




 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 rules 279

 Vulnerability 27

 Bug 51

 Security Hotspot 43


 Code Smell 158

 Quick Fix 50


Tags ▾

Search by name... 🔍


Regular expressions should not be too complicated

 Code Smell


Optional property declarations should not use both '?' and 'undefined' syntax

 Code Smell


Shorthand promises should be used

 Code Smell


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

Regular expressions with the global flag should be used with caution

Analyze your code

 Bug  Major   regex

Regular expressions with the global flag turned on can be a source of tricky bugs for uninformed users, and should therefore be used with caution. Such regular expressions are stateful, that is, they maintain an internal state through the `lastIndex` property, which is updated and used as starting point on every call to `RegExp.prototype.test()` and `RegExp.prototype.exec()`, even when testing a different string. The `lastIndex` property is eventually reset when these functions return `false` and `null` respectively.

This rule raises an issue when:

- a regular expression is tested against different inputs with `RegExp.prototype.test()` or `RegExp.prototype.exec()`
- a regular expression is defined within a loop condition while used with `RegExp.prototype.exec()`
- a regular expression turns on both global `g` and sticky `y` flags

Noncompliant Code Example

```
const datePattern = /\d{4}-\d{2}-\d{2}/g;
datePattern.test('2020-08-06');
datePattern.test('2019-10-10'); // Noncompliant: the regex w

const str = 'foodie fooled football';
while ((result = /foo*/g.exec(str)) !== null) { // Noncompli
  /* ... */
}

const stickyPattern = /abc/y; // Noncompliant: a regex defi
stickyPattern.test(/* ... */);
```




Compliant Solution

```
const datePattern = /\d{4}-\d{2}-\d{2}/;
datePattern.test('2020-08-06');
datePattern.test('2019-10-10'); // Compliant

const reg = /foo*/g;
const str = 'foodie fooled football';
while ((result = reg.exec(str)) !== null) { // Compliant
  /* ... */
}

const stickyPattern = /abc/y; // Compliant
stickyPattern.test(/* ... */);
```

Available In:

 |  | 

https://rules.sonarsource.com/typescript/RSPEC-6351

1/2

 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	
 Code Smell	
Variables and functions should not be redeclared	
 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](#)