SAST & SCA Report

Repository: pnkjshahare/AICodeReview
Pull Request: #42
Generated: 2025-12-09 11:13 UTC

▯ SCA - Dependency Vulnerabilities

No dependency files found. SCA scan skipped.

▯ SAST - Code Vulnerabilities

snyk

December 9th 2025, 11:13:23 am (UTC+00:00)
Source: C:\Users\panka\AppData\Local\Temp\sast_a5eaf372-5eaf-4883-8abd-3dec2311bc1e

Snyk Code Report

H **0** high issues   M **0** medium issues   L **7** low issues

SCAN COVERAGE

.cpp files: **1**   .java files: **2**

L Use of Hardcoded Credentials

Fix Analysis

SNYK-CODE | CWE-798 | NoHardcodedCredentials/test

Do not hardcode credentials in code.

Found in: **SecurityTest.java (line : 11)**

Data Flow

```
11:47
private static final String DB_USERNAME = "admin";
SOURCE SINK 0
```

L Use of Hardcoded Credentials

Fix Analysis

SNYK-CODE | CWE-798 | NoHardcodedCredentials/test

Do not hardcode credentials in code.

Found in: **SecurityTest.java (line : 17)**

Data Flow

```
17:17
"jdbc:mysql://localhost:3306/test", DB USERNAME, DB PASSWORD);
SOURCE SINK 0
```

L Use of Hardcoded Passwords

Fix Analysis

SNYK-CODE | CWE-798,CWE-259 | HardcodedPassword/test

Do not hardcode passwords in code. Found hardcoded password used in here.

Found in: **SecurityTest.java (line : 12)**

Data Flow

SecurityTest.java

```
12:47
private static final String DB PASSWORD = "123456"; // insecure hardcoded password
SOURCE SINK 0
```

L Hardcoded Secret

Fix Analysis

SNYK-CODE | CWE-547 | HardcodedSecret/test

Hardcoded value string is used as a cipher key. Generate the value with a cryptographically strong random number generator such as java.security.SecureRandom instead.

Found in: **SecurityTest.java (line : 45)**

Data Flow

```
44:22
String key = "1234567812345678"; // hardcoded key
SOURCE 0

45:38
SecretKeySpec skeySpec = new SecretKeySpec( key.getBytes(), "AES");
```

SINK 1

L Use of Cipher Without Integrity Protection

Fix Analysis

SNYK-CODE | CWE-327 | InsecureCipherNoIntegrity/test

> The ECB mode used in javax.crypto.Cipher.getInstance does not provide integrity. Consided using Galois/Counter Mode.
>
> Found in: **SecurityTest.java (line : 47)**

Data Flow

```
47:44
Cipher cipher = Cipher.getInstance( "AES") ;
SOURCE 0

47:25
Cipher cipher = Cipher.getInstance( "AES");
SINK 1
```

L Use of Insecure Default AES Cipher

Fix Analysis

SNYK-CODE | CWE-327 | InsecureDefaultAesCipher/test

> Default AES/ECB algorithm (AES) used in javax.crypto.Cipher.getInstance may be insecure, because equal messages get encrypted to equal data. Consider using Galois/Counter Mode (algorithm AES/GCM/NoPadding).
>
> Found in: **SecurityTest.java (line : 47)**

Data Flow

```
47:44
Cipher cipher = Cipher.getInstance( "AES") ;
SOURCE 0

47:25
Cipher cipher = Cipher.getInstance( "AES");
SINK 1
```

L Use of Password Hash With Insufficient Computational Effort

Fix Analysis

SNYK-CODE | CWE-916 | InsecureHash/test

The MD5 hash (used in java.security.MessageDigest.getInstance) is insecure. Consider changing it to a secure hash algorithm

Found in: **SecurityTest.java (line : 38)**

Data Flow

```
38:54
MessageDigest md = MessageDigest.getInstance( "MD5") ; // weak hash
SOURCE 0

38:28
MessageDigest md = MessageDigest.getInstance( "MD5"); // weak hash
SINK 1
```