Q





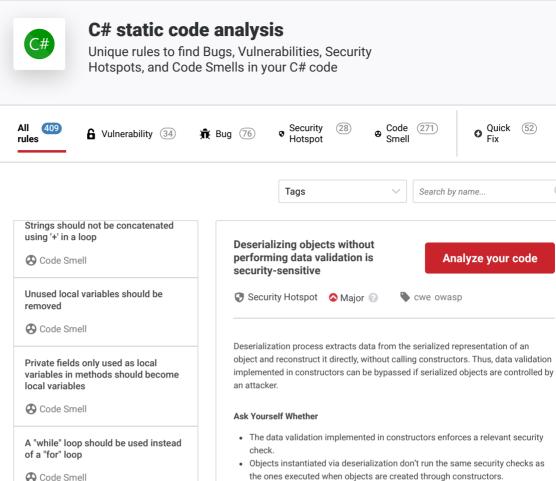
Code Smell

accessibility

Code Smell

Fields should not have public

LIRIs should not be hardcoded



Recommended Secure Coding Practices

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

 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 <u>ISerializable</u> or <u>IDeserializationCallback</u> 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 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

 ${\tt IDeserialization Callback. On Deserialization \ method\ doesn't\ perform\ any\ conditional\ checks:}$

```
[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. ODDESERIALIZATION 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