

- 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...

vulnerability

LDAP queries should not be vulnerable to injection attacks

Vulnerability

OS commands should not be vulnerable to command injection attacks

Vulnerability

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

"IDisposable" should be disposed

Bug

SQL keywords should be delimited by whitespace

Bug

Composite format strings should not lead to unexpected behavior at runtime

Bug

Recursion should not be infinite

Bug

Destructors should not throw exceptions

Bug

"CoSetProxyBlanket" and "CoInitializeSecurity" should not be used

Analyze your code

Vulnerability Blocker owasp

CoSetProxyBlanket and CoInitializeSecurity both work to set the permissions context in which the process invoked immediately after is executed. Calling them from within that process is useless because it's too late at that point; the permissions context has already been set.

Specifically, these methods are meant to be called from non-managed code such as a C++ wrapper that then invokes the managed, i.e. C# or VB.NET, code.

Noncompliant Code Example


```
[DllImport("ole32.dll")]
static extern int CoSetProxyBlanket([MarshalAs(UnmanagedType
[MarshalAs(UnmanagedType.LPWStr)] string pServerPrin
uint dwCapabilities);

public enum RpcAuthnLevel
{
    Default = 0,
    None = 1,
    Connect = 2,
    Call = 3,
    Pkt = 4,
    PktIntegrity = 5,
    PktPrivacy = 6
}

public enum RpcImpLevel
{
    Default = 0,
    Anonymous = 1,
    Identify = 2,
    Impersonate = 3,
    Delegate = 4
}

public enum EoAuthnCap
{
    None = 0x00,
    MutualAuth = 0x01,
    StaticCloaking = 0x20,
    DynamicCloaking = 0x40,
    AnyAuthority = 0x80,
    MakeFullSIC = 0x100,
    Default = 0x800,
    SecureRefs = 0x02,
    AccessControl = 0x04,
    AppID = 0x08,
    Dynamic = 0x10,
    RequireFullSIC = 0x200,
    AutoImpersonate = 0x400,
```


Hard-coded credentials are security-sensitive

 Security Hotspot

Exceptions should not be thrown from unexpected methods

 Code Smell

"operator==" should not be overloaded on reference types

 Code Smell

Type should not be examined on

"System.Type" instances

 Code Smell

```
NoCustomMarshal = 0x2000,
DisableAAA = 0x1000
}

[DllImport("ole32.dll")]
public static extern int CoInitializeSecurity(IntPtr pVoid,
    RpcAuthnLevel level, RpcImpLevel impers, IntPtr pAut

static void Main(string[] args)
{
    var hres1 = CoSetProxyBlanket(null, 0, 0, null, 0, 0

    var hres2 = CoInitializeSecurity(IntPtr.Zero, -1, In
        RpcImpLevel.Impersonate, IntPtr.Zero, EoAuth

}
```

See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-648](#) - Incorrect Use of Privileged APIs

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 