C# static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C# code

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

Analyze your code

Code Smell Minor cwe suspicious

Using the equality == and inequality != operators to compare two objects generally works. The operators can be overloaded, and therefore the comparison can resolve to the appropriate method. However, when the operators are used on interface instances, then == resolves to reference equality, which may result in unexpected behavior if implementing classes override `Equals`. Similarly, when a class overrides `Equals`, but instances are compared with non-overloaded ==, there is a high chance that value comparison was meant instead of the reference one.

Noncompliant Code Example

```
public interface IMyInterface
{
}

public class MyClass : IMyInterface
{
    public override bool Equals(object obj)
    {
        //...
    }
}

public class Program
{
    public static void Method(IMyInterface instance1, IMyInt
    {
        if (instance1 == instance2) // Noncompliant, will do
        {
            Console.WriteLine("Equal");
        }
    }
}
```

Compliant Solution

```
public interface IMyInterface
{
}

public class MyClass : IMyInterface
{
    public override bool Equals(object obj)
    {
        //...
    }
}

public class Program
{
```

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

```
    public static void Method(IMyInterface instance1, IMyInt
    {
        if (object.Equals(instance1, instance2)) // object.E
        {
            Console.WriteLine("Equal");
        }
    }
}
```

Exceptions

The rule does not report on comparisons of `System.Type` instances and on comparisons inside `Equals` overrides.

It also does not raise an issue when one of the operands is `null` nor when one of the operand is cast to `object` (because in this case we want to ensure reference equality even if some == overload is present).

See

- MITRE, CWE-595 - Comparison of Object References Instead of Object Contents
- MITRE, CWE-597 - Use of Wrong Operator in String Comparison

Available In:

sonarlint | sonarcloud | sonarqube