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...

be the same or differ only by capitalization
Code Smell

Switch cases should end with an unconditional "break" statement
Code Smell

"switch" statements should not contain non-case labels
Code Smell

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

## Printf-style format strings should not lead to unexpected behavior at runtime

Analyze your code

Bug Blocker cert

Because `printf`-style format strings are interpreted at runtime, rather than validated by the Java compiler, they can contain errors that lead to unexpected behavior or runtime errors. This rule statically validates the good behavior of `printf`-style formats when calling the `format(...)` methods of `java.util.Formatter`, `java.lang.String`, `java.io.PrintStream`, `MessageFormat`, and `java.io.PrintWriter` classes and the `printf(...)` methods of `java.io.PrintStream` or `java.io.PrintWriter` classes.

### Noncompliant Code Example

```
String.format("The value of my integer is %d", "Hello World"
String.format("Duke's Birthday year is %tX", c);  //Noncompl
String.format("Display %0$d and then %d", 1);   //Noncomplia
String.format("Not enough arguments %d and %d", 1);  //Nonco
String.format("%< is equals to %d", 2);   //Noncompliant; th

MessageFormat.format("Result {1}.", value); // Noncompliant;
MessageFormat.format("Result {{0}.", value); // Noncompliant
MessageFormat.format("Result ' {0}", value); // Noncompliant

java.util.logging.Logger logger;
logger.log(java.util.logging.Level.SEVERE, "Result {1}!", 14

org.slf4j.Logger slf4jLog;
org.slf4j.Marker marker;

slf4jLog.debug(marker, "message {}"); // Noncompliant - Not

org.apache.logging.log4j.Logger log4jLog;
log4jLog.debug("message {}"); // Noncompliant - Not enough a
```

### Compliant Solution

```
String.format("The value of my integer is %d", 3);
String.format("Duke's Birthday year is %tY", c);
String.format("Display %1$d and then %d", 1);
String.format("Not enough arguments %d and %d", 1, 2);
String.format("%d is equals to %<", 2);

MessageFormat.format("Result {0}.", value);
MessageFormat.format("Result {0} & {1}.", value, value);
MessageFormat.format("Result {0}.", myObject);

java.util.logging.Logger logger;
logger.log(java.util.logging.Level.SEVERE, "Result {1},{2}!"

org.slf4j.Logger slf4jLog;
org.slf4j.Marker marker;
```

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

```
slf4jLog.debug(marker, "message {}", 1);

org.apache.logging.log4j.Logger log4jLog;
log4jLog.debug("message {}", 1);
```

**See**

- CERT, FIO47-C. - Use valid format strings

Available In:

sonarlint sonarcloud sonarqube