




 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

 Code Smell
Primitive types should be omitted from initialized or defaulted declarations
 Code Smell
Non-null assertions should not be used
 Code Smell
"undefined" should not be assigned
 Code Smell
Trailing commas should not be used
 Code Smell
Array constructors should not be used
 Code Smell
Quotes for string literals should be used consistently
 Code Smell
Statements should end with semicolons
 Code Smell
Comments should not be located at the end of lines of code
 Code Smell
Loops should not contain more than a single "break" or "continue" statement
 Code Smell
Variable, property and parameter names should comply with a naming convention
 Code Smell
Lines should not end with trailing whitespaces
 Code Smell

Type guards should be used

Analyze your code

-  Code Smell
-  Minor
-  Quick Fix
-  proficiency

A common idiom in JavaScript to differentiate between two possible types is to check for the presence in the object of a member of the desired type. Usually, to simplify the code, a boolean function is created to check the type.

Typescript provides user defined type guard functions. These are just functions with a return type of argumentName is SomeType. Such functions return true if the argument is of the specified type. One of the advantages of using such a function is that in a conditional block where the condition is a type guard, the compiler automatically performs the appropriate casts, so explicit casting becomes unnecessary.

This rule raises an issue when a boolean function checking for the type of its only argument can be replaced with a user-defined type guard function.

Noncompliant Code Example

```
function isSomething(x: BaseType) : boolean { // Noncompliant
    return (<Something>x).foo !== undefined;
}

if (isSomething(v)) {
    (<Something>v).foo();
}
```

Compliant Solution

```
function isSomething(x: BaseType) : x is Something {
    return (<Something>x).foo !== undefined;
}





if (isSomething(v)) {
    v.foo();
}
```

See

[TypeScript advanced types](#)

Available In:

sonarlint | sonarcloud | sonarqube

<div>Files should contain an empty newline at the end</div> <div> Code Smell</div>
<div>An open curly brace should be located at the end of a line</div> <div> Code Smell</div>
<div>Tabulation characters should not be used</div> <div> Code Smell</div>
<div>Function and method names should comply with a naming convention</div> <div> Code Smell</div>