

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C# 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... 🔍

multidimensional array parameters

Code Smell

"async" and "await" should not be used as identifiers

Code Smell

TestCases should contain tests

Code Smell

Short-circuit logic should be used in boolean contexts

Code Smell

JWT should be signed and verified with strong cipher algorithms

Vulnerability

Cipher algorithms should be robust

Vulnerability

Encryption algorithms should be used with secure mode and padding scheme

Vulnerability

Insecure temporary file creation methods should not be used

Vulnerability

Server certificates should be verified during SSL/TLS connections

Vulnerability

LDAP connections should be authenticated

Vulnerability

Cryptographic keys should be robust

Vulnerability

Weak SSL/TLS protocols should not be used

Vulnerability

Recursion should not be infinite

Analyze your code

Bug Blocker ? suspicious

Recursion happens when control enters a loop that has no exit. This can happen a method invokes itself, when a pair of methods invoke each other, or when goto statements are used to move between two segments of code. It can be a useful tool, but unless the method includes a provision to break out of the recursion and return, the recursion will continue until the stack overflows and the program crashes.

Noncompliant Code Example

```
int Pow(int num, int exponent) // Noncompliant; no condition
{
    num = num * Pow(num, exponent-1);
    return num; // this is never reached
}

void WhileLoop() // Noncompliant; no condition under which
{
    while (true)
    {
        var line = Console.ReadLine();
        Console.WriteLine(line);
    }
}

void InternalRecursion(int i)
{
    start:
        goto end;
    end:
        goto start; // Noncompliant; there's no way to break out
}
```

Compliant Solution

```
int Pow(int num, int exponent)
{
    if (exponent > 1) // recursion now conditional and stop-ab
    {
        num = num * Pow(num, exponent-1);
    }
    return num;
}

void WhileLoop()
{
    string line;
    while ((line = Console.ReadLine()) != null) // loop has cl
    {
        Console.WriteLine(line);
    }
}
```

Cipher Block Chaining IVs should be unpredictable

 Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks

 Vulnerability

Hashes should include an unpredictable salt

 Vulnerability

Non-async "Task/Task<T>" methods should not return null

 Bug

```
}  
}
```

Available In:

**sonarcloud** 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)