




 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... 🔍

be replaced with Text Blocks

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

Constructors of an "abstract" class should not be declared "public"

Code Smell

Similar tests should be grouped in a single Parameterized test

Code Smell

Tests should be stable

Code Smell

Test methods should not contain too many assertions

Code Smell

AssertJ "assertThatThrownBy" should not be used alone

Code Smell

Character classes in regular expressions should not contain the same character twice

Code Smell

Names of regular expressions named groups should be used

Code Smell

Regexes containing characters subject to normalization should use the CANON_EQ flag

Code Smell

Reflection should not be vulnerable to injection attacks

Analyze your code

VulnerabilityMajor?injection cwe owasp

User-provided data, such as URL parameters, should always be considered untrusted and tainted. Constructing class or method names directly from tainted data enables attackers to inject specially crafted values that could result in unexpected behavior, e.g. crash of the application.

Typically, the solution is to validate every parameter used to create the name. This can be achieved by validating them against a list of authorized values.

Noncompliant Code Example

```
public void run(javax.servlet.http.HttpServletRequest request) {
    String name = request.getParameter("name");
    Class clazz = Class.forName(name); // Noncompliant
}
```

Compliant Solution

```
public void run(javax.servlet.http.HttpServletRequest request) {
    String name = request.getParameter("name");
    if (this.allowed.contains(name)) {
        Class clazz = Class.forName(name);
    }
}
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-470](#) - Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

Available In:

sonarcloud

sonarqube




Developer Edition

© 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-6173

1/2

<div>Regular expressions should not be too complicated</div> <div> Code Smell</div>
<div>JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion</div> <div> Code Smell</div>
<div>Only one method invocation is expected when testing runtime exceptions</div> <div> Code Smell</div>
<div>Exception testing via JUnit ExpectedException rule should not be mixed with other assertions</div>