




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 **Java**


 JavaScript


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6

 XML



Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules 632

Vulnerability 53

Bug 154

Security Hotspot 36

Code Smell 389

Quick Fix 42

Tags ▾

Search by name... 🔍

Zero should not be a possible denominator

Bug

Locks should be released

Bug

"runFinalizersOnExit" should not be called

Bug

"ScheduledThreadPoolExecutor" should not have 0 core threads

Bug

"Random" objects should be reused

Bug

The signature of "finalize()" should match that of "Object.finalize()"

Bug

Jump statements should not occur in "finally" blocks

Bug

"super.finalize()" should be called at the end of "Object.finalize()" implementations

Bug

Using slow regular expressions is security-sensitive

Security Hotspot

Using publicly writable directories is security-sensitive

Security Hotspot

Using clear-text protocols is security-sensitive

Security Hotspot

Accessing Android external storage is security-sensitive

JWT should be signed and verified with strong cipher algorithms

Analyze your code

Vulnerability Critical cwe privacy cert owasp

If a JSON Web Token (JWT) is not signed with a strong cipher algorithm (or not signed at all) an attacker can forge it and impersonate user identities.

- Don't use none algorithm to sign or verify the validity of a token.
- Don't use a token without verifying its signature before.

Noncompliant Code Example

Using `jwtk/Java JWT` library (to verify a signed token (containing a JWS) don't use the `parse` method as it doesn't throw an exception if an unsigned token is provided):

```
// Signing:
io.jsonwebtoken.Jwts.builder() // Noncompliant, token is not
    .setSubject(USER_LOGIN)
    .compact();
// Verifying:
io.jsonwebtoken.Jwts.parser().setSigningKey(SECRET_KEY).pars
```

Using `auth0/Java JWT` library:

```
// Signing:
com.auth0.jwt.JWT.create()
    .withSubject(SUBJECT)
    .sign(Algorithm.none()); // Noncompliant, use only strong
// Verifying:
JWTVerifier nonCompliantVerifier = com.auth0.jwt.JWT.require
    .withSubject(LOGIN)
    .build();
```

Compliant Solution

Using `Java JWT` library (to verify a signed token (containing a JWS) use the `parseClaimsJws` method that will throw an exception if an unsigned token is provided):





```
// Signing:
Jwts.builder() // Compliant
    .setSubject(USER_LOGIN)
    .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
    .compact();
// Verifying:
Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token)
```

Using `auth0/Java JWT` library. I

```
// Signing:
JWT.create()
```

https://rules.sonarsource.com/java/RSPEC-5659

1/2

 Security Hotspot
Receiving intents is security-sensitive
 Security Hotspot
Broadcasting intents is security-sensitive
 Security Hotspot
Expanding archive files without controlling resource consumption is security-sensitive
 Security Hotspot
Configuring loggers is security-sensitive

```
.withSubject(SUBJECT)
.sign(Algorithm.HMAC256(SECRET_KEY)); // Compliant
// Verifying:
JWTVerifier nonCompliantVerifier = JWT.require(Algorithm.HMA
.withSubject(LOGIN)
.build();
```

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [MITRE, CWE-347](#) - Improper Verification of Cryptographic Signature

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)