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...

"params" should be used on overrides
Code Smell

Generic type parameters should be co/contravariant when possible
Code Smell

Multiple "OrderBy" calls should not be used
Code Smell

Reflection should not be used to increase accessibility of classes, methods, or fields
Code Smell

Static fields should not be updated in constructors
Code Smell

"IEnumerable" LINQs should be simplified
Code Smell

Fields that are only assigned in the constructor should be "readonly"
Code Smell

Static fields should not be used in generic types
Code Smell

Multiline blocks should be enclosed in curly braces
Code Smell

Boolean expressions should not be gratuitous
Code Smell

Types and methods should not have too many generic parameters
Code Smell

Write-only properties should not be used

Types allowed to be deserialized should be restricted

Analyze your code

Vulnerability | Major | cwe owasp

During the deserialization process, the state of an object will be reconstructed from the serialized data stream which can contain dangerous operations.

For example, a well-known attack vector consists in serializing an object of type TempFileCollection with arbitrary files (defined by an attacker) which will be deleted on the application deserializing this object (when the finalize() method of the TempFileCollection object is called). This kind of types are called "gadgets".

Instead of using `BinaryFormatter` and similar serializers, it is recommended to use safer alternatives in most of the cases, such as XmlSerializer or DataContractSerializer. If it's not possible then try to mitigate the risk by restricting the types allowed to be deserialized:

- by implementing an "allow-list" of types, but keep in mind that novel dangerous types are regularly discovered and this protection could be insufficient over time.
- or/and implementing a tamper protection, such as message authentication codes (MAC). This way only objects serialized with the correct MAC hash will be deserialized.

Noncompliant Code Example

For BinaryFormatter, NetDataContractSerializer, SoapFormatter serializers:

```
var myBinaryFormatter = new BinaryFormatter();
myBinaryFormatter.Deserialize(stream); // Noncompliant: a bi
```

JavaScriptSerializer should not use SimpleTypeResolver or other weak resolvers:

```
JavaScriptSerializer serializer1 = new JavaScriptSerializer(
serializer1.Deserialize<ExpectedType>(json);
```

LosFormatter should not be used without MAC verification:

```
LosFormatter formatter = new LosFormatter(); // Noncompliant
formatter.Deserialize(fs);
```

Compliant Solution

BinaryFormatter, NetDataContractSerializer , SoapFormatter serializers should use a binder implementing a whitelist approach to limit types during deserialization (at least one exception should be thrown or a null value returned):

```
sealed class CustomBinder : SerializationBinder
{
   public override Type BindToType(string assemblyName, stri
   {
       if (!(typeName == "type1" || typeName == "type2" || t
       {
```

Code Smell

Exceptions should not be thrown from property getters

Code Smell

Unused type parameters should be removed

Code Smell

Parameters should be passed in the correct order

Code Smell

Two branches in a conditional structure should not have exactly the

```
            throw new SerializationException("Only type1, type
        }
        return Assembly.Load(assemblyName).GetType(typeName);
    }
}

var myBinaryFormatter = new BinaryFormatter();
myBinaryFormatter.Binder = new CustomBinder();
myBinaryFormatter.Deserialize(stream);
```

JavaScriptSerializer should use a resolver implementing a whitelist to limit types during deserialization (at least one exception should be thrown or a null value returned):

```
public class CustomSafeTypeResolver : JavaScriptTypeResolver
{
   public override Type ResolveType(string id)
   {
      if(id != "ExpectedType") {
         throw new ArgumentNullException("Only ExpectedType
      }
      return Type.GetType(id);
   }
}

JavaScriptSerializer serializer = new JavaScriptSerializer(n
serializer.Deserialize<ExpectedType>(json);
```

LosFormatter serializer with MAC verification:

```
LosFormatter formatter = new LosFormatter(true, secret); //
formatter.Deserialize(fs);
```

See

- OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures
- docs.microsoft.com - BinaryFormatter security guide
- OWASP Top 10 2017 Category A8 - Insecure Deserialization
- MITRE, CWE-134 - Use of Externally-Controlled Format String
- MITRE, CWE-502 - Deserialization of Untrusted Data
- SANS Top 25 - Risky Resource Management
- OWASP Deserialization Cheat Sheet

Available In:

sonarlint | sonarcloud | sonarqube