validate all input to ensure safe processing.
- Use logging to track login attempts and failures.

6. Secure Version of Code

```
import sqlite3
import bcrypt
import logging

logging.basicConfig(filename='login.log', level=logging.INFO)

def login(username, password):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute("SELECT password FROM users WHERE username = ?", (username,))
    row = cursor.fetchone()

    if row and bcrypt.checkpw(password.encode(), row[0].encode()):
        logging.info(f"User {username} logged in successfully.")
        print("Login successful!")
    else:
        logging.warning(f"Failed login attempt for user {username}.")
        print("Invalid credentials")
    conn.close()
```

7. Remediation Steps

1. Updated the login script to use parameterized queries.
2. Added bcrypt for secure password hashing.

3. Enabled logging for security audit.

4. Ensured input is processed safely.