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...

Exceptions should not be thrown from unexpected methods
Code Smell

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

## OS commands should not be vulnerable to command injection attacks

Analyze your code

Vulnerability | Blocker | injection cwe owasp sans-top25

Applications that allow execution of operating system commands from user-controlled data should control the command to execute, otherwise an attacker can inject arbitrary commands that will compromise the underlying operating system.

The mitigation strategy can be based on a list of authorized and safe commands to execute and when a shell is spawned to sanitize shell meta-characters. Keep in mind that when a single argument to the command is user-controlled and shell-metachars are sanitized, it can still lead to vulnerabilities if the attacker can inject a dangerous option supported by the command, such as `-exec` available with find, in that case, mark end of option processing on the command line using `--` (double-dash) or restrict options to only trusted values.

### Noncompliant Code Example

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

namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2076OSCommandInjectionNoncompliantCont
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Run(string binary)
        {
            // If the value "/sbin/shutdown" is passed as bi
            // then the machine running the web server will

            Process p = new Process();
            p.StartInfo.FileName = binary; // Noncompliant
            p.StartInfo.RedirectStandardOutput = true;
            p.Start();
            string output = p.StandardOutput.ReadToEnd();
            p.Dispose();

            return View();
        }
    }
}
```

### Compliant Solution

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

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

```
namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2076OSCommandInjectionCompliantControl
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Run(string binary)
        {
            if (binary.Equals("/usr/bin/ls") || binary.Equal
            {
               // only ls and cat commands are authorized

              Process p = new Process();
              p.StartInfo.FileName = binary; // Compliant
              p.StartInfo.RedirectStandardOutput = true;
              p.Start();
              string output = p.StandardOutput.ReadToEnd();
              p.Dispose();
            }

            return View();
        }
    }
}
```

**See**

- OWASP Top 10 2021 Category A3 - Injection
- OWASP OS Command Injection Defense Cheat Sheet
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-78 - Improper Neutralization of Special Elements used in an OS Command
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud | sonarqube Developer Edition