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

## Controlling permissions is security-sensitive

Analyze your code

Security Hotspot | Minor

The access control of an application must be properly implemented in order to restrict access to resources to authorized entities otherwise this could lead to vulnerabilities:

- CVE-2018-12999
- CVE-2018-10285
- CVE-2017-7455

Granting correct permissions to users, applications, groups or roles and defining required permissions that allow access to a resource is sensitive, must therefore be done with care. For instance, it is obvious that only users with administrator privilege should be authorized to add/remove the administrator permission of another user.

### Ask Yourself Whether

- Granted permission to an entity (user, application) allow access to information or functionalities not needed by this entity.
- Privileges are easily acquired (eg: based on the location of the user, type of device used, defined by third parties, does not require approval ...).
- Inherited permission, default permission, no privileges (eg: anonymous user) is authorized to access to a protected resource.

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

### Recommended Secure Coding Practices

At minimum, an access control system should:

- Use a well-defined access control model like RBAC or ACL.
- Entities' permissions should be reviewed regularly to remove permissions that are no longer needed.
- Respect the principle of least privilege ("*an entity has access only the information and resources that are necessary for its legitimate purpose*").

### Sensitive Code Example

```
using System.Threading;
using System.Security.Permissions;
using System.Security.Principal;
using System.IdentityModel.Tokens;

class SecurityPrincipalDemo
{
    class MyIdentity : IIdentity // Sensitive, custom IIdent
    {
        // ...
    }

    class MyPrincipal : IPrincipal // Sensitive, custom IPri
    {
        // ...
```

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

```
    }
    [System.Security.Permissions.PrincipalPermission(Securit
    static void CheckAdministrator()
    {
        WindowsIdentity MyIdentity = WindowsIdentity.GetCurr
        HttpContext.User = ...; // Sensitive: review all ref
        AppDomain domain = AppDomain.CurrentDomain;
        domain.SetPrincipalPolicy(PrincipalPolicy.WindowsPri
        MyIdentity identity = new MyIdentity(); // Sensitive
        MyPrincipal MyPrincipal = new MyPrincipal(MyIdentity
        Thread.CurrentPrincipal = MyPrincipal; // Sensitive
        domain.SetThreadPrincipal(MyPrincipal); // Sensitive

        // All instantiation of PrincipalPermission should b
        PrincipalPermission principalPerm = new PrincipalPer
        principalPerm.Demand();

        SecurityTokenHandler handler = ...;
        // Sensitive: this creates an identity.
        ReadOnlyCollection<ClaimsIdentity> identities = hand
    }

     // Sensitive: review how this function uses the identit
    void modifyPrincipal(MyIdentity identity, MyPrincipal pr
    {
        // ...
    }
}
```

**See**

- OWASP Top 10 2017 Category A5 - Broken Access Control
- SANS Top 25 - Porous Defenses
- MITRE, CWE-276 - Incorrect Default Permissions
- MITRE, CWE-732 - Incorrect Permission Assignment for Critical Resource
- MITRE, CWE-668 - Exposure of Resource to Wrong Sphere
- MITRE, CWE-277 - Insecure Inherited Permissions

**Deprecated**

This rule is deprecated, and will eventually be removed.

Available In:

sonarcloud | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
Privacy Policy