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

Empty "default" clauses should be removed
Code Smell

Redundant property names should be omitted in anonymous classes
Code Smell

Declarations and initializations should be as concise as possible
Code Smell

Default parameter values should not be passed as arguments
Code Smell

Constructor and destructor declarations should not be redundant
Code Smell

Method parameters should be declared with base types
Code Smell

The simplest possible condition syntax should be used
Code Smell

Redundant parentheses should not be used
Code Smell

"GC.SuppressFinalize" should not be invoked for types without destructors
Code Smell

Members should not be initialized to default values
Code Smell

Sequential tests should not check the same condition
Code Smell

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

Analyze your code

Bug Minor

`object.Equals()` overrides can be optimized by checking first for reference equality between `this` and the parameter. This check can be implemented by calling `object.ReferenceEquals()` or `base.Equals()`, where `base` is `object`. However, using `base.Equals()` is a maintenance hazard because while it works if you extend `Object` directly, if you introduce a new base class that overrides `Equals`, it suddenly stops working.

This rule raises an issue if `base.Equals()` is used but `base` is not `object`.

Noncompliant Code Example

```
class Base
{
  private int baseField;

  public override bool Equals(object other)
  {
    if (base.Equals(other)) // Okay; base is object
    {
      return true;
    }

    return this.baseField == ((Base)other).baseField;
  }
}

class Derived : Base
{
  private int derivedField;

  public override bool Equals(object other)
  {
    if (base.Equals(other))  // Noncompliant
    {
      return true;
    }

    return this.derivedField == ((Derived)other).derivedFiel
  }
}
```

Compliant Solution

```
class Base
{
  private int baseField;

  public override bool Equals(object other)
  {
```

Redundant modifiers should not be used

Code Smell

Methods and properties that don't access instance data should be static

Code Smell

"Exception" should not be caught when not required by called methods

Code Smell

"sealed" classes should not have "protected" members

Code Smell

```
    if (object.ReferenceEquals(this, other))  // base.Equals
    {
      return true;
    }

    return this.baseField == ((Base)other).baseField;
  }
}

class Derived : Base
{
  private int derivedField;

  public override bool Equals(object other)
  {
    if (object.ReferenceEquals(this, other))
    {
      return true;
    }

    return base.Equals(other) && this.derivedField == ((Deri
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube