sonar RULES Products

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...

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

Comma operator should not be used
Code Smell

Regular expressions should not contain empty groups
Code Smell

"await" should only be used with promises

Analyze your code

Code Smell | Critical | confusing

It is possible to use `await` on values which are not `Promises`, but it's useless and misleading. The point of `await` is to pause execution until the `Promise`'s asynchronous code has run to completion. With anything other than a `Promise`, there's nothing to wait for.

This rule raises an issue when an `await`ed value is guaranteed not to be a `Promise`.

Noncompliant Code Example

```
let x = 42;
await x; // Noncompliant
```

Compliant Solution

```
let x = new Promise(resolve => resolve(42));
await x;

let y = p ? 42 : new Promise(resolve => resolve(42));
await y;
```

Available In:
sonarlint | sonarcloud | sonarqube

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