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...

Code Smell

Static fields should not be updated in constructors
Code Smell

"IEnumerable" LINQs should be simplified
Code Smell

Fields that are only assigned in the constructor should be "readonly"
Code Smell

Static fields should not be used in generic types
Code Smell

Multiline blocks should be enclosed in curly braces
Code Smell

Boolean expressions should not be gratuitous
Code Smell

Types and methods should not have too many generic parameters
Code Smell

Write-only properties should not be used
Code Smell

Exceptions should not be thrown from property getters
Code Smell

Unused type parameters should be removed
Code Smell

Parameters should be passed in the correct order
Code Smell

Members should not have conflicting transparency annotations

Analyze your code

Vulnerability | Major | owasp pitfall

Transparency attributes, `SecurityCriticalAttribute` and `SecuritySafeCriticalAttribute` are used to identify code that performs security-critical operations. The second one indicates that it is safe to call this code from transparent, while the first one does not. Since the transparency attributes of code elements with larger scope take precedence over transparency attributes of code elements that are contained in the first element a class, for instance, with a `SecurityCriticalAttribute` can not contain a method with a `SecuritySafeCriticalAttribute`.

This rule raises an issue when a member is marked with a `System.Security` security attribute that has a different transparency than the security attribute of a container of the member.

Noncompliant Code Example

```
using System;
using System.Security;

namespace MyLibrary
{

    [SecurityCritical]
    public class Foo
    {
        [SecuritySafeCritical] // Noncompliant
        public void Bar()
        {
        }
    }
}
```

Compliant Solution

```
using System;
using System.Security;

namespace MyLibrary
{

    [SecurityCritical]
    public class Foo
    {
        public void Bar()
        {
        }
    }
}
```

Two branches in a conditional structure should not have exactly the same implementation

Code Smell

Unused assignments should be removed

Code Smell

Tests should not be ignored

Code Smell

"switch" statements should not have too many "case" clauses

Code Smell

See

- OWASP Top 10 2021 Category A5 - Security Misconfiguration
- OWASP Top 10 2017 Category A6 - Security Misconfiguration

Available In:

sonarlint | sonarcloud | sonarqube