




 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... 🔍

Setting loose POSIX file permissions is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Comma operator should not be used

Code Smell

Regular expressions should not contain empty groups

Code Smell

Regular expressions should not contain multiple spaces

Code Smell

Chai assertions should have only one reason to succeed

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

Tests should check which exception is thrown

Code Smell

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

Code Smell

Names of regular expressions named groups should be used

Code Smell

Loop counters should not be assigned to from within the loop body

Analyze your code

Code Smell

Critical

Loop counters should not be modified in the body of the loop. However other loop control variables representing logical values may be modified in the loop, for example a flag to indicate that something has been completed, which is then tested in the for statement.

Noncompliant Code Example

```
var names = [ "Jack", "Jim", "", "John" ];
for (var i = 0; i < names.length; i++) {
  if (!names[i]) {
    i = names.length; // Noncompliant
  } else {
    console.log(names[i]);
  }
}
```





Compliant Solution

```
var names = [ "Jack", "Jim", "", "John" ];
for (var name of names) {
  if (!name) {
    break; // Compliant
  } else {
    console.log(name);
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

© 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](#)

<div>Regular expressions should not be too complicated</div> <div> Code Smell</div>
<div>Optional property declarations should not use both '?' and 'undefined' syntax</div> <div> Code Smell</div>
<div>Shorthand promises should be used</div> <div> Code Smell</div>
<div>Template literals should not be nested</div> <div> Code Smell</div>
<div>"undefined" should not be passed as</div>