

- 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

Public constant members should not be used

Code Smell

Array covariance should not be used

Code Smell

"nameof" should be used

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

Code Smell

Control flow statements "if", "switch", "for", "foreach", "while", "do" and "try" should not be nested too deeply

Code Smell

"switch/Select" statements should contain a "default/Case Else" clauses

Code Smell

"if ... else if" constructs should end with "else" clauses

Code Smell

Control structures should use curly braces

Code Smell

Expressions should not be too

"ISerializable" should be implemented correctly

Analyze your code

Code Smell Major pitfall

The `ISerializable` interface is the mechanism to control the type serialization process. If not implemented correctly this could result in an invalid serialization and hard to detect bugs.

This rule raises an issue on types that implement `ISerializable` without following the serialization pattern recommended by Microsoft.

Specifically this rule checks for these problems:

- The `System.SerializableAttribute` attribute is missing.
- Non-serializable fields are not marked with the `System.NonSerializedAttribute` attribute.
- There is no serialization constructor.
- An unsealed type has a serialization constructor that is not `protected`.
- A sealed type has a serialization constructor that is not `private`.
- An unsealed type has a `ISerializable.GetObjectData` that is not both `public` and `virtual`.
- A derived type has a serialization constructor that does not call the base constructor.
- A derived type has a `ISerializable.GetObjectData` method that does not call the base method.
- A derived type has serializable fields but the `ISerializable.GetObjectData` method is not overridden.

Classes which inherit from `Exception` are implementing `ISerializable`. Make sure `[Serializable]` attribute is used and that `ISerializable` is correctly implemented. Even if you don't plan to explicitly serialize the object yourself, it might still require serialization, for instance when crossing the boundary of an `AppDomain`.

Noncompliant Code Example





```
public class Foo : ISerializable // Noncompliant the [Serializable]
{
}
```

or

```
public class Bar
{
}

[Serializable]
public class Foo : ISerializable // Noncompliant the seriali
{
    private readonly Bar bar; // Noncompliant the field is n
}
```

Compliant Solution

Expressions should not be too complex
 Code Smell
ASP.NET HTTP request validation feature should not be disabled
 Vulnerability
Serialization constructors should be secured
 Vulnerability
Calculations should not overflow
 Bug
Floating point numbers should not be tested for equality

```
public class Bar
{
}

[Serializable]
public class Foo : ISerializable
{
    [NonSerialized]
    private readonly Bar bar;

    public Foo()
    {
        // ...
    }

    protected Foo(SerializationInfo info, StreamingContext context)
    {
        // ...
    }

    public virtual void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        // ...
    }
}

[Serializable]
public sealed class SubFoo : Foo
{
    private int val;

    public SubFoo()
    {
        // ...
    }

    private SubFoo(SerializationInfo info, StreamingContext context) : base(info, context)
    {
        // ...
    }

    public override void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        base.GetObjectData(info, context);
        // ...
    }
}
```

Exceptions

- Classes in test projects are not checked.

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 