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...

HTTP request redirections should not be open to forging attacks
Vulnerability

Deserialization should not be vulnerable to injection attacks
Vulnerability

Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks
Vulnerability

"CoSetProxyBlanket" and "CoInitializeSecurity" should not be used
Vulnerability

Database queries should not be vulnerable to injection attacks
Vulnerability

XML parsers should not be vulnerable to XXE attacks
Vulnerability

A secure password should be used when connecting to a database
Vulnerability

XPath expressions should not be vulnerable to injection attacks
Vulnerability

I/O function calls should not be vulnerable to path injection attacks
Vulnerability

LDAP queries should not be vulnerable to injection attacks
Vulnerability

OS commands should not be vulnerable to command injection attacks
Vulnerability

HTTP request redirections should not be open to forging attacks

Analyze your code

Vulnerability | Blocker | injection cwe sans-top25 owasp

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Applications performing HTTP redirects based on tainted data could enable an attacker to redirect users to a malicious site to, for example, steal login credentials.

This problem could be mitigated in any of the following ways:

- Validate the user-provided data based on an allowlist and reject input not matching.
- Redesign the application to not perform redirects based on user-provided data.

Noncompliant Code Example

```
using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{

    public IActionResult RedirectMe(string url)
    {
        return Redirect(url);
    }

    public IActionResult SetLocationHeader(string url)
    {
        Response.Headers["Location"] = url; // Noncompliant
        return StatusCode(302);
    }
}
```

Compliant Solution

```
using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{
    private readonly string[] whiteList = { "/", "/login", "

    public IActionResult RedirectMe(string url)
    {
        // Match the incoming URL against a whitelist
        if (!whiteList.Contains(url))
        {
            return BadRequest();
        }

        return Redirect(url);
    }
}
```

Classes should implement their "ExportAttribute" interfaces

Bug

Neither "Thread.Resume" nor "Thread.Suspend" should be used

Bug

"SafeHandle.DangerousGetHandle" should not be called

Bug

Type inheritance should not be recursive

Bug

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- Microsoft Documentation ASP.NET Core - Prevent Open Redirect Attacks in ASP.NET Core
- Microsoft Documentation ASP.NET MVC - Preventing Open Redirection Attacks
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-601 - URL Redirection to Untrusted Site ('Open Redirect')
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud | sonarqube Developer Edition