

- 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... 🔍

"Exception" should not be caught when not required by called methods

Code Smell

"sealed" classes should not have "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

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive

Analyze your code

Security Hotspot Minor ⓘ cwe owasp sans-top25

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2018-0269
- CVE-2017-14460

Same origin policy in browsers prevents, by default and for security-reasons, a javascript frontend to perform a cross-origin HTTP request to a resource that has a different origin (domain, protocol, or port) from its own. The requested target can append additional HTTP headers in response, called CORS, that act like directives for the browser and change the access control policy / relax the same origin policy.

Ask Yourself Whether

- You don't trust the origin specified, example: Access-Control-Allow-Origin: untrustedwebsite.com.
- Access control policy is entirely disabled: Access-Control-Allow-Origin: *
- Your access control policy is dynamically defined by a user-controlled input like origin header.

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

Recommended Secure Coding Practices






- The Access-Control-Allow-Origin header should be set only for a trusted origin and for specific resources.
- Allow only selected, trusted domains in the Access-Control-Allow-Origin header. Prefer whitelisting domains over blacklisting or allowing any domain (do not use * wildcard nor blindly return the Origin header content without any checks).

Sensitive Code Example

ASP.NET Core MVC:

```
[HttpGet]
public string Get()
{
    Response.Headers.Add("Access-Control-Allow-Origin", "*")
    Response.Headers.Add(HeaderNames.AccessControlAllowOrig
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddDefaultPolicy(builder =>
        {
            builder.WithOrigins("*"); // Sensitive
        }
    }
}
```


Unused "using" should be removed  Code Smell
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

```
});

options.AddPolicy(name: "EnableAllPolicy", builder =
{
    builder.WithOrigins("*"); // Sensitive
});

options.AddPolicy(name: "OtherPolicy", builder =>
{
    builder.AllowAnyOrigin(); // Sensitive
});

});

services.AddControllers();
}
```

ASP.NET MVC:

```
public class HomeController : ApiController
{
    public HttpResponseMessage Get()
    {
        var response = HttpContext.Current.Response;

        response.Headers.Add("Access-Control-Allow-Origin",
        response.Headers.Add(HeaderNames.AccessControlAllowO
        response.AppendHeader(HeaderNames.AccessControlAllow

    }
}
```

```
[EnableCors(origins: "*", headers: "*", methods: "GET")] //
public HttpResponseMessage Get() => new HttpResponseMessage(
{
    Content = new StringContent("content")
});
```

User-controlled origin:

```
String origin = Request.Headers["Origin"];
Response.Headers.Add("Access-Control-Allow-Origin", origin);
```

Compliant Solution

ASP.NET Core MVC:

```
[HttpGet]
public string Get()
{
    Response.Headers.Add("Access-Control-Allow-Origin", "htt
    Response.Headers.Add(HeaderNames.AccessControlAllowOrigi
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddDefaultPolicy(builder =>
        {
            builder.WithOrigins("https://trustedwebsite.com"
        });

        options.AddPolicy(name: "EnableAllPolicy", builder =
        {
            builder.WithOrigins("https://trustedwebsite.com"
        });
    });

    services.AddControllers();
}
```

ASP.Net MVC:

```
public class HomeController : ApiController
{
```

```
public HttpResponseMessage Get()
{
    var response = HttpContext.Current.Response;

    response.Headers.Add("Access-Control-Allow-Origin",
    response.Headers.Add(HeaderNames.AccessControlAllowO
    response.AppendHeader(HeaderNames.AccessControlAllow

    }
}
```

```
[EnableCors(origins: "https://trustedwebsite.com", headers:
public HttpResponseMessage Get() => new HttpResponseMessage(
{
    Content = new StringContent("content")
});
```

User-controlled origin validated with an allow-list:

```
String origin = Request.Headers["Origin"];

if (trustedOrigins.Contains(origin))
{
    Response.Headers.Add("Access-Control-Allow-Origin", orig
}
```

See

- [OWASP Top 10 2021 Category A5](#) - Security Misconfiguration
- [OWASP Top 10 2021 Category A7](#) - Identification and Authentication Failures
- [developer.mozilla.org](#) - CORS
- [developer.mozilla.org](#) - Same origin policy
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [OWASP HTML5 Security Cheat Sheet](#) - Cross Origin Resource Sharing
- [MITRE, CWE-346](#) - Origin Validation Error
- [MITRE, CWE-942](#) - Overly Permissive Cross-domain Whitelist
- [SANS Top 25](#) - Porous Defenses

Available In:

sonarcloud  | **sonarqube** 