




 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... 🔍

Stream.toList() method should be used instead of "collectors" when unmodifiable list needed

Code Smell

Operator "instanceof" should be used instead of "A.class.isInstance()"

Code Smell

String multiline concatenation should be replaced with Text Blocks

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

Constructors of an "abstract" class should not be declared "public"

Code Smell

Similar tests should be grouped in a single Parameterized test

Code Smell

Tests should be stable

Code Smell

Test methods should not contain too many assertions

Code Smell

AssertJ "assertThatThrownBy" should not be used alone

Code Smell

Character classes in regular expressions should not contain the same character twice

Code Smell

XML parsers should not load external schemas

Analyze your code

Vulnerability Major ⓘ

By default XML processors attempt to load all XML schemas and DTD (their locations are defined with `xsi:schemaLocation` attributes and `DOCTYPE` declarations), potentially from an external storage such as file system or network, which may lead, if no restrictions are put in place, to **server-side request forgery (SSRF)** vulnerabilities.

Noncompliant Code Example

For [DocumentBuilder](#), [SAXParser](#) and [Schema](#) JAPX factories:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true); // Noncompliant
factory.setFeature("http://apache.org/xml/features/nonvalidating", true);

SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true); // Noncompliant
factory.setFeature("http://apache.org/xml/features/nonvalidating", true);

SchemaFactory schemaFactory = SchemaFactory.newInstance(XMLSCHEMA_1_1);
schemaFactory.setFeature("http://apache.org/xml/features/nonvalidating", true);
```

For [Dom4j](#) library:

```
SAXReader xmlReader = new SAXReader(); // Noncompliant
xmlReader.setFeature("http://apache.org/xml/features/nonvalidating", true);
```

For [Jdom2](#) library:

```
SAXBuilder builder = new SAXBuilder();
builder.setFeature("http://apache.org/xml/features/nonvalidating", true);
```

Compliant Solution

For [DocumentBuilder](#), [SAXParser](#) and [Schema](#) JAPX factories:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setFeature("http://apache.org/xml/features/nonvalidating", false);




SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setFeature("http://apache.org/xml/features/nonvalidating", false);

SchemaFactory schemaFactory = SchemaFactory.newInstance(XMLSCHEMA_1_1);
schemaFactory.setFeature("http://apache.org/xml/features/nonvalidating", false);
```

For [Dom4j](#) library:

https://rules.sonarsource.com/java/RSPEC-6374

1/2

| |
|---|
| Names of regular expressions named groups should be used |
|  Code Smell |
| Regexes containing characters subject to normalization should use the CANON_EQ flag |
|  Code Smell |
| Regular expressions should not be too complicated |
|  Code Smell |
| JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion |

```
SAXReader xmlReader = new SAXReader(); // Noncompliant
xmlReader.setFeature("http://apache.org/xml/features/nonvalida
```

For [Jdom2](#) library:

```
SAXBuilder builder = new SAXBuilder();
builder.setFeature("http://apache.org/xml/features/nonvalida
```

Exceptions

This rules does not raise an issue when an EntityResolver is set.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newI
factory.setValidating(true);
DocumentBuilder builder = factory.newDocumentBuilder();
builder.setEntityResolver(new MyEntityResolver());

SAXBuilder builder = new SAXBuilder();
builder.setFeature("http://apache.org/xml/features/nonvalida
builder.setEntityResolver(new EntityResolver());
```

See

- [Oracle Java Documentation](#) - XML External Entity Injection Attack
- [OWASP Top 10 2017 Category A4](#) - XML External Entities (XXE)
- [OWASP XXE Prevention Cheat Sheet](#)
- [MITRE, CWE-611](#) - Information Exposure Through XML External Entity Reference
- [MITRE, CWE-827](#) - Improper Control of Document Type Definition

Available In:

