




 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... 🔍

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 is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

String literals should not be duplicated

Analyze your code

Code Smell

Critical

design

Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences.

On the other hand, constants can be referenced from many places, but only need to be updated in a single place.

Noncompliant Code Example

With the default threshold of 3:

```
public void run() {
    prepare("action1");
    execute("action1");
    release("action1");
}

// Noncompliant

@SuppressWarning("all")
private void method1() { /* ... */ }
@SuppressWarning("all")
private void method2() { /* ... */ }

// Compliant

public String method3(String a) {
    System.out.println("'" + a + "'");
    return "";
}

// Compliant
```

Compliant Solution

```
private static final String ACTION_1 = "action1";

public void run() {
    prepare(ACTION_1);
    execute(ACTION_1);
    release(ACTION_1);
}





// Compliant
```

Exceptions

To prevent generating some false-positives, literals having less than 5 characters are excluded.

Available In:

sonarlint | sonarcloud | sonarqube

 Security Hotspot
<b>Deprecated annotations should include explanations</b>  Code Smell
<b>Restricted Identifiers should not be used as Identifiers</b>  Code Smell
<b>Redundant constructors/methods should be avoided in records</b>  Code Smell
<b>Records should be used instead of ordinary classes when representing immutable data structure</b>

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](#)