




 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


Abstract class names should comply with a naming convention	
Strings literals should be placed on the left side when checking for equality	
Files should contain an empty newline at the end	
Source code should be indented consistently	
A close curly brace should be located at the beginning of a line	
Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines	
Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line	
An open curly brace should be located at the beginning of a line	
An open curly brace should be located at the end of a line	
Tabulation characters should not be used	
Functions should not be defined with a variable number of arguments	

Classes should not be loaded dynamically

Analyze your code

 Vulnerability

 Critical



Dynamically loaded classes could contain malicious code executed by a static class initializer. I.E. you wouldn't even have to instantiate or explicitly invoke methods on such classes to be vulnerable to an attack.

This rule raises an issue for each use of dynamic class loading.

Noncompliant Code Example

```
String className = System.getProperty("messageClassName");
Class clazz = Class.forName(className); // Noncompliant
```

- See
- OWASP Top 10 2017 Category A1 - Injection
  - MITRE, CWE-470 - Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

Deprecated

This rule is deprecated; use {rule:javasecurity:S6173} instead.

Available In:

sonarlint  | sonarcloud  | sonarqube 

<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