

































-  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



## TypeScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TYPESCRIPT code

All rules279

Vulnerability27

Bug51


Security Hotspot43

Code Smell158


Quick Fix50


Tags ▾


Search by name... 🔍


 Code Smell


Primitive return types should be used





 Default type parameters should be omitted





 Type assertions should use "as"





 Method overloads should be grouped together





 Interfaces should not be empty



 Trailing commas should be used



 "import" should be used to include external code



 Braces and parentheses should be used consistently with arrow functions



 Destructuring syntax should be used for assignments



 Template strings should be used instead of concatenation




 Shorthand object properties should be grouped at the beginning or end of an object declaration





 Object literal shorthand syntax should

### Only "while", "do", "for" and "switch" statements should be labelled

Analyze your code

 Code Smell

 Major ?

 pitfall




Any statement or block of statements can be identified by a label, but those labels should be used only on while, do-while, for and switch statements. Using labels in any other context leads to unstructured, confusing code.

#### Noncompliant Code Example

```
myLabel: if (i % 2 == 0) { // Noncompliant
  if (i == 12) {
    console.log("12");
    break myLabel;
  }
  console.log("Even number, but not 12");
}
```

#### Compliant Solution





```
myLabel: for (i = 0; i < 10; i++) { // Compliant
  console.log("Loop");
  break myLabel;
}
```

Available In:  
 |  | 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)

https://rules.sonarsource.com/typescript/RSPEC-1439

1/2

<b>be used</b>  Code Smell
<b>Strings and non-strings should not be added</b>  Code Smell
<b>Primitive types should be omitted from initialized or defaulted declarations</b>  Code Smell
<b>Non-null assertions should not be used</b>  Code Smell