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...

Security Hotspot

Enabling JavaScript support for WebViews is security-sensitive

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 mobile applications is security-sensitive

Security Hotspot

Authorizing non-authenticated users to use keys in the Android KeyStore is security-sensitive

Security Hotspot

Allowing user enumeration is security-sensitive

Security Hotspot

Allowing requests with excessive content length is security-sensitive

Security Hotspot

Disabling auto-escaping in template engines is security-sensitive

Security Hotspot

Allowing deserialization of LDAP objects is security-sensitive

Security Hotspot

Setting loose POSIX file permissions

Constants should not be defined in interfaces

Analyze your code

Code Smell Critical bad-practice

According to Joshua Bloch, author of "Effective Java":

> The constant interface pattern is a poor use of interfaces.
> That a class uses some constants internally is an implementation detail.
> Implementing a constant interface causes this implementation detail to leak into the class's exported API. It is of no consequence to the users of a class that the class implements a constant interface. In fact, it may even confuse them. Worse, it represents a commitment: if in a future release the class is modified so that it no longer needs to use the constants, it still must implement the interface to ensure binary compatibility. If a nonfinal class implements a constant interface,
> all of its subclasses will have their namespaces polluted by the constants in the interface.

This rule raises an issue when an interface consists solely of fields, without any other members.

Noncompliant Code Example

```
interface Status {                      // Noncompliant
   int OPEN = 1;
   int CLOSED = 2;
}
```

Compliant Solution

```
public enum Status {                    // Compliant
  OPEN,
  CLOSED;
}
```

or

```
public final class Status {             // Compliant
   public static final int OPEN = 1;
   public static final int CLOSED = 2;
}
```

Available In:

sonarlint sonarcloud sonarqube

is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Deprecated annotations should include explanations

Code Smell

Restricted Identifiers should not be used as Identifiers

Code Smell

Redundant constructors/methods