sonar RULES

Products

- 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 responses should not be vulnerable to session fixation
Vulnerability

Extracting archives should not lead to zip slip vulnerabilities
Vulnerability

Dynamic code execution should not be vulnerable to injection attacks
Vulnerability

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

Dynamic code execution should not be vulnerable to injection attacks

Analyze your code

Vulnerability   Blocker   injection cwe owasp sans-top25

Applications that execute code dynamically should neutralize any externally-provided values used to construct the code. Failure to do so could allow an attacker to execute arbitrary code. This could enable a wide range of serious attacks like accessing/modifying sensitive information or gain full system access.

The mitigation strategy should be based on whitelisting of allowed values or casting to safe types.

Noncompliant Code Example

```
using Microsoft.AspNetCore.Mvc;
using System.CodeDom.Compiler;

namespace WebApplicationDotNetCore.Controllers
{
    public class DynamicCodeExecutionNoncompliantController
    {
        public ActionResult UnsafeCodeExecution(string code)
        {
            var provider = CodeDomProvider.CreateProvider("C
            var compilerParameters = new CompilerParameters
            var compilerResults = provider.CompileAssemblyFr
            object myInstance = compilerResults.CompiledAsse
            var result = (string)myInstance.GetType().GetMet
            return Content(result);
        }
    }
}
```

Compliant Solution

```
using Microsoft.AspNetCore.Mvc;
using System.CodeDom.Compiler;
using System.Linq;

namespace WebApplicationDotNetCore.Controllers
{
    public class DynamicCodeExecutionCompliantController : C
    {
        private readonly string[] allowedInnerInvocations =

        public ActionResult SafeCodeExecution(string innerIn
        {
            // Match the input against a whitelist
            if (!allowedInnerInvocations.Contains(innerInvoc
            {
                return BadRequest();
            }
            // Code created is based on controlled template
```

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

Classes should implement their "ExportAttribute" interfaces

Bug

```
            var code = CreateFromTemplate(innerInvocationCod

            var provider = CodeDomProvider.CreateProvider("C
            var compilerParameters = new CompilerParameters
            var compilerResults = provider.CompileAssemblyFr
            object myInstance = compilerResults.CompiledAsse
            var result = (string)myInstance.GetType().GetMet
            return Content(result);
        }

        private string CreateFromTemplate(string innerInvoca
        {
            // Create code to be compiled from known templat
            // ...
        }
    }
}
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-95 - Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud sonarqube Developer Edition