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...

Strings should not be concatenated using '+' in a loop
Code Smell

Unused local variables should be removed
Code Smell

Private fields only used as local variables in methods should become local variables
Code Smell

A "while" loop should be used instead of a "for" loop
Code Smell

"Equals" and the comparison operators should be overridden when implementing "IComparable"
Code Smell

Nested code blocks should not be used
Code Smell

Overriding members should do more than simply call the same member in the base class
Code Smell

"Any()" should be used to test for emptiness
Code Smell

Boolean literals should not be redundant
Code Smell

Empty statements should be removed
Code Smell

Fields should not have public accessibility
Code Smell

URIs should not be hardcoded

Deserializing objects without performing data validation is security-sensitive

Analyze your code

Security Hotspot | Major | cwe owasp

Deserialization process extracts data from the serialized representation of an object and reconstruct it directly, without calling constructors. Thus, data validation implemented in constructors can be bypassed if serialized objects are controlled by an attacker.

Ask Yourself Whether

- The data validation implemented in constructors enforces a relevant security check.
- Objects instantiated via deserialization don't run the same security checks as the ones executed when objects are created through constructors.

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

Recommended Secure Coding Practices

- At the end of the deserialization process it is recommended to perform the same validation checks as the ones performed in constructors, especially when the serialized object can be controlled by an attacker.

Sensitive Code Example

When a `serializable` class doesn't inherit from ISerializable or IDeserializationCallback types and has a constructor using its parameters in conditions:

```
[Serializable]
public class InternalUrl
{
    private string url;

    public InternalUrl(string tmpUrl) // Sensitive
    {
       if(!tmpUrl.StartsWith("http://localhost/")) // there
       {
          url= "http://localhost/default";
       }
       else
       {
          url= tmpUrl;
       }
    }
}
```

When a class inherit from ISerializable type, has a regular constructor using its parameters in conditions, but doesn't perform the same validation after deserialization:

```
[Serializable]
public class InternalUrl : ISerializable
```

Code Should not be hardcoded

Code Smell

Types should be named in PascalCase

Code Smell

Track uses of "TODO" tags

Code Smell

Classes with "IDisposable" members should implement "IDisposable"

Bug

Calls to "async" methods should not be blocking

Code Smell

```
{
    private string url;

    public InternalUrl(string tmpUrl) // Sensitive
    {
        if(!tmpUrl.StartsWith("http://localhost/")) // there
        {
            url= "http://localhost/default";
        }
        else
        {
            url= tmpUrl;
        }
    }

    // special constructor used during deserialization
    protected InternalUrl(SerializationInfo info, StreamingC
    {
       url= (string) info.GetValue("url", typeof(string));
       // the same validation as seen in the regular constru
     }

    void ISerializable.GetObjectData(SerializationInfo info,
    {
        info.AddValue("url", url);
    }
}
```

When a class inherit from IDeserializationCallback type, has a constructor using its parameters in conditions but the `IDeserializationCallback.OnDeserialization` method doesn't perform any conditional checks:

```
[Serializable]
public class InternalUrl : IDeserializationCallback
{
    private string url;

    public InternalUrl(string tmpUrl) // Sensitive
    {
        if(!tmpUrl.StartsWith("http://localhost/")) // there
        {
            url= "http://localhost/default";
        }
        else
        {
            url= tmpUrl;
        }
    }

    void IDeserializationCallback.OnDeserialization(object s
    {
       // the same validation as seen in the constructor is
    }
}
```

Compliant Solution

When using ISerializable type to control deserialization, perform the same checks inside regular constructors than in the special constructor `SerializationInfo info, StreamingContext context` used during deserialization:

```
[Serializable]
public class InternalUrl : ISerializable
{
    private string url;

    public InternalUrl(string tmpUrl)
    {
        if(!tmpUrl.StartsWith("http://localhost/")) // there
        {
            url= "http://localhost/default";
        }
        else
        {
            url= tmpUrl;
        }
    }
```

```
    // special constructor used during deserialization
    protected InternalUrl(SerializationInfo info, StreamingC
    {
       string tmpUrl= (string) info.GetValue("url", typeof(s

       if(!tmpUrl.StartsWith("http://localhost/") { // Compl
          url= "http://localhost/default";
       }
       else {
          url= tmpUrl;
       }
     }

    void ISerializable.GetObjectData(SerializationInfo info,
    {
        info.AddValue("url", url);
    }
}
```

When using IDeserializationCallback type to control deserialization, perform the same checks inside regular constructors than after deserialization with `IDeserializationCallback.OnDeserialization` method:

```
[Serializable]
public class InternalUrl : IDeserializationCallback
{
    private string url;

    public InternalUrl(string tmpUrl)
    {
       if(!tmpUrl.StartsWith("http://localhost/")) // there
       {
          url= "http://localhost/default";
       }
       else
       {
          url= tmpUrl;
       }
    }

    void IDeserializationCallback.OnDeserialization(object s
    {
        if(!url.StartsWith("http://localhost/"))
        {
            url= "http://localhost/default";
        }
        else
        {
        }
    }
}
```

See

- OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures
- OWASP Top 10 2017 Category A8 - Insecure Deserialization
- docs.microsoft.com - security-and-serialization
- MITRE, CWE-502 - Deserialization of Untrusted Data

Available In:

sonarcloud | sonarqube