- 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#

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...

Inappropriate casts should not be made

Code Smell

Constructors should only call non-overridable methods

Code Smell

"GC.Collect" should not be called

Code Smell

Methods should not be empty

Code Smell

Exceptions should not be thrown in finally blocks

Code Smell

Method overrides should not change parameter defaults

Code Smell

Types allowed to be deserialized should be restricted

Vulnerability

Server-side requests should not be vulnerable to forging attacks

Vulnerability

Members should not have conflicting transparency annotations

Vulnerability

"PartCreationPolicyAttribute" should be used with "ExportAttribute"

Bug

"ConstructorArgument" parameters should exist in constructors

Bug

Windows Forms entry points should be marked with STAThread

Getters and setters should access the expected fields

Analyze your code

Bug Critical pitfall

Properties provide a way to enforce encapsulation by providing `public`, `protected` or `internal` methods that give controlled access to `private` fields. However in classes with multiple fields it is not unusual that cut and paste is used to quickly create the needed properties, which can result in the wrong field being accessed by a getter or setter.

This rule raises an issue in any of these cases:

- A setter does not update the field with the corresponding name.
- A getter does not access the field with the corresponding name.

For simple properties it is better to use auto-implemented properties (C# 3.0 or later).

Field and property names are compared as case-insensitive. All underscore characters are ignored.

Noncompliant Code Example

```
class A
{
    private int x;
    private int y;

    public int X
    {
        get { return x; }
        set { x = value; }
    }

    public int Y
    {
        get { return x; }  // Noncompliant: field 'y' is not
        set { x = value; } // Noncompliant: field 'y' is not
    }
}
```

Compliant Solution

```
class A
{
    private int x;
    private int y;

    public int X
    {
        get { return x; }
        set { x = value; }
    }
    public int Y
    {
        get { return y; }
        set { y = value; }
    }
}
```

Available In:

sonarlint sonarcloud sonarqube

Bug

Collection elements should not be replaced unconditionally

Bug

Exceptions should not be created without being thrown

Bug

Collection sizes and array length comparisons should make sense

Bug

Serialization event handlers should be implemented correctly