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...

Bug

Calls to "async" methods should not be blocking
Code Smell

Child class fields should not shadow parent class fields
Code Smell

Track lack of copyright and license headers
Code Smell

Exit methods should not be called
Code Smell

Classes should "Dispose" of members from the classes' own "Dispose" methods
Bug

Reading the Standard Input is security-sensitive
Security Hotspot

Using command line arguments is security-sensitive
Security Hotspot

Using Sockets is security-sensitive
Security Hotspot

Encrypting data is security-sensitive
Security Hotspot

Using regular expressions is security-sensitive
Security Hotspot

Interface methods should be callable by derived types
Code Smell

Child class fields should not differ from parent class fields only by

Parameter validation in "async"/"await" methods should be wrapped

Analyze your code

Code Smell | Major | async-await

Because of the way `async/await` methods are rewritten by the compiler, any exceptions thrown during the parameters check will happen only when the task is observed. That could happen far away from the source of the buggy code or never happen for fire-and-forget tasks.

Therefore it is recommended to split the method into two: an outer method handling the parameter checks (without being `async/await`) and an inner method to handle the iterator block with the `async/await` pattern.

This rule raises an issue when an `async` method throws any exception derived from `ArgumentException` and contains `await` keyword.

Noncompliant Code Example

```
public static async Task SkipLinesAsync(this TextReader read
{
    if (reader == null) { throw new ArgumentNullException(na
    if (linesToSkip < 0) { throw new ArgumentOutOfRangeExcep

    for (var i = 0; i < linesToSkip; ++i)
    {
        var line = await reader.ReadLineAsync().ConfigureAwa
        if (line == null) { break; }
    }
}
```

Compliant Solution

```
public static Task SkipLinesAsync(this TextReader reader, in
{
    if (reader == null) { throw new ArgumentNullException(na
    if (linesToSkip < 0) { throw new ArgumentOutOfRangeExcep

    return reader.SkipLinesInternalAsync(linesToSkip);
}

private static async Task SkipLinesInternalAsync(this TextRe
{
    for (var i = 0; i < linesToSkip; ++i)
    {
        var line = await reader.ReadLineAsync().ConfigureAwa
        if (line == null) { break; }
    }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

from parent class fields only by capitalization

Code Smell

Pointers to unmanaged memory should not be visible

Code Smell

Number patterns should be regular

Code Smell

"out" and "ref" parameters should not be used

Code Smell

Unchanged local variables should be "const"