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...

"protected" members
Code Smell

Underscores should be used to make large numbers readable
Code Smell

"ToString()" calls should not be redundant
Code Smell

"==" should not be used when "Equals" is overridden
Code Smell

An abstract class should have both abstract and concrete methods
Code Smell

Multiple variables should not be declared on the same line
Code Smell

Culture should be specified for "string" operations
Code Smell

"switch" statements should have at least 3 "case" clauses
Code Smell

break statements should not be used except for switch cases
Code Smell

String literals should not be duplicated
Code Smell

Files should contain an empty newline at the end
Code Smell

Unused "using" should be removed
Code Smell

HTTP response headers should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Minor | injection

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Applications constructing HTTP response headers based on tainted data could allow attackers to change security sensitive headers like Cross-Origin Resource Sharing headers.

Web application frameworks and servers might also allow attackers to inject new line characters in headers to craft malformed HTTP response. In this case the application would be vulnerable to a larger range of attacks like HTTP Response Splitting/Smuggling. Most of the time this type of attack is mitigated by default modern web application frameworks but there might be rare cases where older versions are still vulnerable.

As a best practice, applications that use user-provided data to construct the response header should always validate the data first. Validation should be based on a whitelist.

Noncompliant Code Example

```
string value = Request.QueryString["value"];
Response.AddHeader("X-Header", value); // Noncompliant
```

Compliant Solution

```
string value = Request.QueryString["value"];
// Allow only alphanumeric characters
if (value == null || !Regex.IsMatch(value, "^[a-zA-Z0-9]+$")
{
  throw new Exception("Invalid value");
}
Response.AddHeader("X-Header", value);
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Attack Category - HTTP Response Splitting
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-113 - Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- SANS Top 25 - Insecure Interaction Between Components

Deprecated

This rule is deprecated; use {rule:csharpsquid:S5122}, {rule:csharpsquid:S5146}, {rule:csharpsquid:S6287} instead.

Available In:

sonarcloud | sonarqube Developer Edition

A close curly brace should be located at the beginning of a line

Code Smell

Tabulation characters should not be used

Code Smell

Methods and properties should be named in PascalCase

Code Smell

Track uses of in-source issue suppressions

Code Smell