




 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... 

Future keywords should not be used as names

 Code Smell

Thread suspensions should not be vulnerable to Denial of Service attacks

 Vulnerability

A new session should be created during user authentication

 Vulnerability

JWT should be signed and verified with strong cipher algorithms

 Vulnerability

Cipher algorithms should be robust

 Vulnerability

Encryption algorithms should be used with secure mode and padding scheme

 Vulnerability

Server hostnames should be verified during SSL/TLS connections

 Vulnerability

Insecure temporary file creation methods should not be used

 Vulnerability

Passwords should not be stored in plain-text or with a fast hashing algorithm

 Vulnerability

Server certificates should be verified during SSL/TLS connections

 Vulnerability

Persistent entities should not be used as arguments of "@RequestMapping" methods

 Vulnerability

Methods should not call same-class methods with incompatible "@Transactional" values

Analyze your code

 Bug

 Blocker

 ?

 spring

When using Spring proxies, calling a method in the same class (e.g. `this.aMethod()`) with an incompatible `@Transactional` requirement will result in runtime exceptions because Spring only "sees" the caller and makes no provisions for properly invoking the callee.

Therefore, certain calls should never be made within the same class:

From	To
non-@Transactional	MANDATORY, NESTED, REQUIRED, REQUIRES_NEW
MANDATORY	NESTED, NEVER, NOT_SUPPORTED, REQUIRES_NEW
NESTED	NESTED, NEVER, NOT_SUPPORTED, REQUIRES_NEW
NEVER	MANDATORY, NESTED, REQUIRED, REQUIRES_NEW
NOT_SUPPORTED	MANDATORY, NESTED, REQUIRED, REQUIRES_NEW
REQUIRED or @Transactional	NESTED, NEVER, NOT_SUPPORTED, REQUIRES_NEW
REQUIRES_NEW	NESTED, NEVER, NOT_SUPPORTED, REQUIRES_NEW
SUPPORTS	MANDATORY, NESTED, NEVER, NOT_SUPPORTED, REQUIRED, REQUIRES_NEW

Noncompliant Code Example

```
@Override
public void doTheThing() {
    // ...
    actuallyDoTheThing(); // Noncompliant
}

@Override
@Transactional
public void actuallyDoTheThing() {
    // ...
}
```

Available In:

sonarlint  | sonarcloud  | sonarqube 

"HttpSecurity" URL patterns should be correctly ordered

 Vulnerability

LDAP connections should be authenticated

 Vulnerability

Cryptographic keys should be robust

 Vulnerability

Weak SSL/TLS protocols should not be used

 Vulnerability