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...

Regular expressions should not contain control characters
Bug

Alternation in regular expressions should not contain empty alternatives
Bug

Mocha timeout should be disabled by setting it to "0".
Bug

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

Assertions should not be given twice the same argument
Bug

Alternatives in regular expressions should be grouped when used with anchors
Bug

Promise rejections should not be caught by 'try' block
Bug

Collection elements should not be replaced unconditionally
Bug

Constructors should not be declared inside interfaces
Bug

Errors should not be created without being thrown
Bug

Collection sizes and array length comparisons should make sense
Bug

All branches in a conditional structure

Getters and setters should access the expected fields

Analyze your code

Bug Critical pitfall

Getters and setters provide a way to enforce encapsulation by providing `public` methods that give controlled access to `private` fields. However in classes with multiple fields it is not unusual that copy and paste is used to quickly create the needed getters and setters, which can result in the wrong field being accessed by a getter or setter.

This rule raises an issue in any of these cases:

- A setter does not update the field with the corresponding name.
- A getter does not access the field with the corresponding name.

Noncompliant Code Example

```
class A {
  private _x: number = 0;
  private y: number = 0;

  public get x() {  // Noncompliant: field 'x' is not used i
    return this.y;
  }

  public setX(val: number) { // Noncompliant: field 'x' is n
    this.y = val;
  }

  public getY() { // Noncompliant: field 'y' is not used in
    return this.x;
  }
}
```

Compliant Solution

```
class A {
  private _x: number = 0;
  private y: number = 0;

  public get x() {
    return this._x;
  }

  public setX(val: number) {
    this.x = val;
  }

  public getY() {
    return this.y;
  }
}
```

should not have exactly the same implementation

Bug

Destructuring patterns should not be empty

Bug

The output of functions that don't return anything should not be used

Bug

Comma and logical OR operators should not be used in switch cases

Bug

Available In:

sonarlint | sonarcloud | sonarqube