




 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... 🔍

Functions should not be defined inside loops

Code Smell

"switch" statements should not have too many "case" clauses

Code Smell

Only "while", "do", "for" and "switch" statements should be labelled

Code Smell

Sections of code should not be commented out

Code Smell

Unused function parameters should be removed

Code Smell

Track uses of "FIXME" tags

Code Smell

Assignments should not be made from within sub-expressions

Code Smell

Labels should not be used

Code Smell

Variables should not be shadowed

Code Smell

Redundant pairs of parentheses should be removed

Code Smell

Nested blocks of code should not be left empty

Code Smell

Functions should not have too many parameters

Code Smell

Destructuring patterns should not be empty

Analyze your code

BugMajor?

Destructuring is a convenient way of extracting multiple values from data stored in (possibly nested) objects and arrays. However, it is possible to create an empty pattern that has no effect. When empty curly braces or brackets are used to the right of a property name most of the time the intent was to use a default value instead.

This rule raises an issue when empty destructuring pattern is used.

Noncompliant Code Example

```
var {a: {}, b} = myObj; // Noncompliant
function foo({first: [], second}) { // Noncompliant
  // ...
}
```

Compliant Solution

```
var {a = {}, b} = myObj;
function foo({first = [], second}) {
  // ...
}
```

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-3799

1/2

<div>OS commands should not be vulnerable to argument injection attacks</div> <div> Vulnerability</div>
<div>Repeated patterns in regular expressions should not match the empty string</div> <div> Bug</div>
<div>Empty collections should not be accessed or iterated</div> <div> Bug</div>
<div>"delete" should be used only with object properties</div> <div> Bug</div>