




 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... 🔍


Promise rejections should not be caught by 'try' block




Collection elements should not be replaced unconditionally




Errors should not be created without being thrown




Collection sizes and array length comparisons should make sense




All branches in a conditional structure should not have exactly the same implementation




Destructuring patterns should not be empty




The output of functions that don't return anything should not be used




Comma and logical OR operators should not be used in switch cases




Generators should "yield" something




Attempts should not be made to update "const" variables



Strict equality operators should not be used with dissimilar types







"new" operators should be used with functions



### Results of "in" and "instanceof" should be negated rather than operands

Analyze your code

 Bug  Critical  Quick Fix 

Mixing up the order of operations will almost always yield unexpected results.

Similarly, mis-applied negation will also yield bad results. For instance consider the difference between `!key in dict` and `!(key in dict)`. The first looks for a boolean value (`!key`) in `dict`, and the other looks for a string and inverts the result. `!obj instanceof SomeClass` has the same problem.

This rule raises an issue when the left operand of an `in` or `instanceof` operator is negated.

#### Noncompliant Code Example




```
if (!"prop" in myObj) { // Noncompliant; "in" operator is
  doTheThing(); // this block will be never executed
}

if (!foo instanceof MyClass) { // Noncompliant; "!foo" retu
  doTheOtherThing(); // this block is never executed
}
```

#### Compliant Solution

```
if (!("prop" in myObj)) {
  doTheThing();
}





if (!(foo instanceof MyClass)) {
  doTheOtherThing();
}
```

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/javascript/RSPEC-3812

1/2

 Bug
<b>Non-existent operators '+=', '=-' and '!=' should not be used</b>  Bug
<b>"NaN" should not be used in comparisons</b>  Bug
<b>Setters should not return values</b>  Bug
<b>Properties of variables with "null" or "undefined" values should not be accessed</b>