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...

Members should not be initialized to default values
Code Smell

Sequential tests should not check the same condition
Code Smell

Redundant modifiers should not be used
Code Smell

Methods and properties that don't access instance data should be static
Code Smell

"Exception" should not be caught when not required by called methods
Code Smell

"sealed" classes should not have "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

Integral numbers should not be shifted by zero or more than their number of bits-1

Analyze your code

Bug Minor

Shifting an integral number by 0 is equivalent to doing nothing but makes the code confusing for maintainers.

If the first operand is an `int` or `uint` (32-bit quantity), the shift count is given by the low-order five bits of the second operand. That is, the actual shift count is 0 to 31 bits.

Note that integral number with a less than 32-bit quantity (e.g. `short`, `ushort`...) are implicitly converted to `int` before the shifting operation and so the rule for `int`/`uint` applies.

If the first operand is a `long` or `ulong` (64-bit quantity), the shift count is given by the low-order six bits of the second operand. That is, the actual shift count is 0 to 63 bits.

Noncompliant Code Example

```
public void Main()
{
    short s = 1;
    short shortShift1 = (short)(s << 0); // Noncompliant
    short shortShift1 = (short)(s << 16); // Compliant as sh
    short shortShift3 = (short)(s << 32); // Noncompliant, t

    int i = 1;
    int intShift1 = i << 0; // Noncompliant
    int intShift2 = i << 32; // Noncompliant, this is equiva

    long lg = 1;
    long longShift1 = lg << 0; // Noncompliant
    long longShift2 = lg << 64; // Noncompliant, this is equ
}
```

Compliant Solution

```
public void Main()
{
    short s = 1;
    short shortShift1 = s;
    short shortShift1 = (short)(s << 16);
    short shortShift3 = (short)(s << 1);

    int i = 1;
    var intShift1 = i;
    var intShift2 = i << 1;

    long lg = 1;
    var longShift1 = lg;
    var longShift2 = lg << 1;
}
```

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

Exceptions

This rule doesn't raise an issue when the shift by zero is obviously for cosmetic reasons:

- When the value shifted is a literal.
- When there is a similar shift at the same position on line before or after. E.g.:

```
bytes[loc+0] = (byte)(value >> 8);
bytes[loc+1] = (byte)(value >> 0);
```

See

- Microsoft documentation - Bitwise and shift operators

Available In:

sonarlint | sonarcloud | sonarqube