sonar

RULES

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

## 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...

Extremely, should not be used in ascending 'for' loops

Code Smell

Collection constructors should not be used as java.util.function.Function

Code Smell

"else" statements should be clearly matched with an "if"

Code Smell

"Class.forName()" should not load JDBC 4.0+ drivers

Code Smell

Java features should be preferred to Guava

Code Smell

Nullness of parameters should be guaranteed

Code Smell

"Integer.toHexString" should not be used to build hexadecimal strings

Code Smell

Asserts should not be used to check the parameters of a public method

Code Smell

Assignments should not be redundant

Code Smell

Methods should not have identical implementations

Code Smell

"java.nio.Files#delete" should be preferred

Code Smell

Unused "private" classes should be removed

Code Smell

### Regular expressions should not overflow the stack

Analyze your code

Bug

Major

regex

The Java regex engine uses recursive method calls to implement backtracking. Therefore when a repetition inside a regular expression contains multiple paths (i.e. the body of the repetition contains an alternation (|), an optional element or another repetition), trying to match the regular expression can cause a stack overflow on large inputs. This does not happen when using a possessive quantifier (such as ++ instead of \*) or when using a character class inside a repetition (e.g. [ab]++ instead of (a|b)\*).

The size of the input required to overflow the stack depends on various factors, including of course the stack size of the JVM. One thing that significantly increases the size of the input that can be processed is if each iteration of the repetition goes through a chain of multiple constant characters because such consecutive characters will be matched by the regex engine without invoking any recursion.

For example, on a JVM with a stack size of 1MB, the regex (? : a | b ) \* will overflow the stack after matching around 6000 characters (actual numbers may differ between JVM versions and even across multiple runs on the same JVM) whereas (? : abc | def ) \* can handle around 15000 characters.

Since often times stack growth can't easily be avoided, this rule will only report issues on regular expressions if they can cause a stack overflow on realistically sized inputs. You can adjust the maxStackConsumptionFactor parameter to adjust this.

#### Noncompliant Code Example

```
Pattern.compile("(a|b)*"); // Noncompliant
Pattern.compile("(.|\\n)*"); // Noncompliant
Pattern.compile("(ab?)*"); // Noncompliant
```

#### Compliant Solution

```
Pattern.compile("[ab]*"); // Character classes don't cause r
Pattern.compile("(?s).*"); // Enabling the (?s) flag makes '
Pattern.compile("(ab?)+"); // Possessive quantifiers don't
```

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-5998

1/2

<p><b>"Stream.peek" should be used with caution</b></p> <p> Code Smell</p>	
<p><b>"Map.get" and value test should be replaced with single method call</b></p> <p> Code Smell</p>	
<p><b>"@RequestMapping" methods should not be "private"</b></p> <p> Code Smell</p>	
<p><b>Raw types should not be used</b></p> <p> Code Smell</p>	