




 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... 🔍

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

### "ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization

Analyze your code

Vulnerability

Minor

cwe owasp

ActiveMQ can send/receive JMS Object messages (named ObjectMessage in ActiveMQ context) to comply with JMS specification. Internally, ActiveMQ relies on Java serialization mechanism for marshaling/unmarshaling of the message payload. Deserialization based on data supplied by the user could lead to remote code execution attacks, where the structure of the serialized data is changed to modify the behavior of the object being unserialized.

To limit the risk to be victim of such attack, ActiveMQ 5.12.2+ enforces developers to explicitly whitelist packages that can be exchanged using ObjectMessages.

#### Noncompliant Code Example

```
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory();
factory.setTrustAllPackages(true); // Noncompliant

ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory();
// no call to factory.setTrustedPackages(...);
```

#### Compliant Solution

```
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory();
factory.setTrustedPackages(Arrays.asList("org.mypackage1", "org.mypackage2"));
```

#### See

- [OWASP Top 10 2021 Category A8](#) - Software and Data Integrity Failures
- [OWASP Top 10 2017 Category A8](#) - Insecure Deserialization
- [MITRE, CWE-502](#) - Deserialization of Untrusted Data
- [ActiveMQ ObjectMessage Security Advisory](#)
- [CVE-2015-5254](#)

Available In:





sonarlint

sonarcloud

sonarqube

https://rules.sonarsource.com/java/RSPEC-5301

1/2

<b>Local-Variable Type Inference should be used</b>  Code Smell
<b>Migrate your tests from JUnit4 to the new JUnit5 annotations</b>  Code Smell
<b>Track uses of disallowed classes</b>  Code Smell
<b>Track uses of "@SuppressWarnings" annotations</b>  Code Smell