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...

Enumerations should have "Int32" storage
Code Smell

Generic methods should provide type parameters
Code Smell

Multidimensional arrays should not be used
Code Smell

"static readonly" constants should be "const" instead
Code Smell

Strings or integral types should be used for indexers
Code Smell

Parameter names should not duplicate the names of their methods
Code Smell

Track use of "NotImplementedException"
Code Smell

Empty "default" clauses should be removed
Code Smell

Redundant property names should be omitted in anonymous classes
Code Smell

Declarations and initializations should be as concise as possible
Code Smell

Default parameter values should not be passed as arguments
Code Smell

Constructor and destructor

Logging should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Minor | injection cwe owasp sans-top25

User-provided data, such as URL parameters, POST data payloads or cookies, should always be considered untrusted and tainted. Applications logging tainted data could enable an attacker to inject characters that would break the log file pattern. This could be used to block monitors and SIEM (Security Information and Event Management) systems from detecting other malicious events.

This problem could be mitigated by sanitizing the user-provided data before logging it.

Noncompliant Code Example

```
using System;
using Microsoft.AspNetCore.Mvc;


namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC5145LogInjectionLog4NetNoncompliantCon
    {
        private static readonly log4net.ILog _logger = log4n

        public IActionResult Index()
        {
            return View();
        }

        public void LogSomething(string id)
        {
            if (id != null)
            {
                _logger.Info("ID: " + id);
            }
        }
    }
}
```

Compliant Solution

```
using System;
using Microsoft.AspNetCore.Mvc;

namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC5145LogInjectionLog4NetCompliantContro
    {
        private static readonly log4net.ILog _logger = log4n

        public IActionResult Index()
        {
```

declarations should not be redundant

Code Smell

Method parameters should be declared with base types

Code Smell

The simplest possible condition syntax should be used

Code Smell

Redundant parentheses should not be used

Code Smell

"GC.SuppressFinalize" should not be

```
            return View();
        }

        public void LogSomething(string id)
        {

            if (id != null)
            {
                // Replace pattern-breaking characters
                id = id.Replace('\n', '_').Replace('\r', '_'
                _logger.Info("ID: " + id);
            }
        }
    }
}
```

See

- OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
- OWASP Cheat Sheet - Logging
- OWASP Attack Category - Log Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-117 - Improper Output Neutralization for Logs
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud | sonarqube Developer Edition