

- 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

Inherited member visibility should not be decreased

Code Smell

Threads should not lock on objects with weak identity

Code Smell

A conditionally executed single line should be denoted by indentation

Code Smell

Conditionals should start on new lines

Code Smell

Assemblies should have version information

Code Smell

Exception types should be "public"

Code Smell

Cognitive Complexity of methods should not be too high

Code Smell

"params" should not be introduced on overrides

Code Smell

"[DefaultValue]" should not be used when "[DefaultParameterValue]" is meant

Code Smell

"[Optional]" should not be used on "ref" or "out" parameters

Code Smell

Non-flags enums should not be used in bitwise operations

Code Smell

Inner class members should not

### Cipher algorithms should be robust

Analyze your code

Vulnerability Critical cwe privacy owasp sans-top25

**Strong cipher algorithms** are cryptographic systems resistant to cryptanalysis, they are not vulnerable to well-known attacks like brute force attacks for example.

A general recommendation is to only use cipher algorithms intensively tested and promoted by the cryptographic community.

More specifically for block cipher, it's not recommended to use algorithm with a block size inferior than 128 bits.

#### Noncompliant Code Example

For [System.Security.Cryptography](#) library, these old cryptographic algorithms should no longer be used for any reason:

```
var tripleDES1 = new TripleDESCryptoServiceProvider(); // Noncompliant
var simpleDES = new DESCryptoServiceProvider(); // Noncompliant
var RC2 = new RC2CryptoServiceProvider(); // Noncompliant: R
```

For Bouncycastle library, [AESFastEngine has a side channel leak](#), it is possible to gain information about the key used to initialize the cipher:

```
AesFastEngine aesFast = new AesFastEngine(); // Noncompliant
```

#### Compliant Solution

For [System.Security.Cryptography](#) library, it's recommended to use `AesCryptoServiceProvider`:

```
var AES = new AesCryptoServiceProvider(); // Compliant
```

For Bouncycastle library, it's recommended to use `AESEngine`:

```
var AES = new AESEngine(); // Compliant
```

#### See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [MITRE, CWE-327](#) - Use of a Broken or Risky Cryptographic Algorithm
- [SANS Top 25](#) - Porous Defenses

Available In:

sonarlint | sonarcloud | sonarqube

inner class members should not shadow outer class "static" or type members

 Code Smell


"Explicit" conversions of "foreach" loops should not be used

 Code Smell

Instance members should not write to "static" fields

 Code Smell

"IndexOf" checks should not be for positive numbers

 Code Smell

---

© 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](#)