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

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

Redundant modifiers should not be used
Code Smell

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

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

Analyze your code

Bug | Minor | Quick Fix

While the properties of a `readonly` reference type field can still be changed after initialization, those of a `readonly` value field, such as a `struct`, cannot.

If the member could be either a `class` or a `struct` then assignment to its properties could be unreliable, working sometimes but not others.

Noncompliant Code Example

```
interface IPoint
{
  int X { get; set; }
  int Y { get; set; }
}

class PointManager<T> where T: IPoint
{
  readonly T point;  // this could be a struct
  public PointManager(T point)
  {
    this.point = point;
  }

  public void MovePointVertically(int newX)
  {
    point.X = newX; //Noncompliant; if point is a struct, th
    Console.WriteLine(point.X);
  }
}
```

Compliant Solution

```
interface IPoint
{
  int X { get; set; }
  int Y { get; set; }
}

class PointManager<T> where T : IPoint
{
  readonly T point;  // this could be a struct
  public PointManager(T point)
  {
    this.point = point;
  }

  public void MovePointVertically(int newX) // assignment ha
  {
    Console.WriteLine(point.X);
```

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

Code Smell

"sealed" classes should not have "protected" members

Code Smell

Underscores should be used to make large numbers readable

Code Smell

"ToString()" calls should not be redundant

Code Smell

```
  }
}
```

or

```
interface IPoint
{
  int X { get; set; }
  int Y { get; set; }
}

class PointManager<T> where T : class, IPoint
{
  readonly T point;  // this can only be a class
  public PointManager(T point)
  {
    this.point = point;
  }

  public void MovePointVertically(int newX)
  {
    point.X = newX;  // this assignment is guaranteed to wor
    Console.WriteLine(point.X);
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube