




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 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... 🔍


 Code Smell


Names of regular expressions named groups should be used




 Regular expressions should not be too complicated


Shorthand promises should be used




 Template literals should not be nested


"in" should not be used on arrays




 Assignments should not be redundant


Functions should not have identical implementations




 Sparse arrays should not be declared


Array-mutating methods should not be used misleadingly






 Collection and array contents should be used

Functions should always return the same type



 Arguments to built-in functions should match documented types

Replacement strings should reference existing regular expression groups

 Bug  Major  regex

If the first parameter of `String.replace` is a regular expression, a special syntax can be used in the replacement string to reference capturing groups. Use `$n` to reference the group by number and `${<Name>}` to reference the group by name. Because replacements strings in `String.replace` are interpreted at runtime, nothing prevents you to reference nonexistent group, with nonexistent index or bad name, then the resulting string will be wrong. This rule statically validates that all referenced groups exist when replacing with `String.replace` or `String.replaceAll` methods.




Noncompliant Code Example

```
const str = 'James Bond';
console.log(str.replace(/(\w+)\s(\w+)/, '$1, $0 $1')); // No
console.log(str.replace(/(?<firstName>\w+)\s(?<lastName>\w+)/, '$1, $0 $1'));
```

Compliant Solution

```
const str = 'James Bond';
console.log(str.replace(/(\w+)\s(\w+)/, '$2, $1 $2'));
console.log(str.replace(/(?<firstName>\w+)\s(?<lastName>\w+)/, '$1, $0 $1'));
```





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](#)

https://rules.sonarsource.com/javascript/RSPEC-6328

1/2

Literals should not be thrown  Code Smell
Functions should not be called both with and without "new"  Code Smell
Array indexes should be numeric  Code Smell
Assertion arguments should be passed in the correct order  Code Smell