




 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... 🔍

Abstract class names should comply with a naming convention

Code Smell

Strings literals should be placed on the left side when checking for equality

Code Smell

Files should contain an empty newline at the end

Code Smell

Source code should be indented consistently

Code Smell

A close curly brace should be located at the beginning of a line

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line

Code Smell

An open curly brace should be located at the beginning of a line

Code Smell

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Functions should not be defined with a variable number of arguments

Code Smell

Allowing both safe and unsafe HTTP methods is security-sensitive

Analyze your code

Security Hotspot

Minor

cwe spring sans-top25 owasp

An HTTP method is safe when used to perform a read-only operation, such as retrieving information. In contrast, an unsafe HTTP method is used to change the state of an application, for instance to update a user's profile on a web application.

Common safe HTTP methods are GET, HEAD, or OPTIONS.

Common unsafe HTTP methods are POST, PUT and DELETE.

Allowing both safe and unsafe HTTP methods to perform a specific operation on a web application could impact its security, for example CSRF protections are most of the time only protecting operations performed by unsafe HTTP methods.

Ask Yourself Whether

- HTTP methods are not defined at all for a route/controller of the application.
- Safe HTTP methods are defined and used for a route/controller that can change the state of an application.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

For all the routes/controllers of an application, the authorized HTTP methods should be explicitly defined and safe HTTP methods should only be used to perform read-only operations.

Sensitive Code Example

```
@RequestMapping("/delete_user") // Sensitive: by default all
public String deletel(String username) {
    // state of the application will be changed here
}

@RequestMapping(path = "/delete_user", method = {RequestMethod.
String delete2(@RequestParam("id") String id) {
    // state of the application will be changed here
}
```

Compliant Solution

```
@RequestMapping("/delete_user", method = RequestMethod.POST)
public String deletel(String username) {
    // state of the application will be changed here
}

@RequestMapping(path = "/delete_user", method = RequestMethod.
String delete2(@RequestParam("id") String id) {
    // state of the application will be changed here
}
```

https://rules.sonarsource.com/java/RSPEC-3752

1/2

Local-Variable Type Inference should be used Code Smell**Migrate your tests from JUnit4 to the new JUnit5 annotations** Code Smell**Track uses of disallowed classes** Code Smell**Track uses of "@SuppressWarnings" annotations** Code Smell**See**

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2021 Category A4](#) - Insecure Design
- [OWASP Top 10 2017 Category A5](#) - Broken Access Control
- [MITRE, CWE-352](#) - Cross-Site Request Forgery (CSRF)
- [OWASP: Cross-Site Request Forgery](#)
- [SANS Top 25](#) - Insecure Interaction Between Components
- [Spring Security Official Documentation: Use proper HTTP verbs \(CSRF protection\)](#)

Available In:

sonarcloud  **sonarqube** 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, 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](#)