

- 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...



"protected" members

Code Smell

Underscores should be used to make large numbers readable

Code Smell

"ToString()" calls should not be redundant

Code Smell

"==" should not be used when "Equals" is overridden

Code Smell

An abstract class should have both abstract and concrete methods

Code Smell

Multiple variables should not be declared on the same line

Code Smell

Culture should be specified for "string" operations

Code Smell

"switch" statements should have at least 3 "case" clauses

Code Smell

break statements should not be used except for switch cases

Code Smell

String literals should not be duplicated

Code Smell

Files should contain an empty newline at the end

Code Smell

Unused "using" should be removed

Code Smell

Runtime type checking should be simplified

Analyze your code

Code Smell Minor Quick Fix clumsy

To check the type of an object there are several options:

- `expr is SomeType` or `expr.GetType() == typeof(SomeType)` if the type is known at compile time,
- `typeInstance.IsInstanceOfType(expr)` if the type is calculated during runtime.

If runtime calculated Types need to be compared:

- `typeInstance1.IsAssignableFrom(typeInstance2)`.

Depending on whether the type is returned by a `GetType()` or `typeof()` call, the `IsAssignableFrom()` and `IsInstanceOfType()` might be simplified. Similarly, if the type is sealed, the type comparison with `==` can be converted to an `is` call. Simplifying the calls also make `null` checking unnecessary because both `is` and `IsInstanceOfType` performs it already.

Finally, utilizing the most concise language constructs for type checking makes the code more readable, so

- `expr as T != null` checks should be simplified to `expr is T`, and
- `expr is T` should be converted to `expr != null`, when `expr` is of type `T`.

Noncompliant Code Example

```
class Fruit { }
sealed class Apple : Fruit { }

class Program
{
    static void Main()
    {
        var apple = new Apple();
        var b = apple != null && apple.GetType() == typeof(Apple); // Noncompliant
        if (apple != null)
        {
            b = typeof(Apple).IsAssignableFrom(apple.GetType()); // Noncompliant
        }
        var appleType = typeof(Apple);
        if (apple != null)
        {
            b = appleType.IsAssignableFrom(apple.GetType()); // Noncompliant
        }

        Fruit f = apple;
        if (f as Apple != null) // Noncompliant
        {
        }
        if (apple is Apple) // Noncompliant
        {
        }
    }
}
```

A close curly brace should be located at the beginning of a line

 Code Smell


Tabulation characters should not be used

 Code Smell

Methods and properties should be named in PascalCase

 Code Smell

Track uses of in-source issue suppressions

 Code Smell

```
}  
}
```

Compliant Solution

```
class Fruit { }  
sealed class Apple : Fruit { }  
  
class Program  
{  
    static void Main()  
    {  
        var apple = new Apple();  
        var b = apple is Apple;  
        b = apple is Apple;  
        b = apple is Apple;  
        var appleType = typeof(Apple);  
        b = appleType.IsInstanceOfType(apple);  
  
        Fruit f = apple;  
        if (f is Apple)  
        {  
        }  
        if (apple != null)  
        {  
        }  
    }  
}
```

Exceptions

Calling `GetType` on an object of `Nullable<T>` type returns the underlying generic type parameter `T`, thus a comparison with `typeof(Nullable<T>)` can't be simplified to use the `is` operator, which doesn't make difference between `T` and `T?`.

```
int? i = 42;  
bool condition = i.GetType() == typeof(int?); // false;  
condition = i is int?; // true
```

No issue is reported on the following expressions:

- `expr is T` when either operand of the `is` operator is a value type. In that case CS0183 or CS0184 reports
- `expr is object`, as this is a common and efficient pattern to do null checks

Available In:

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