




 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... 🔍

 Bug


The base should be provided to "parseInt"




Function declarations should not be made within blocks




Writing cookies is security-sensitive




"continue" should not be used




Primitive return types should be used




Default type parameters should be omitted




Type assertions should use "as"




Method overloads should be grouped together




Interfaces should not be empty




Trailing commas should be used



"import" should be used to include external code





Braces and parentheses should be used consistently with arrow functions




Functions should not be defined inside loops

Analyze your code

 Code Smell




 Major ?

 suspicious

Defining a function inside of a loop can yield unexpected results. Such a function keeps references to the variables which are defined in outer scopes. All function instances created inside the loop therefore see the same values for these variables, which is probably not expected.

Noncompliant Code Example




```
var funs = [];
for (var i = 0; i < 13; i++) {
  funs[i] = function() { // Non-Compliant
    return i;
  };
}
console.log(funs[0]()); // 13 instead of 0
console.log(funs[1]()); // 13 instead of 1
console.log(funs[2]()); // 13 instead of 2
console.log(funs[3]()); // 13 instead of 3
...
```

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-1515

1/2

<div><div>Destructuring syntax should be used for assignments</div><div> Code Smell</div></div>
<div><div>Template strings should be used instead of concatenation</div><div> Code Smell</div></div>
<div><div>Shorthand object properties should be grouped at the beginning or end of an object declaration</div><div> Code Smell</div></div>
<div><div>Object literal shorthand syntax should be used</div></div>