sonar RULES

Products ⌄

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...

Code Smell

Unused "private" fields should be removed
Code Smell

Collapsible "if" statements should be merged
Code Smell

Unused labels should be removed
Code Smell

Standard outputs should not be used directly to log anything
Code Smell

OS commands should not be vulnerable to argument injection attacks
Vulnerability

"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization
Vulnerability

Logging should not be vulnerable to injection attacks
Vulnerability

Exceptions should not be thrown from servlet methods
Vulnerability

Return values should not be ignored when they contain the operation status code
Bug

Repeated patterns in regular expressions should not match the empty string
Bug

AssertJ assertions "allMatch" and "doesNotContains" should also test

Blocks should be synchronized on "private final" fields

Analyze your code

Bug Major cwe multi-threading cert

Synchronizing on a class field synchronizes not on the field itself, but on the object assigned to it. So synchronizing on a non-`final` field makes it possible for the field's value to change while a thread is in a block synchronized on the old value. That would allow a second thread, synchronized on the new value, to enter the block at the same time.

The story is very similar for synchronizing on parameters; two different threads running the method in parallel could pass two different object instances in to the method as parameters, completely undermining the synchronization.

Noncompliant Code Example

```
private String color = "red";

private void doSomething(){
  synchronized(color) {  // Noncompliant; lock is actually o
    //...
    color = "green"; // other threads now allowed into this
    // ...
  }
  synchronized(new Object()) { // Noncompliant this is a no-
     // ...
  }
}
```

Compliant Solution

```
private String color = "red";
private final Object lockObj = new Object();

private void doSomething(){
  synchronized(lockObj) {
    //...
    color = "green";
    // ...
  }
}
```

See

- MITRE, CWE-412 - Unrestricted Externally Accessible Lock
- MITRE, CWE-413 - Improper Resource Locking
- CERT, LCK00-J. - Use private final lock objects to synchronize classes that may interact with untrusted code

Available In:

sonarlint sonarcloud sonarqube

for emptiness

Bug

Double Brace Initialization should not be used

Bug

Non-primitive fields should not be "volatile"

Bug

"toArray" should be passed an array of the proper type

Bug