




 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... 🔍

Formatting SQL queries is security-sensitive

Security Hotspot

Comma operator should not be used

Code Smell

Regular expressions should not contain empty groups

Code Smell

Regular expressions should not contain multiple spaces

Code Smell

Chai assertions should have only one reason to succeed

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

Tests should check which exception is thrown

Code Smell

Character classes in regular expressions should not contain the same character twice

Code Smell

Names of regular expressions named groups should be used

Code Smell

Regular expressions should not be too complicated

Code Smell

"void" should not be used

Analyze your code

Code Smell

Critical

confusing

The `void` operator evaluates its argument and unconditionally returns `undefined`. It can be useful in pre-ECMAScript 5 environments, where `undefined` could be reassigned, but generally, its use makes code harder to understand.

#### Noncompliant Code Example

```
void doSomething();
```

#### Compliant Solution

```
doSomething();
```

#### Exceptions

No issue is raised when `void 0` is used in place of `undefined`.

```
if (parameter === void 0) {...}
```

No issue is raised when `void` is used before immediately invoked function expressions.

```
void (function() {
  ...
})();
```

No issue is raised when `void`'s argument is a promise.





```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('done');
  }, 3000);
});
void promise;
```

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-3735

1/2

<div>Optional property declarations should not use both '?' and 'undefined' syntax</div> <div> Code Smell</div>
<div>Shorthand promises should be used</div> <div> Code Smell</div>
<div>Template literals should not be nested</div> <div> Code Smell</div>
<div>"undefined" should not be passed as the value of optional parameters</div> <div> Code Smell</div>
<div>"?" should not be used as a parameter</div>