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...

be too deep
Code Smell

Nested blocks of code should not be left empty
Code Smell

Methods should not have too many parameters
Code Smell

Collapsible "if" statements should be merged
Code Smell

OS commands should not be vulnerable to argument injection attacks
Vulnerability

Logging should not be vulnerable to injection attacks
Vulnerability

Empty collections should not be accessed or iterated
Bug

Mutable, non-private fields should not be "readonly"
Bug

"string.ToCharArray()" should not be called redundantly
Bug

"base.Equals" should not be used to check for reference equality in "Equals" if "base" is not "object"
Bug

Property assignments should not be made for "readonly" fields not constrained to reference types
Bug

Optional parameters should be passed to "base" calls

Analyze your code

Bug Major

Generally, writing the least code that will *readably* do the job is a good thing, so omitting default parameter values seems to make sense. Unfortunately, when you omit them from the `base` call in an override, you're not actually getting the job done thoroughly, because you're ignoring the value the caller passed in. The result will likely not be what the caller expected.

Noncompliant Code Example

```
public class BaseClass
{
    public virtual void MyMethod(int i = 1)
    {
        Console.WriteLine(i);
    }
}

public class DerivedClass : BaseClass
{
    public override void MyMethod(int i = 1)
    {
        // ...
        base.MyMethod(); // Noncompliant; caller's value is
    }

    static int Main(string[] args)
    {
        DerivedClass dc = new DerivedClass();
        dc.MyMethod(12);  // prints 1
    }
}
```

Compliant Solution

```
public class BaseClass
{
    public virtual void MyMethod(int i = 1)
    {
        Console.WriteLine(i);
    }
}

public class DerivedClass : BaseClass
{
    public override void MyMethod(int i = 1)
    {
        // ...
        base.MyMethod(i);
    }
```

Flags enumerations should explicitly initialize all their members

Bug

"GetHashCode" should not reference mutable fields

Bug

Results of integer division should not be assigned to floating point variables

Bug

Integral numbers should not be shifted by zero or more than their number of bits-1

Bug

```
    static int Main(string[] args)
    {
        DerivedClass dc = new DerivedClass();
        dc.MyMethod(12);  // prints 12
    }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
Privacy Policy