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...

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
Code Smell

Fields should not have public accessibility
Code Smell

URIs should not be hardcoded
Code Smell

Types should be named in PascalCase
Code Smell

Track uses of "TODO" tags
Code Smell

Disabling ASP.NET "Request Validation" feature is security-sensitive

Analyze your code

Security Hotspot | Major | cwe owasp

ASP.NET 1.1+ comes with a feature called *Request Validation*, preventing the server to accept content containing un-encoded HTML. This feature comes as a first protection layer against Cross-Site Scripting (XSS) attacks and act as a simple Web Application Firewall (WAF) rejecting requests potentially containing malicious content.

While this feature is not a silver bullet to prevent all XSS attacks, it helps to catch basic ones. It will for example prevent `<script type="text/javascript" src="https://malicious.domain/payload.js">` to reach your Controller.

Note: *Request Validation* feature being only available for ASP.NET, no Security Hotspot is raised on ASP.NET Core applications.

Ask Yourself Whether

- the developer doesn't know the impact to deactivate the Request Validation feature
- the web application accepts user-supplied data
- all user-supplied data are not validated

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

Recommended Secure Coding Practices

- Activate the Request Validation feature for all HTTP requests

Sensitive Code Example

At Controller level:

```
[ValidateInput(false)]
public ActionResult Welcome(string name)
{
  ...
}
```

At application level, configured in the Web.config file:

```
<configuration>
   <system.web>
      <pages validateRequest="false" />
      ...
      <httpRuntime requestValidationMode="0.0" />
   </system.web>
</configuration>
```

Compliant Solution

At Controller level:

Classes with "IDisposable" members should implement "IDisposable"

Bug

Calls to "async" methods should not be blocking

Code Smell

Child class fields should not shadow parent class fields

Code Smell

Track lack of copyright and license headers

Code Smell

```
[ValidateInput(true)]
public ActionResult Welcome(string name)
{
  ...
}
```

or

```
public ActionResult Welcome(string name)
{
  ...
}
```

At application level, configured in the Web.config file:

```
<configuration>
   <system.web>
      <pages validateRequest="true" />
      ...
      <httpRuntime requestValidationMode="4.5" />
   </system.web>
</configuration>
```

See

- OWASP Top 10 2021 Category A3 - Injection
- HttpRuntimeSection.RequestValidationMode Property
- OWASP ASP.NET Request Validation
- OWASP Cheat Sheet - XSS Prevention Cheat Sheet
- OWASP Top 10 2017 Category A7 - Cross-Site Scripting (XSS)
- MITRE, CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Available In:

sonarcloud | sonarqube