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

Non-flags enums should not be marked with "FlagsAttribute"
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 should match an existing one
Code Smell

"ISerializable" should be implemented correctly
Code Smell

"Assembly.Load" should be used
Code Smell

"IDisposable" should be implemented correctly
Code Smell

"ServiceContract" and "OperationContract" attributes should be used together
Code Smell

Composite format strings should be used correctly
Code Smell

Exceptions should not be explicitly rethrown
Code Smell

Overflow checking should not be disabled for "Enumerable.Sum"

Analyze your code

Code Smell | Critical | error-handling

`Enumerable.Sum()` always executes addition in a `checked` context, so an `OverflowException` will be thrown if the value exceeds `MaxValue` even if an `unchecked` context was specified. Using an `unchecked` context anyway represents a misunderstanding of how `Sum` works.

This rule raises an issue when an `unchecked` context is specified for a `Sum` on integer types.

Noncompliant Code Example

```
void Add(List<int> list)
{
  int d = unchecked(list.Sum());  // Noncompliant

  unchecked
  {
    int e = list.Sum();  // Noncompliant
  }
}
```

Compliant Solution

```
void Add(List<int> list)
{
  int d = list.Sum();

  try
  {
    int e = list.Sum();
  }
  catch (System.OverflowException e)
  {
    // exception handling...
  }
}
```

Exceptions

When the `Sum()` call is inside a `try-catch` block, no issues are reported.

```
void Add(List<int> list)
{
  unchecked
  {
    try
    {
      int e = list.Sum();
    }
```

```
    catch (System.OverflowException e)
    {
      // exception handling...
    }
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

"abstract" classes should not have "public" constructors

Code Smell

Assertion arguments should be passed in the correct order

Code Smell

Ternary operators should not be nested

Code Smell

Events should be invoked

Code Smell