




 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 rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

Code Smell

String offset-based methods should be preferred for finding substrings from offsets

Code Smell

"default" clauses should be last

Code Smell

"equals" method parameters should not be marked "@Nonnull"

Code Smell

A conditionally executed single line should be denoted by indentation

Code Smell

Conditionals should start on new lines

Code Smell

Cognitive Complexity of methods should not be too high

Code Smell

Factory method injection should be used in "@Configuration" classes

Code Smell

"static" base class members should not be accessed via derived types

Code Smell

Instance methods should not write to "static" fields

Code Smell

"indexOf" checks should not be for positive numbers

Code Smell

Method overrides should not change contracts

Code Smell

Whitespace and control characters in

Basic authentication should not be used

Analyze your code

VulnerabilityCritical🔍cwe sans-top25 owasp

Basic authentication's only means of obfuscation is Base64 encoding. Since Base64 encoding is easily recognized and reversed, it offers only the thinnest veil of protection to your users, and should not be used.

Noncompliant Code Example

```
// Using HttpPost from Apache HttpClient
String encoding = Base64Encoder.encode ("login:passwd");
org.apache.http.client.methods.HttpPost httpPost = new HttpPost
httpPost.setHeader("Authorization", "Basic " + encoding); /

or

// Using HttpURLConnection
String encoding = Base64.getEncoder().encodeToString(("login
HttpURLConnection conn = (HttpURLConnection) url.openConnection
conn.setRequestMethod("POST");
conn.setDoOutput(true);
conn.setRequestProperty("Authorization", "Basic " + encoding
```

See

- OWASP Top 10 2021 Category A4 - Insecure Design
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Web Service Security Cheat Sheet
- MITRE, CWE-522 - Insufficiently Protected Credentials
- SANS Top 25 - Porous Defenses

Available In:

sonarlint





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](#)

https://rules.sonarsource.com/java/RSPEC-2647

1/2

literals should be explicit  Code Smell
Null should not be returned from a "Boolean" method  Code Smell
Classes should not access their own subclasses during initialization  Code Smell
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop  Code Smell