

- 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

### Classes should "Dispose" of members from the classes' own "Dispose" methods

Analyze your code

Bug Critical cwe denial-of-service

It is possible in an `IDisposable` to call `Dispose` on class members from any method, but the contract of `Dispose` is that it will clean up all unmanaged resources. Move disposing of members to some other method, and you risk resource leaks.

This rule also applies for disposable ref structs.

#### Noncompliant Code Example

```
public class ResourceHolder : IDisposable
{
    private FileStream fs;
    public void OpenResource(string path)
    {
        this.fs = new FileStream(path, FileMode.Open);
    }
    public void CloseResource()
    {
        this.fs.Close();
    }





    public void CleanUp()
    {
        this.fs.Dispose(); // Noncompliant; Dispose not called i
    }

    public void Dispose()
    {
        // method added to satisfy demands of interface
    }
}
```

#### Compliant Solution

```
public class ResourceHolder : IDisposable
{
    private FileStream fs;
    public void OpenResource(string path)
    {
        this.fs = new FileStream(path, FileMode.Open);
    }
    public void CloseResource()
    {
        this.fs.Close();
    }

    public void Dispose()
    {
        this.fs.Dispose();
    }
}
```

<b>A close curly brace should be located at the beginning of a line</b>
 Code Smell
<b>Tabulation characters should not be used</b>
 Code Smell
<b>Methods and properties should be named in PascalCase</b>
 Code Smell
<b>Track uses of in-source issue suppressions</b>
 Code Smell

```
}  
}
```

See

- [MITRE, CWE-459](#) - Incomplete Cleanup

Available In:

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