




 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... 🔍

Code Smell

Functions should use "return" consistently

Code Smell

"arguments" should not be accessed directly

Code Smell

Comparison operators should not be used with strings

Code Smell

Private properties that are only assigned in the constructor or at declaration should be "readonly"

Code Smell

Property getters and setters should come in pairs

Code Smell

JavaScript parser failure

Code Smell

The ternary operator should not be used

Code Smell

"===" and "!==" should be used instead of "==" and "!="

Code Smell

Functions should not have too many lines of code

Code Smell

Track comments matching a regular expression

Code Smell

Statements should be on separate lines

Code Smell

Variables and functions should not be redeclared

Analyze your code

Code Smell

Major

confusing

This rule checks that a declaration doesn't use a name that is already in use. Indeed, it is possible to use the same symbol multiple times as either a variable or a function, but doing so is likely to confuse maintainers. Further it's possible that such reassignments are made in error, with the developer not realizing that the value of the variable is overwritten by the new assignment.

This rule also applies to function parameters.

Noncompliant Code Example

```
var a = 'foo';
function a() {} // Noncompliant
console.log(a); // prints "foo"

function myFunc(arg) {
  var arg = "event"; // Noncompliant, argument value is lost
}

fun(); // prints "bar"

function fun() {
  console.log("foo");
}

fun(); // prints "bar"

function fun() { // Noncompliant
  console.log("bar");
}

fun(); // prints "bar"
```

Compliant Solution

```
var a = 'foo';
function otherName() {}
console.log(a);

function myFunc(arg) {
  var newName = "event";
}

fun(); // prints "foo"


function fun() {
  print("foo");
}

fun(); // prints "foo"
```


https://rules.sonarsource.com/typescript/RSPEC-2814

1/2


Magic numbers should not be used

 Code Smell


Collapsible "if" statements should be merged

 Code Smell

Standard outputs should not be used directly to log anything




 Code Smell

Files should not have too many lines of code

 Code Smell

Lines should not be too long

```
function printBar() {  
  print("bar");  
}  
  
printBar(); // prints "bar"
```

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](#)