

- 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... 🔍

Inheritance list should not be redundant

Code Smell

Redundant casts should not be used

Code Smell

Strings should not be concatenated using '+' in a loop

Code Smell

Unused local variables should be removed

Code Smell

Private fields only used as local variables in methods should become local variables

Code Smell

A "while" loop should be used instead of a "for" loop

Code Smell

"Equals" and the comparison operators should be overridden when implementing "IComparable"

Code Smell

Nested code blocks should not be used

Code Smell

Overriding members should do more than simply call the same member in the base class

Code Smell

"Any()" should be used to test for emptiness

Code Smell

Boolean literals should not be redundant

Code Smell

Empty statements should be removed

### Constructing arguments of system commands from user input is security-sensitive

Analyze your code

Security Hotspot Major injection cwe owasp sans-top25

Constructing arguments of system commands from user input is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2016-9920
- CVE-2021-29472

Arguments of system commands are processed by the executed program. The arguments are usually used to configure and influence the behavior of the programs. Control over a single argument might be enough for an attacker to trigger dangerous features like executing arbitrary commands or writing files into specific directories.

#### Ask Yourself Whether

- Malicious arguments can result in undesired behavior in the executed command.
- Passing user input to a system command is not necessary.

There is a risk if you answered yes to any of those questions.

#### Recommended Secure Coding Practices

- Avoid constructing system commands from user input when possible.
- Ensure that no risky arguments can be injected for the given program, e.g., type-cast the argument to an integer.
- Use a more secure interface to communicate with other programs, e.g., the standard input stream (stdin).

#### Sensitive Code Example






Arguments like `-delete` or `-exec` for the `find` command can alter the expected behavior and result in vulnerabilities:

```
using System.Diagnostics;
Process p = new Process();
p.StartInfo.FileName = "/usr/bin/find";
p.StartInfo.ArgumentList.Add(input); // Sensitive
```

#### Compliant Solution

Use an allow-list to restrict the arguments to trusted values:

```
using System.Diagnostics;
Process p = new Process();
p.StartInfo.FileName = "/usr/bin/find";
if (allowed.Contains(input)) {
    p.StartInfo.ArgumentList.Add(input);
}
```

 Code Smell
<b>Fields should not have public accessibility</b>
 Code Smell
<b>URLs should not be hardcoded</b>
 Code Smell
<b>Types should be named in PascalCase</b>
 Code Smell
<b>Track uses of "TODO" tags</b>
 Code Smell

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-88](#) - Argument Injection or Modification
- [SANS Top 25](#) - Insecure Interaction Between Components
- [CVE-2021-29472](#) - PHP Supply Chain Attack on Composer

Available In:

**sonarcloud**  | **sonarqube**  Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)