

- 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

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

Hard-coded credentials are security-sensitive

Security Hotspot

Exceptions should not be thrown from unexpected methods

Code Smell

XML parsers should not be vulnerable to XXE attacks

Analyze your code

Vulnerability Blocker cwe owasp

XML standard allows the use of entities, declared in the DOCTYPE of the document, which can be [internal](#) or [external](#).

When parsing the XML file, the content of the external entities is retrieved from an external storage such as the file system or network, which may lead, if no restrictions are put in place, to arbitrary file disclosures or [server-side request forgery \(SSRF\)](#) vulnerabilities.

It's recommended to limit resolution of external entities by using one of these solutions:

- If DOCTYPE is not necessary, completely disable all DOCTYPE declarations.
- If external entities are not necessary, completely disable their declarations.
- If external entities are necessary then:
 - Use XML processor features, if available, to authorize only required protocols (eg: https).
 - And use an entity resolver (and optionally an XML Catalog) to resolve only trusted entities.

Noncompliant Code Example

System.Xml.XmlDocument

```
// .NET Framework < 4.5.2
XmlDocument parser = new XmlDocument(); // Noncompliant: Xml
parser.LoadXml("xxe.xml");
```

or

```
// .NET Framework 4.5.2+
XmlDocument parser = new XmlDocument();
parser.XmlResolver = new XmlUrlResolver(); // Noncompliant:
parser.LoadXml("xxe.xml");
```

System.Xml.XmlTextReader


```
// .NET Framework < 4.5.2
XmlTextReader reader = new XmlTextReader("xxe.xml"); // Nonc
while (reader.Read())
{ ... }
```

or

```
// .NET Framework 4.5.2+
XmlTextReader reader = new XmlTextReader("xxe.xml");
reader.XmlResolver = new XmlUrlResolver(); // Noncompliant:
while (reader.Read())
{ ... }
```

System.Xml.XmlReader

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

 Code Smell

Type should not be examined on

"System.Type" instances

 Code Smell

Test method signatures should be correct

 Code Smell

Method overloads with default parameter values should not overlap

 Code Smell

```
// .NET Framework 4.5.2+
XmlReaderSettings settings = new XmlReaderSettings();
settings.DtdProcessing = DtdProcessing.Parse;
settings.XmlResolver = new XmlUrlResolver();
XmlReader reader = XmlReader.Create("xe.xml", settings); //
while (reader.Read())
{ ... }
```

System.Xml.XPath.XPathDocument

```
// prior to .NET 4.5.2
XPathDocument doc = new XPathDocument("example.xml"); // Non
XPathNavigator nav = doc.CreateNavigator();
string xml = nav.InnerXml.ToString();
```

Compliant Solution

System.Xml.XmlDocument

```
XmlDocument parser = new XmlDocument();
parser.XmlResolver = null; // Compliant: XmlResolver has been
parser.LoadXml("xe.xml");
```

or

```
XmlDocument parser = new XmlDocument(); // Compliant: XmlDocument
parser.LoadXml("xe.xml");
```

System.Xml.XmlTextReader

```
// .NET 4.5.2+
XmlTextReader reader = new XmlTextReader("xe.xml"); // Compliant
while (reader.Read())
{ ... }

// .NET 4.0 to .NET 4.5.1
XmlTextReader reader = new XmlTextReader("xe.xml");
reader.DtdProcessing = DtdProcessing.Prohibit; // Compliant:

// < .NET 4.0
XmlTextReader reader = new XmlTextReader(stream);
reader.ProhibitDtd = true; // Compliant: default is false
```

System.Xml.XmlReader

```
XmlReader reader = XmlReader.Create("xe.xml"); // Compliant
while (reader.Read())
{ ... }
```

System.Xml.XPath.XPathDocument

```
// prior to .NET 4.5.2
XmlReader reader = XmlReader.Create("example.xml");
XPathDocument doc = new XPathDocument(reader); // Compliant:
XPathNavigator nav = doc.CreateNavigator();
string xml = nav.InnerXml.ToString();
```

See

- [OWASP Top 10 2021 Category A5](#) - Security Misconfiguration
- [OWASP Top 10 2017 Category A4](#) - XML External Entities (XXE)
- [OWASP XXE Prevention Cheat Sheet](#)
- [MITRE, CWE-611](#) - Information Exposure Through XML External Entity Reference
- [MITRE, CWE-827](#) - Improper Control of Document Type Definition

Available In:

 |  | 

