

- 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
	Type should not be examined on "System.Type" instances
	Code Smell
	Test method signatures should be correct
	Code Smell
	Method overloads with default parameter values should not overlap
	Code Smell
	"value" parameters should be used
	Code Smell
	"is" should not be used with "this"
	Code Smell
	Methods named "Dispose" should implement "IDisposable.Dispose"
	Code Smell
	Tests should include assertions
	Code Smell
	Silly bit operations should not be performed
	Code Smell
	Public methods should not have 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

"SafeHandle.DangerousGetHandle" should not be called

Analyze your code





Bug Blocker ? leak unpredictable

Not surprisingly, the `SafeHandle.DangerousGetHandle` method is dangerous. That's because it may not return a valid handle. Using it can lead to leaks and vulnerabilities. While it is possible to use the method successfully, it's extremely difficult to do correctly, so the method should simply be avoided altogether.

Noncompliant Code Example

```
static void Main(string[] args)
{
    System.Reflection.FieldInfo fieldInfo = ...;
    SafeHandle handle = (SafeHandle)fieldInfo.GetValue(rKey)
    IntPtr dangerousHandle = handle.DangerousGetHandle(); /
}
```

Available In:
sonarlint | **sonarcloud** | **sonarqube**

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