

- 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

Track lack of copyright and license headers

Code Smell

Exit methods should not be called

Code Smell

Classes should "Dispose" of members from the classes' own "Dispose" methods

Bug

Reading the Standard Input is security-sensitive

Security Hotspot

Using command line arguments is security-sensitive

Security Hotspot

Using Sockets is security-sensitive

Security Hotspot

Encrypting data is security-sensitive

Security Hotspot

Using regular expressions is security-sensitive

Security Hotspot

Interface methods should be callable by derived types

Code Smell

Child class fields should not differ from parent class fields only by capitalization

Code Smell

Pointers to unmanaged memory should not be visible

Code Smell

Number patterns should be regular

Parameter validation in yielding methods should be wrapped

Analyze your code

Code Smell Major ? yield

Because of the way `yield` methods are rewritten by the compiler (they become lazily evaluated state machines) any exceptions thrown during the parameters check will happen only when the collection is iterated over. That could happen far away from the source of the buggy code.

Therefore it is recommended to split the method into two: an outer method handling the validation (no longer lazy) and an inner (lazy) method to handle the iteration.

This rule raises an issue when a method throws any exception derived from `ArgumentException` and contains the `yield` keyword.

Noncompliant Code Example

```
public static IEnumerable<TSource> TakeWhile<TSource>(this I
{
    if (source == null) { throw new ArgumentNullException(na
    if (predicate == null) { throw new ArgumentNullException

    foreach (var element in source)
    {
        if (!predicate(element)) { break; }
        yield return element;
    }
}
```





Compliant Solution

```
public static IEnumerable<TSource> TakeWhile<TSource>(this I
{
    if (source == null) { throw new ArgumentNullException(na
    if (predicate == null) { throw new ArgumentNullException
    return TakeWhileIterator<TSource>(source, predicate);
}

private static IEnumerable<TSource> TakeWhileIterator<TSource
{
    foreach (TSource element in source)
    {
        if (!predicate(element)) break;
        yield return element;
    }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

 Code Smell
"out" and "ref" parameters should not be used  Code Smell
Unchanged local variables should be "const"  Code Smell
"ConfigureAwait(false)" should be used  Code Smell
"interface" instances should not be cast to concrete types