

- 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... 🔍

Security Hotspot

Setting loose file permissions is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Using hardcoded IP addresses is security-sensitive

Security Hotspot

"goto" statement should not be used

Code Smell

"new Guid()" should not be used

Code Smell

Parameter validation in "async"/"await" methods should be wrapped

Code Smell

Parameter validation in yielding methods should be wrapped

Code Smell

Events should have proper arguments

Code Smell

"P/Invoke" methods should not be visible

Code Smell

Native methods should be wrapped

Code Smell

Methods should not have identical implementations

Code Smell

Non-flags enums should not be marked with "FlagsAttribute"

"IndexOf" checks should not be for positive numbers

Analyze your code

Code Smell Critical ? suspicious

Most checks against an `IndexOf` value compare it with `-1` because `0` is a valid index. Any checks which look for values `> 0` ignore the first element, which is likely a bug. If the intent is merely to check inclusion of a value in a `string`, `List`, or an array, consider using the `Contains` method instead.

This rule raises an issue when an `IndexOf` value retrieved from a `string`, `List` or array is tested against `> 0`.

This rule also raises an issue when `IndexOfAny`, `LastIndexOf` or `LastIndexOfAny` from a `string` is tested against `> 0`

Noncompliant Code Example

```
string color = "blue";
string name = "ishmael";

List<string> strings = new List<string>();
strings.Add(color);
strings.Add(name);
string[] stringArray = strings.ToArray();

if (strings.IndexOf(color) > 0) // Noncompliant
{
    // ...
}

if (name.IndexOf("ish") > 0) // Noncompliant
{
    // ...
}





if (name.IndexOf("ae") > 0) // Noncompliant
{
    // ...
}

if (Array.IndexOf(stringArray, color) > 0) // Noncompliant
{
    // ...
}
```

Compliant Solution

```
string color = "blue";
string name = "ishmael";

List<string> strings = new List<string> ();
strings.Add(color);
strings.Add(name);
string[] stringArray = strings.ToArray();
```

 Code Smell
Classes implementing "IEquatable<T>" should be sealed
 Code Smell
"GC.SuppressFinalize" should not be called
 Code Smell
Objects should not be disposed more than once
 Code Smell
Parameter names used into ArgumentException constructors

```
if (strings.IndexOf(color) > -1)
{
    // ...
}

if (name.IndexOf("ish") >= 0)
{
    // ...
}

if (name.Contains("ae"))
{
    // ...
}

if (Array.IndexOf(stringArray, color) >= 0)
{
    // ...
}
```

Available In:

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