sonar RULES

Products

- 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

Logging should not be vulnerable to injection attacks

Analyze your code

Vulnerability Minor injection cwe owasp sans-top25

User-provided data, such as URL parameters, POST data payloads or cookies, should always be considered untrusted and tainted. Applications logging tainted data could enable an attacker to inject characters that would break the log file pattern. This could be used to block monitors and SIEM (Security Information and Event Management) systems from detecting other malicious events.

This problem could be mitigated by sanitizing the user-provided data before logging it.

Noncompliant Code Example

```
protected void doGet(HttpServletRequest req, HttpServletResp
  String param1 = req.getParameter("param1");
  Logger.info("Param1: " + param1 + " " + Logger.getName());
  // ...
}
```

Compliant Solution

```
protected void doGet(HttpServletRequest req, HttpServletResp
  String param1 = req.getParameter("param1");

  // Replace pattern-breaking characters
  param1 = param1.replaceAll("[\n\r\t]", "_");

  Logger.info("Param1: " + param1 + " " + Logger.getName());
  // ...
}
```

See

- OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
- OWASP Cheat Sheet - Logging
- OWASP Attack Category - Log Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-117 - Improper Output Neutralization for Logs
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud sonarqube Developer Edition

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