




 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 rules285

Vulnerability29

Bug62


Security Hotspot43

Code Smell151


Quick Fix41


Tags ▾


Search by name... 🔍


 Vulnerability


Template literal placeholder syntax should not be used in regular strings


 Bug


 Built-in objects should not be overridden


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


 Results of operations on strings should not be ignored


 Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression


 "for in" should not be used with iterables

 Functions should use "return" consistently

 Variables and functions should not be declared in the global scope


 Arithmetic operators should only have numbers as operands


 Values not convertible to numbers should not be used in numeric comparisons


 Arithmetic operations should not result in "NaN"

Functions should always return the same type

Analyze your code

 Code Smell

 Major ?

 confusing

Unlike strongly typed languages, JavaScript does not enforce a return type on a function. This means that different paths through a function can return different types of values, which can be very confusing to the user and significantly harder to maintain.

Noncompliant Code Example

```
function foo(a) { // Noncompliant
  if (a === 1) {
    return true;
  }
  return 3;
}
```

Compliant Solution

```
function foo(a) {
  if (a === 1) {
    return true;
  }
  return false;
}
```




Exceptions

Functions returning this are ignored.

```
function foo() {
  // ...
  return this;
}
```

Functions returning expressions having type any are ignored.

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-3800

1/2

 Code Smell
"arguments" should not be accessed directly  Code Smell
Comparison operators should not be used with strings  Code Smell
Property getters and setters should come in pairs  Code Smell
JavaScript parser failure