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...

Bug

"Iterator.hasNext()" should not call "Iterator.next()"
Bug

Identical expressions should not be used on both sides of a binary operator
Bug

Loops with at most one iteration should be refactored
Bug

Variables should not be self-assigned
Bug

"StringBuilder" and "StringBuffer" should not be instantiated with a character
Bug

Methods should not be named "tostring", "hashcode" or "equal"
Bug

"Thread.run()" should not be called directly
Bug

"equals" method overrides should accept "Object" parameters
Bug

The Object.finalize() method should not be called
Bug

Enabling file access for WebViews is security-sensitive
Security Hotspot

Enabling JavaScript support for WebViews is security-sensitive

Fields in a "Serializable" class should either be transient or serializable

Analyze your code

Code Smell Critical cwe serialization

Fields in a `Serializable` class must themselves be either `Serializable` or `transient` even if the class is never explicitly serialized or deserialized. For instance, under load, most J2EE application frameworks flush objects to disk, and an allegedly `Serializable` object with non-transient, non-serializable data members could cause program crashes, and open the door to attackers. In general a `Serializable` class is expected to fulfil its contract and not have an unexpected behaviour when an instance is serialized.

This rule raises an issue on non-`Serializable` fields, and on collection fields when they are not `private` (because they could be assigned non-`Serializable` values externally), and when they are assigned non-`Serializable` types within the class.

Noncompliant Code Example

```
public class Address {
  //...
}

public class Person implements Serializable {
  private static final long serialVersionUID = 1905122041950

  private String name;
  private Address address;  // Noncompliant; Address isn't s
}
```

Compliant Solution

```
public class Address implements Serializable {
  private static final long serialVersionUID = 2405172041950
}

public class Person implements Serializable {
  private static final long serialVersionUID = 1905122041950

  private String name;
  private Address address;
}
```

Exceptions

The alternative to making all members `serializable` or `transient` is to implement special methods which take on the responsibility of properly serializing and de-serializing the object. This rule ignores classes which implement the following methods:

Security Hotspot

Constructing arguments of system commands from user input is security-sensitive

Security Hotspot

Using unencrypted files in mobile applications is security-sensitive

Security Hotspot

Using biometric authentication without a cryptographic solution is security-sensitive

Security Hotspot

Using unencrypted databases in

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

See

- MITRE, CWE-594 - Saving Unserializable Objects to Disk
- Oracle Java 6, Serializable
- Oracle Java 7, Serializable

Available In:

sonarlint sonarcloud sonarqube