




 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... 🔍

"protected" members

Code Smell

Underscores should be used to make large numbers readable

Code Smell

"ToString()" calls should not be redundant

Code Smell

"==" should not be used when "Equals" is overridden

Code Smell

An abstract class should have both abstract and concrete methods

Code Smell

Multiple variables should not be declared on the same line

Code Smell

Culture should be specified for "string" operations

Code Smell

"switch" statements should have at least 3 "case" clauses

Code Smell

break statements should not be used except for switch cases

Code Smell

String literals should not be duplicated

Code Smell

Files should contain an empty newline at the end

Code Smell

Unused "using" should be removed

Code Smell

## Interfaces should not simply inherit from base interfaces with colliding members

Analyze your code

Code Smell Minor ? design

When an interface inherits from two interfaces that both define a member with the same name, trying to access that member through the derived interface will result in the compiler error CS0229 Ambiguity between 'IBase1.SomeProperty' and 'IBase2.SomeProperty'.

So instead, every caller will be forced to cast instances of the derived interface to one or the other of its base interfaces to resolve the ambiguity and be able to access the member. Instead, it is better to resolve the ambiguity in the definition of the derived interface either by:

- renaming the member in one of the base interfaces to remove the collision
- also defining that member in the derived interface. Use this only if all copies of the member are meant to hold the same value.

### Noncompliant Code Example

```
public interface IBase1
{
    string SomeProperty { get; set; }
}

public interface IBase2
{
    string SomeProperty { get; set; }
}

public interface IDerived : IBase1, IBase2 // Noncompliant,
{
}

public class MyClass : IDerived
{
    // Implements both IBase1.SomeProperty and IBase2.SomeProp
    public string SomeProperty { get; set; } = "Hello";

    public static void Main()
    {
        MyClass myClass = new MyClass();
        Console.WriteLine(myClass.SomeProperty); // Writes "Hell
        Console.WriteLine(((IBase1)myClass).SomeProperty); // Wr
        Console.WriteLine(((IBase2)myClass).SomeProperty); // Wr
        Console.WriteLine(((IDerived)myClass).SomeProperty); //
    }
}
```

### Compliant Solution

```
public interface IDerived : IBase1, IBase2
{
}
```

A close curly brace should be located at the beginning of a line

 Code Smell

Tabulation characters should not be used

 Code Smell

Methods and properties should be named in PascalCase

 Code Smell

Track uses of in-source issue suppressions

 Code Smell

```
new string SomeProperty { get; set; }
}

public class MyClass : IDerived
{
    // Implements IBasel.SomeProperty, IBase2.SomeProperty and
    public string SomeProperty { get; set; } = "Hello";

    public static void Main()
    {
        MyClass myClass = new MyClass();
        Console.WriteLine(myClass.SomeProperty); // Writes "Hell
        Console.WriteLine(((IBasel)myClass).SomeProperty); // Wr
        Console.WriteLine(((IBase2)myClass).SomeProperty); // Wr
        Console.WriteLine(((IDerived)myClass).SomeProperty); //
    }
}
```

or

```
public interface IBasel
{
    string SomePropertyOne { get; set; }
}

public interface IBase2
{
    string SomePropertyTwo { get; set; }
}

public interface IDerived : IBasel, IBase2
{
}
```

Available In:

 |  | 