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...

Wildcard imports should not be used
Code Smell

Modulus results should not be checked for direct equality
Code Smell

Comparators should be "Serializable"
Code Smell

"Serializable" classes should have a "serialVersionUID"
Code Smell

"switch" statements and expressions should not be nested
Code Smell

Constructors should only call non-overridable methods
Code Smell

Methods should not be too complex
Code Smell

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Code Smell

"if ... else if" constructs should end with "else" clauses
Code Smell

Control structures should use curly braces
Code Smell

Expressions should not be too complex
Code Smell

Mockito argument matchers should be used on all parameters

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

Analyze your code

Code Smell Major cwe

Using `Integer.toHexString` is a common mistake when converting sequences of bytes into hexadecimal string representations. The problem is that the method trims leading zeroes, which can lead to wrong conversions. For instance a two bytes value of `0x4508` would be converted into `45` and `8` which once concatenated would give `0x458`.

This is particularly damaging when converting hash-codes and could lead to a security vulnerability.

This rule raises an issue when `Integer.toHexString` is used in any kind of string concatenations.

Noncompliant Code Example

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] bytes = md.digest(password.getBytes("UTF-8"));

StringBuilder sb = new StringBuilder();
for (byte b : bytes) {
    sb.append(Integer.toHexString( b & 0xFF )); // Noncompli
}
```

Compliant Solution

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] bytes = md.digest(password.getBytes("UTF-8"));

StringBuilder sb = new StringBuilder();
for (byte b : bytes) {
    sb.append(String.format("%02X", b));
}
```

See

- MITRE, CWE-704 - Incorrect Type Conversion or Cast
- Derived from FindSecBugs rule BAD_HEXA_CONVERSION

Available In:
sonarlint sonarcloud sonarqube

Bug

Spring "@Controller" classes should not use "@Scope"

Bug

Constructor injection should be used instead of field injection

Bug

Classes that don't define "hashCode()" should not be used in hashes

Bug

Floating point numbers should not be tested for equality