

































-  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... 🔍

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

"HttpSecurity" URL patterns should be correctly ordered

Vulnerability

LDAP connections should be authenticated

Vulnerability

Cryptographic keys should be robust

Vulnerability

Recursion should not be infinite

Analyze your code

BugBlocker?🔍 suspicious

Recursion is a method of solving a computational problem where a method calls itself with smaller instances of the same input. This can happen when a method invokes itself or when a pair of methods invoke each other. It can be a useful tool, but unless the method includes a provision to break out of the recursion and return, the recursion will continue until the stack overflows and the program crashes.

Noncompliant Code Example

```
int myPow(int num, int exponent) {
    num = num * my_pow(num, exponent - 1); // Noncompliant
    return num; // this is never reached
}
```

Compliant Solution

```
int myPow(int num, int exponent) {
    if(exponent > 1) {
        num = num * my_pow(num, exponent - 1);
    }
    return num;
}
```





Available In:

sonarcloud

© 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-2190

1/2

|   |
|---|
| <div>Weak SSL/TLS protocols should not be used</div> <div> Vulnerability</div>         |
| <div>"SecureRandom" seeds should not be predictable</div> <div> Vulnerability</div>    |
| <div>Cipher Block Chaining IVs should be unpredictable</div> <div> Vulnerability</div> |
| <div>Basic authentication should not be used</div> <div> Vulnerability</div>           |