




 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



## JavaScript static code analysis

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

All rules 285

Vulnerability 29

Bug 62

Security Hotspot 43

Code Smell 151

Quick Fix 41

Tags ▾

Search by name... 🔍

Cyclomatic Complexity of functions should not be too high

Code Smell

"strict" mode should be used with caution

Code Smell

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply

Code Smell

"switch" statements should have "default" clauses

Code Smell

"if ... else if" constructs should end with "else" clauses

Code Smell

Control structures should use curly braces

Code Smell

String literals should not be duplicated

Code Smell

Expressions should not be too complex

Code Smell

Local storage should not be used

Vulnerability

Template literal placeholder syntax should not be used in regular strings

Bug

Built-in objects should not be overridden

Bug

"for...in" loops should filter properties before acting on them

"in" should not be used on arrays

Analyze your code

Code Smell

Major

Quick Fix

pitfall

The `in` operator used on an array is valid but the code will likely not have the expected behavior. The `in` operator deals with the indexes of the array, not with the values.

If checking for an array slot is indeed desired, using `hasOwnProperty` makes the code intention clearer.

Noncompliant Code Example

```
function func1() {
  let arr = ["a", "b", "c"];

  let expectedValue = "b";
  if (expectedValue in arr) { // Noncompliant, will be always true
    return expectedValue + " found in the array";
  } else {
    return expectedValue + " not found";
  }
}

function func2() {
  let arr = ["a", "b", "c"];

  let expectedValue = "1"; // index #1 is corresponding to "b"
  if (expectedValue in arr) { // Noncompliant, will be always true
    return expectedValue + " found in the array";
  } else {
    return expectedValue + " not found";
  }
}
```

Compliant Solution

```
function func() {
  let arr = ["a", "b", "c"];





  let expectedValue = "b";
  if (arr.includes(expectedValue)) {
    return expectedValue + " was found in the array";
  } else {
    return expectedValue + " not found";
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

https://rules.sonarsource.com/javascript/RSPEC-4619

1/2

 Bug
<b>Results of operations on strings should not be ignored</b>  Bug
<b>Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression</b>  Code Smell
<b>"for in" should not be used with iterables</b>  Code Smell

© 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](#)