

































-  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 rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags

Search by name...

Exceptions should not be thrown in finally blocks

Code Smell

Constant names should comply with a naming convention

Code Smell

The Object.finalize() method should not be overridden

Code Smell

XML operations should not be vulnerable to injection attacks

Vulnerability

JSON operations should not be vulnerable to injection attacks

Vulnerability

XML signatures should be validated securely

Vulnerability

XML parsers should not be vulnerable to Denial of Service attacks

Vulnerability

XML parsers should not load external schemas

Vulnerability

Mobile database encryption keys should not be disclosed

Vulnerability

Reflection should not be vulnerable to injection attacks

Vulnerability

Authorizations should be based on strong decisions

Vulnerability

OpenSAML2 should be configured to prevent authentication bypass

Assertion methods should not be used within the try block of a try-catch catching an Error

Analyze your code

Bug

Critical

junit tests

Assertion methods are throwing a "java.lang.AssertionError". If this call is done within the try block of a try-catch cathing a similar error, you should make sure to test some properties of the exception. Otherwise, the assertion will never fail.

Noncompliant Code Example

```
@Test
public void should_throw_assertion_error() {
    try {
        throwAssertionError();
        Assert.fail("Expected an AssertionError!"); // Noncompliant
    } catch (AssertionError e) {}
}

private void throwAssertionError() {
    throw new AssertionError("My assertion error");
}
```

Compliant Solution

```
assertThrows(AssertionError.class, () -> throwAssertionError());

try {
    throwAssertionError();
    Assert.fail("Expected an AssertionError!"); // Compliant,
} catch (AssertionError e) {
    Assert.assertThat(e.getMessage(), is("My assertion error"))
}
```

See

- [JUnit 4 exception testing documentation](#)

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](#)

https://rules.sonarsource.com/java/RSPEC-5779

1/2

 Vulnerability
<div>Server-side requests should not be vulnerable to forging attacks</div> <div> Vulnerability</div>
<div>Collections should not be modified while they are iterated</div> <div> Bug</div>
<div>Equals method should be overridden in records containing array fields</div> <div> Bug</div>
<div>Reflection should not be used to increase accessibility of records'</div> <div> Bug</div>