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...

Bug

Nullable type comparison should not be redundant
Bug

Methods with "Pure" attribute should return a value
Bug

One-way "OperationContract" methods should have "void" return type
Bug

Optional parameters should be passed to "base" calls
Bug

Classes should not have only "private" constructors
Bug

Expressions used in "Debug.Assert" should not produce side effects
Bug

Caller information parameters should come at the end of the parameter list
Bug

Static fields should appear in the order they must be initialized
Bug

Classes directly extending "object" should not call "base" in "GetHashCode" or "Equals"
Bug

Anonymous delegates should not be used to unsubscribe from Events
Bug

Delegates should not be subtracted
Bug

Locks should be released

Analyze your code

Bug | Critical | cwe multi-threading

If a lock is known to be held or acquired, and then released within a method, then it must be released along all execution paths of that method.

Failing to do so will expose the conditional locking logic to the method's callers and hence be deadlock-prone.

The types tracked by the rule are: `Monitor`, `Mutex`, `ReaderWriterLock`, `ReaderWriterLockSlim` and `SpinLock` from the `System.Threading` namespace.

Noncompliant Code Example

```
class MyClass
{
  private object obj = new object();

  public void DoSomethingWithMonitor()
  {
    Monitor.Enter(obj); // Noncompliant
    if (IsInitialized())
    {
      // ...
      Monitor.Exit(obj);
    }
  }

  private ReaderWriterLockSlim lockObj = new ReaderWriterLoc

  public void DoSomethingWithReaderWriteLockSlim()
  {
    lockObj.EnterReadLock(); // Noncompliant
    if (IsInitialized())
    {
      // ...
      lockObj.ExitReadLock();
    }
  }
}
```

Compliant Solution

```
class MyClass
{
  private object obj = new object();

  public void DoSomethingWithMonitor()
  {
    lock(obj) // lock() {...} is easier to use than explicit
    {
      if (IsInitialized())
```

"async" methods should not return "void"

Bug

"ThreadStatic" should not be used on non-static fields

Bug

"IDisposables" created in a "using" statement should not be returned

Bug

"ThreadStatic" fields should not be initialized

Bug

```
      {
      }
    }
  }

  private ReaderWriterLockSlim lockObj = new ReaderWriterLoc

  public void DoSomethingWithReaderWriteLockSlim()
  {
    lockObj.EnterReadLock();
    try
    {
      if (IsInitialized())
      {
      }
    }
    finally
    {
      lockObj.ExitReadLock();
    }
  }
}
```

See

- MITRE, CWE-459 - Incomplete Cleanup
- Synchronization of access to a shared resource.

Available In:

sonarlint | sonarcloud | sonarqube