

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C# 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... 🔍

"IDisposableables" should be disposed



SQL keywords should be delimited by whitespace



Composite format strings should not lead to unexpected behavior at runtime



Recursion should not be infinite



Destructors should not throw exceptions



Hard-coded credentials are security-sensitive



Exceptions should not be thrown from unexpected methods



"operator==" should not be overloaded on reference types



Type should not be examined on "System.Type" instances



Test method signatures should be correct



Method overloads with default parameter values should not overlap



"value" parameters should be used



XPath expressions should not be vulnerable to injection attacks

Analyze your code

Vulnerability Blocker injection cwe owasp

User-provided data, such as URL parameters, should always be considered untrusted and tainted. Constructing XPath expressions directly from tainted data enables attackers to inject specially crafted values that changes the initial meaning of the expression itself. Successful XPath injection attacks can read sensitive information from XML documents.

Noncompliant Code Example

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

namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2091XPathInjectionNoncompliant : ControllerBase
    {
        public XmlDocument doc { get; set; }

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

        public IActionResult Authenticate(string user, string password)
        {
            String expression = "/users/user[@name='" + user + "']";

            // An attacker can bypass authentication by setting user = '' or 1=1 or ''='';
            return Content(doc.SelectSingleNode(expression).InnerText);
        }
    }
}
```

Compliant Solution

```
using System;
using System.Text.RegularExpressions;
using System.Xml;
using Microsoft.AspNetCore.Mvc;


namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2091XPathInjectionCompliant : ControllerBase
    {
        public XmlDocument doc { get; set; }

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


        public IActionResult Authenticate(string user, string password)
        {
            String expression = "/users/user[@name='" + user + "']";

            // An attacker can bypass authentication by setting user = '' or 1=1 or ''='';
            return Content(doc.SelectSingleNode(expression).InnerText);
        }
    }
}
```


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

public IActionResult Authenticate(string user, string password)
{
    // Restrict the username and password to letters
    if (!Regex.IsMatch(user, "^[a-zA-Z]+$") || !Regex.IsMatch(password, "^[a-zA-Z]+$"))
    {
        return BadRequest();
    }

    String expression = "/users/user[@name='" + user + "']";
    return Content(doc.SelectSingleNode(expression).InnerText);
}
}
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-643](#) - Improper Neutralization of Data within XPath Expressions

Available In:

sonarcloud  | **sonarqube**  Developer Edition