

































-  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... 🔍

Template literals should not be nested

Code Smell

"undefined" should not be passed as the value of optional parameters

Code Smell

"in" should not be used on arrays

Code Smell

Assignments should not be redundant

Code Smell

Functions should not have identical implementations

Code Smell

Sparse arrays should not be declared

Code Smell

Array-mutating methods should not be used misleadingly

Code Smell

Collection and array contents should be used

Code Smell

Literals should not be thrown

Code Smell

Array indexes should be numeric

Code Smell

Assertion arguments should be passed in the correct order

Code Smell

Ternary operators should not be nested

Code Smell

"delete" should not be used on arrays

Replacement strings should reference existing regular expression groups

Analyze your code

BugMajor?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/typescript/RSPEC-6328

1/2

 Code Smell
Variables and functions should not be redeclared  Code Smell
"indexOf" checks should not be for positive numbers  Code Smell
"arguments.caller" and "arguments.callee" should not be used  Code Smell
Multiline blocks should be enclosed in curly braces