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...

Empty nullable value should not be accessed
Bug

Nullable type comparison should not be redundant
Bug

Methods with "Pure" attribute should return a value
Bug

One-way "OperationContract" methods should have "void" return type
Bug

Optional parameters should be passed to "base" calls
Bug

Classes should not have only "private" constructors
Bug

Expressions used in "Debug.Assert" should not produce side effects
Bug

Caller information parameters should come at the end of the parameter list
Bug

Static fields should appear in the order they must be initialized
Bug

Classes directly extending "object" should not call "base" in "GetHashCode" or "Equals"
Bug

Anonymous delegates should not be used to unsubscribe from Events
Bug

Delegates should not be subtracted

Disabling CSRF protections is security-sensitive

Analyze your code

Security Hotspot | Critical | cwe sans-top25 owasp

A cross-site request forgery (CSRF) attack occurs when a trusted user of a web application can be forced, by an attacker, to perform sensitive actions that he didn't intend, such as updating his profile or sending a message, more generally anything that can change the state of the application.

The attacker can trick the user/victim to click on a link, corresponding to the privileged action, or to visit a malicious web site that embeds a hidden web request and as web browsers automatically include cookies, the actions can be authenticated and sensitive.

Ask Yourself Whether

- The web application uses cookies to authenticate users.
- There exist sensitive operations in the web application that can be performed when the user is authenticated.
- The state / resources of the web application can be modified by doing HTTP POST or HTTP DELETE requests for example.

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

Recommended Secure Coding Practices

- Protection against CSRF attacks is strongly recommended:
  - to be activated by default for all unsafe HTTP methods.
  - implemented, for example, with an unguessable CSRF token
- Of course all sensitive operations should not be performed with safe HTTP methods like `GET` which are designed to be used only for information retrieval.

Noncompliant Code Example

```
public void ConfigureServices(IServiceCollection services)
{
    // ...
    services.AddControllersWithViews(options => options.Filt
    // ...
}
```

```
[HttpPost, IgnoreAntiforgeryToken] // Sensitive
public IActionResult ChangeEmail(ChangeEmailModel model) =>
```

Compliant Solution

```
public void ConfigureServices(IServiceCollection services)
{
    // ...
    services.AddControllersWithViews(options => options.Filt
    // or
    services.AddControllersWithViews(options => options.Filt
```

Delegates should not be subtracted

Bug

"async" methods should not return "void"

Bug

"ThreadStatic" should not be used on non-static fields

Bug

"IDisposables" created in a "using" statement should not be returned

Bug

"ThreadStatic" fields should not be

```
    // ...
}
```

```
[HttpPost]
[AutoValidateAntiforgeryToken]
public IActionResult ChangeEmail(ChangeEmailModel model) =>
```

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- MITRE, CWE-352 - Cross-Site Request Forgery (CSRF)
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- OWASP: Cross-Site Request Forgery
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud | sonarqube