




 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... 🔍

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

Vulnerability

Server-side requests should not be vulnerable to forging attacks

Vulnerability

Collections should not be modified while they are iterated

Bug

Equals method should be overridden in records containing array fields

Zero should not be a possible denominator

Analyze your code

Bug

Critical

cwe denial-of-service cert

If the denominator to a division or modulo operation is zero it would result in a fatal error.

When working with double or float, no fatal error will be raised, but it will lead to unusual result and should be avoided anyway.

This rule supports primitive int, long, double, float as well as BigDecimal and BigInteger.

Noncompliant Code Example

```
void test_divide() {
    int z = 0;
    if (unknown()) {
        // ..
        z = 3;
    } else {
        // ..
    }
    z = 1 / z; // Noncompliant, possible division by zero
}
```

Compliant Solution

```
void test_divide() {
    int z = 0;
    if (unknown()) {
        // ..
        z = 3;
    } else {
        // ..
        z = 1;
    }
    z = 1 / z;
}
```

See





- MITRE, CWE-369 - Divide by zero
- CERT, NUM02-J. - Ensure that division and remainder operations do not result in divide-by-zero errors
- CERT, INT33-C. - Ensure that division and remainder operations do not result in divide-by-zero errors

Available In:

sonarlint | sonarcloud | sonarqube

https://rules.sonarsource.com/java/RSPEC-3518

1/2

 Bug
Reflection should not be used to increase accessibility of records' fields
 Bug
AssertJ assertions with "Consumer" arguments should contain assertion inside consumers
 Bug
The regex escape sequence \cX should only be used with characters in the @-_ range
 Bug

© 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](#)