

- 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...

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 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

Bug

Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression

Code Smell

Use a testable date/time provider.

Code Smell

Property names should not match get

Composite format strings should be used correctly

Analyze your code

Code Smell Major ? confusing

Because composite format strings are interpreted at runtime, rather than validated by the compiler, they can contain errors that lead to unexpected behaviors or runtime errors. This rule statically validates the good behavior of composite formats when calling the methods of `String.Format`, `StringBuilder.AppendFormat`, `Console.Write`, `Console.WriteLine`, `TextWriter.Write`, `TextWriter.WriteLine`, `Debug.WriteLine(String, Object[])`, `Trace.TraceError(String, Object[])`, `Trace.TraceInformation(String, Object[])`, `Trace.TraceWarning(String, Object[])` and `TraceSource.TraceInformation(String, Object[])`.

Noncompliant Code Example

```
s = string.Format("{0}", arg0, arg1); // Noncompliant, arg1
s = string.Format("{0} {2}", arg0, arg1, arg2); // Noncompliant, arg2
s = string.Format("foo"); // Noncompliant, there is no need
```

Compliant Solution

```
s = string.Format("{0}", arg0);
s = string.Format("{0} {1}", arg0, arg2);
s = "foo";
```

Exceptions

- No issue is raised if the format string is not a const.

```
var pattern = "{0} {1} {2}";
var res = string.Format(pattern, 1, 2); // Compliant, not co
```





- No issue is raised if the argument is not an inline creation array.

```
var array = new int[] {};
var res = string.Format("{0} {1}", array); // Compliant we d
```

- This rule doesn't check whether the format specifier (defined after the `:`) is actually valid.

Available In:

sonarlint | sonarcloud | sonarqube

Property names should not match get methods  Code Smell
Locales should be set for data types  Code Smell
Literals should not be passed as localized parameters  Code Smell
Operators should be overloaded consistently  Code Smell
Method signatures should not contain