### Programming Language: Java

Live test on this sites

<https://owasp.org/SecureCodingDojo/codereview101/>

**Application Scenario:** A web-based user authentication system that handles user input, database queries, and sensitive data transmission.

### Code Review Findings and Recommendations

#### 1. Input Validation

**Observation:** The application accepts user input for login forms and other features without adequate validation, leading to potential injection attacks.

**Code Example:**

**Vulnerable Code:**

String username = request.getParameter("username");

String password = request.getParameter("password");

String query = "SELECT \* FROM users WHERE username = '" + username + "' AND password = '" + password + "'";

ResultSet rs = stmt.executeQuery(query);

**Secure Code:**

String query = "SELECT \* FROM users WHERE username = ? AND password = ?";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, username);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

**Recommendation:** Use parameterized queries to prevent SQL injection and validate user input using whitelisting.

#### 2. Data Encryption

**Observation:** Passwords are stored in plain text in the database, posing a significant security risk.

**Code Example:**

**Vulnerable Code:**

String insertQuery = "INSERT INTO users (username, password) VALUES ('" + username + "', '" + password + "')";

stmt.executeUpdate(insertQuery);

**Secure Code:**

String hashedPassword = BCrypt.hashpw(password, BCrypt.gensalt());

String insertQuery = "INSERT INTO users (username, password) VALUES (?, ?)";

PreparedStatement stmt = conn.prepareStatement(insertQuery);

stmt.setString(1, username);

stmt.setString(2, hashedPassword);

stmt.executeUpdate();

**Recommendation:** Always hash passwords using a secure algorithm like BCrypt before storing them in the database.

#### 3. Secure Transmission

**Observation:** Sensitive data is transmitted over HTTP, exposing it to interception.

**Code Example:**

**Vulnerable Code:**

URL url = new URL("http://example.com/login?username=" + username + "&password=" + password);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("GET");

**Secure Code:**

URL url = new URL("https://example.com/login");

HttpsURLConnection con = (HttpsURLConnection) url.openConnection();

con.setRequestMethod("POST");

con.setDoOutput(true);

String urlParams = "username=" + URLEncoder.encode(username, "UTF-8") + "&password=" + URLEncoder.encode(password, "UTF-8");

con.getOutputStream().write(urlParams.getBytes("UTF-8"));

**Recommendation:** Enforce HTTPS for all sensitive data transmissions.

#### 4. Cross-Site Scripting (XSS) Prevention

**Observation:** User-generated content is rendered on web pages without proper sanitization.

**Code Example:**

**Vulnerable Code:**

out.println("<p>" + request.getParameter("comment") + "</p>");

**Secure Code:**

out.println("<p>" + StringEscapeUtils.escapeHtml4(request.getParameter("comment")) + "</p>");

**Recommendation:** Escape all user input before rendering it on web pages to prevent XSS attacks.

### Tools Used

**Static Code Analyzers:**

SonarQube

Checkmar

**Manual Code Review:**

Analyzed critical areas such as input validation, authentication, and data handling.