

- 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... 🔍

Lines should not be too long

Code Smell

HTTP response headers should not be vulnerable to injection attacks

Vulnerability

Console logging should not be used

Vulnerability

Generic parameters not constrained to reference types should not be compared to "null"

Bug

The length returned from a stream read should be checked

Bug

Method parameters, caught exceptions and foreach variables' initial values should not be ignored

Bug

Controlling permissions is security-sensitive

Security Hotspot

Writing cookies is security-sensitive

Security Hotspot

Methods should be named according to their synchronicities

Code Smell

Extensions should be in separate namespaces

Code Smell

Extension methods should not extend "object"

Code Smell

Operator overloads should have named alternatives

Code Smell

### Unused private types or members should be removed

Analyze your code

Code Smell Major Quick Fix unused

private or internal types or private members that are never executed or referenced are dead code: unnecessary, inoperative code that should be removed. Cleaning out dead code decreases the size of the maintained codebase, making it easier to understand the program and preventing bugs from being introduced.

#### Noncompliant Code Example

```
public class Foo
{
    private void UnusedPrivateMethod() {...} // Noncompliant

    private class UnusedClass {...} // Noncompliant
}
```

#### Compliant Solution

```
public class Foo
{
    public Foo()
    {
        UsedPrivateMethod();
    }

    private void UsedPrivateMethod()
    {
        var c = new UsedClass();
    }




    private class UsedClass {...}
}
```

#### Exceptions

This rule doesn't raise issues on:

- Empty constructors
- Attributed members
- Main method
- Methods with event handler signature void Foo(object, EventArgs) that are declared in partial class
- Empty serialization constructor on type with System.SerializableAttribute attribute.
- Internals in assemblies that have a System.Runtime.CompilerServices.InternalsVisibleToAttribute attribute.

Available In:

<b>Non-abstract attributes should be sealed</b>  Code Smell
<b>Overloads with a "StringComparison" parameter should be used</b>  Code Smell
<b>Overloads with a "CultureInfo" or an "IFormatProvider" parameter should be used</b>  Code Smell
<b>Types should not extend outdated base types</b>