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

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

Cipher Block Chaining IVs should be unpredictable
Vulnerability

Regular expressions should not be

Composite format strings should not lead to unexpected behavior at runtime

Analyze your code

Bug | Blocker

Because composite format strings are interpreted at runtime, rather than validated by the compiler, they can contain errors that lead to unexpected behaviors or runtime errors. This rule statically validates the good behavior of composite formats when calling the methods of `String.Format`, `StringBuilder.AppendFormat`, `Console.Write`, `Console.WriteLine`, `TextWriter.Write`, `TextWriter.WriteLine`, `Debug.WriteLine(String, Object[])`, `Trace.TraceError(String, Object[])`, `Trace.TraceInformation(String, Object[])`, `Trace.TraceWarning(String, Object[])` and `TraceSource.TraceInformation(String, Object[])`.

Noncompliant Code Example

```
s = string.Format("[0}", arg0);
s = string.Format("{{0}", arg0);
s = string.Format("{0}}", arg0);
s = string.Format("{-1}", arg0);
s = string.Format("{0} {1}", arg0);
```

Compliant Solution

```
s = string.Format("{0}", 42); // Compliant
s = string.Format("{0,10}", 42); // Compliant
s = string.Format("{0,-10}", 42); // Compliant
s = string.Format("{0:0000}", 42); // Compliant
s = string.Format("{2}-{0}-{1}", 1, 2, 3); // Compliant
s = string.Format("no format"); // Compliant
```

Exceptions

- No issue is raised if the format string is not a `const`.

```
var pattern = "{0} {1} {2}";
var res = string.Format(pattern, 1, 2); // Compliant, not co
```

- No issue is raised if the argument is not an inline creation array.

```
var array = new int[] {};
var res = string.Format("{0} {1}", array); // Compliant we d
```

- This rule doesn't check whether the format specifier (defined after the `:`) is actually valid.

Available In:

sonarlint | sonarcloud | sonarqube

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

Calls to delegate's method "BeginInvoke" should be paired with calls to "EndInvoke"

Bug