sonar RULES

Products

- 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...

Child class fields should not differ from parent class fields only by capitalization
Code Smell

Pointers to unmanaged memory should not be visible
Code Smell

Number patterns should be regular
Code Smell

"out" and "ref" parameters should not be used
Code Smell

Unchanged local variables should be "const"
Code Smell

"ConfigureAwait(false)" should be used
Code Smell

"interface" instances should not be cast to concrete types
Code Smell

Literal boolean values should not be used in assertions
Code Smell

Optional parameters should not be used
Code Smell

Public constant members should not be used
Code Smell

Array covariance should not be used
Code Smell

"nameof" should be used

Classes implementing "IEquatable<T>" should be sealed

Analyze your code

Code Smell Major pitfall

When a class implements the `IEquatable<T>` interface, it enters a contract that, in effect, states "I know how to compare two instances of type T or any type derived from T for equality.". However if that class is derived, it is very unlikely that the base class will know how to make a meaningful comparison. Therefore that implicit contract is now broken.

Alternatively `IEqualityComparer<T>` provides a safer interface and is used by collections or `Equals` could be made `virtual`.

This rule raises an issue when an unsealed, `public` or `protected` class implements `IEquitable<T>` and the `Equals` is neither `virtual` nor `abstract`.

Noncompliant Code Example

```
using System;

namespace MyLibrary
{
  public class Base : IEquatable<Base> // Noncompliant
  {
    public bool Equals(Base other)
    {
      if (other == null) { return false; }
      // do comparison of base properties
      return true;
    }

    public override bool Equals(object other)  => Equals(oth
  }

  class A : Base
  {
    public bool Equals(A other)
    {
      if (other == null) { return false; }
      // do comparison of A properties
      return base.Equals(other);
    }

    public override bool Equals(object other)  => Equals(oth
  }

  class B : Base
  {
    public bool Equals(B other)
    {
      if (other == null) { return false; }
      // do comparison of B properties
      return base.Equals(other);
    }
```

Code Smell

Modulus results should not be checked for direct equality

Code Smell

"for" loop increment clauses should modify the loops' counters

Code Smell

"switch" statements should not be nested

Code Smell

Methods and properties should not be too complex

```
      public override bool Equals(object other)  => Equals(oth
    }

    internal class Program
    {
      static void Main(string[] args)
      {
          A a = new A();
          B b = new B();
           Console.WriteLine(a.Equals(b)); // This calls the W
                                           // to be called whi
                                           // a and b are diff
                                           // called and Equal
                                           // different types.
      }
    }
}
```

Compliant Solution

```
using System;

namespace MyLibrary
{
    public sealed class Foo : IEquatable<Foo>
    {
        public bool Equals(Foo other)
        {
            // Your code here
        }
    }
}
```

See

IEqualityComparer<T> Interface

Available In:

sonarlint sonarcloud sonarqube