

































-  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...

All code should be reachable

Bug

Loops with at most one iteration should be refactored

Bug

Variables should not be self-assigned

Bug

Bitwise operators should not be used in boolean contexts

Bug

Constructing arguments of system commands from user input is security-sensitive

Security Hotspot

Allowing requests with excessive content length is security-sensitive

Security Hotspot

Statically serving hidden files is security-sensitive

Security Hotspot

Using intrusive permissions is security-sensitive

Security Hotspot

Disabling auto-escaping in template engines is security-sensitive

Security Hotspot

Using shell interpreter when executing OS commands is security-sensitive

Security Hotspot

Setting loose POSIX file permissions is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Union and intersection types should not be defined with duplicated elements

Analyze your code

Code Smell

Critical

Quick Fix

The TypeScript type system offers a basic support for composite types:

- Union Types represent a value that can be one of the several types. They are constructed using a vertical bar (|) like the following type `NumberOrString = number | string`.
- Intersection Types combine multiple types into one, so that the object of such type will have all the members of all intersection type elements. They are constructed using an ampersand (&) like the following type `SerializablePerson = Person & Serializable`. Intersection Types are often used to represent mixins.

Duplicating types when defining a union or interaction type makes the code less readable. Moreover duplicated types might be a simple mistake and another type should be used instead.

Noncompliant Code Example

```
function padLeft(value: string, padding: string | number | s
// ...
}

function extend(p : Person) : Person & Person & Loggable { /
// ...
}
```






Compliant Solution

```
function padLeft(value: string, padding: string | number | b
// ...
}

function extend(p : Person) : Person & Loggable {
// ...
}
```

Available In:  
sonarlint | sonarcloud | sonarqube

https://rules.sonarsource.com/typescript/RSPEC-46211/2

 Code Smell
<b>Comma operator should not be used</b>  Code Smell
<b>Regular expressions should not contain empty groups</b>  Code Smell
<b>Regular expressions should not contain multiple spaces</b>  Code Smell
<b>Chai assertions should have only one reason to succeed</b>  Code Smell