




 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... 🔍

 Code Smell


Variables should be declared with "let" or "const"




Unchanged variables should be marked "const"




Wildcard imports should not be used




"switch" statements should not be nested




Cyclomatic Complexity of functions should not be too high




"strict" mode should be used with caution




Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply




"switch" statements should have "default" clauses




"if ... else if" constructs should end with "else" clauses



Control structures should use curly braces




String literals should not be duplicated





Expressions should not be too


Shorthand promises should be used

Analyze your code

 Code Smell

 Major ?

 Quick Fix ?

 proficiency

When a `Promise` needs to only "resolve" or "reject", it's more efficient and readable to use the methods specially created for such use cases:
`Promise.resolve(value)` and `Promise.reject(error)`.




Noncompliant Code Example

```
let fulfilledPromise = new Promise(resolve => resolve(42));
let rejectedPromise = new Promise(function(resolve, reject)
  reject('fail');
});
```

Compliant Solution

```
let fulfilledPromise = Promise.resolve(42);
let rejectedPromise = Promise.reject('fail');
```





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/typescript/RSPEC-4634

1/2

<div>Expressions should not be too complex</div> <div> Code Smell</div>
<div>Template literal placeholder syntax should not be used in regular strings</div> <div> Bug</div>
<div>Built-in objects should not be overridden</div> <div> Bug</div>
<div>"for...in" loops should filter properties before acting on them</div> <div> Bug</div>