

- 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



C# static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C# code

- All rules 409
- Vulnerability 34
- Bug 76
- Security Hotspot 28
- Code Smell 271
- Quick Fix 52

Tags ▾

Search by name... 🔍

"protected" members
Code Smell
Underscores should be used to make large numbers readable
Code Smell
"ToString()" calls should not be redundant
Code Smell
"==" should not be used when "Equals" is overridden
Code Smell
An abstract class should have both abstract and concrete methods
Code Smell
Multiple variables should not be declared on the same line
Code Smell
Culture should be specified for "string" operations
Code Smell
"switch" statements should have at least 3 "case" clauses
Code Smell
break statements should not be used except for switch cases
Code Smell
String literals should not be duplicated
Code Smell
Files should contain an empty newline at the end
Code Smell
Unused "using" should be removed
Code Smell

"ConfigureAwait(false)" should be used

Analyze your code

- Code Smell
- Critical
- multi-threading
- async-await
- suspicious

After an awaited `Task` has executed, you can continue execution in the original, calling thread or any arbitrary thread. Unless the rest of the code needs the context from which the `Task` was spawned, `Task.ConfigureAwait(false)` should be used to keep execution in the `Task` thread to avoid the need for context switching and the possibility of deadlocks.

This rule raises an issue when code in a class library targeting .Net Framework awaits a `Task` and continues execution in the original calling thread.

The rule does not raise for .Net Core libraries as there is no `SynchronizationContext` in .Net Core.

Noncompliant Code Example

```
var response = await httpClient.GetAsync(url); // Noncompliant
```

Compliant Solution

```
var response = await httpClient.GetAsync(url).ConfigureAwait(false);
```

Available In:
 sonarlint | sonarcloud | sonarqube

A close curly brace should be located at the beginning of a line

 Code Smell

Tabulation characters should not be used

 Code Smell

Methods and properties should be named in PascalCase

 Code Smell

Track uses of in-source issue suppressions

 Code Smell