DAS ENDE EINER TECHNOLOGIE

[Serializable] ist tot

Wer noch den BinaryFormatter in Anwendungen nutzt, sollte jetzt aktiv werden.

Die Überschrift klingt, als würden sehr dunkle Wolken aufziehen – vor allem für bestehende Projekte und für .NET Entwicklerteams, die .NET schon lange treu sind. Serialisierung hat im Hause Microsoft und im Speziellen des .NET-Teams eine lange Geschichte. Doch Microsoft schraubt und verändert unter der Haube und entfernt alte Dinge aus dem Motor. Bald trifft es unsere klassische Serialisierung.

Die alte Serialisierung

Arbeitet man mit Newtonsoft.Json oder dem neuen JSON-Serialisierer, der seit .NET Core 3 der De-facto-Standard ist, so stolpert man in der Regel nicht über das Attribut *[Serializable]*. Das liegt an der Historie. In den ersten Versionen des .NET Frameworks war für JSON noch so gut wie nichts integriert beziehungsweise hat auch die Industrie nicht nach JSON gerufen – vielmehr waren XML und SOAP das Maß der Dinge.

Damals um die 2000er-Jahre war XML das Datenaustauschformat der Wahl. Microsoft hat zu der Zeit sogar eine eigene Konferenz „XML in Action" veranstaltet, um den Entwicklern das Format nahezubringen. In der Folge hat Microsoft unterschiedliche Wege im .NET-Umfeld implementiert, um mit XML-Daten zu arbeiten: *XmlSerializer*, *XmlDocument*, *XmlWriter* oder *XDocument* – um nur die gängigsten Wege zu erwähnen (Bild 1).

Das funktioniert auch immer noch perfekt und Microsoft tastet diese Funktionalität nicht an. XML ist agnostisch vom Framework und der Sprache und universell. Das generierte XML kann mit jeder anderen Entwicklungstechnik verwendet werden – egal ob .NET, Java oder andere (Bild 2).

```
using System.Xml.Serialization;

var p = new Person
{
    FirstName = "Christian",
    LastName = "Giesswein"
};

var serializer = new XmlSerializer(typeof(Person))
var sw = new StringWriter();
serializer.Serialize(sw, p);

2 usages
public class Person
{
    1 usage
    public string FirstName { get; set; }
    1 usage
    public string LastName { get; set; }
}
```

Da kommen fast schon nostalgische Gefühle auf: Serialisierung nach XML (Bild 1)

Die binäre Serialisierung

Ganz anders sieht es beim Thema der binären Serialisierung aus. Wollte man Daten binär versenden oder speichern, so ging man häufig den Weg über den BinaryFormatter, der Daten aus der .NET-Welt in ein binäres Format brachte, um diese beim Empfänger oder einfach nur später wieder zu laden.

Will man das kleine Beispiel anpassen, damit dieses mit dem BinaryFormatter arbeitet, verweigert unter .NET 7 bereits der Compiler seine Mitwirkung (Bild 3).

Microsoft hat mit .NET 5 begonnen, den BinaryFormatter Schritt für Schritt als obsolet zu kennzeichnen.

Mit .NET 7 erhält man statt einer Warnung bereits eine Fehlermeldung. Diese kann man zwar mit einem *#PRAGMA* deaktivieren, doch viel besser wird es damit in Zukunft nicht wirklich.

Zur Laufzeit muss man nun gerade beim BinaryFormatter dafür Sorge tragen, dass der Typ, den man serialisieren möchte als serialisierbar gekennzeichnet ist (Bild 4).

Mit einer kleinen Anpassung läuft unser Code mit .NET 7 wieder (Bild 5). Unter ASP.NET Core muss man sogar noch eine Projekteinstellung tätigen, damit er arbeitet. In einer Konsolenanwendung ist dies vorerst nicht notwendig.

Das Problem

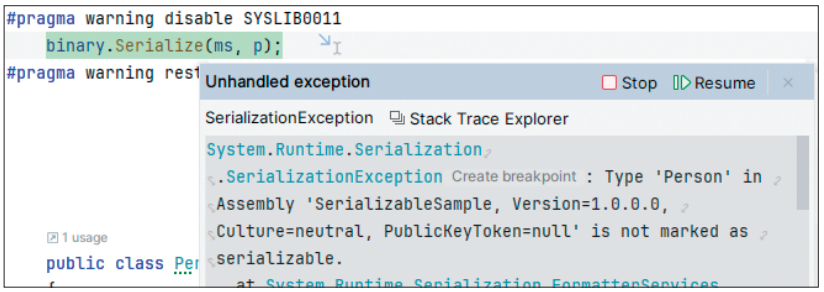
Warum hat denn nun Microsoft den BinaryFormatter nicht mehr lieb? Dazu hat Microsoft unter [1] die Beweggründe genannt. Grundsätzlich

```
<?xml version="1.0" encoding="utf-16"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <FirstName>Christian</FirstName>
  <LastName>Giesswein</LastName>
</Person>
```

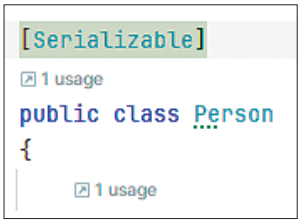
Weltformat XML: Egal ob .NET, Java oder eine andere Technologie – auf XML verstehen sich alle (Bild 2)

```
binary.Serialize(ms, p);
SYSLIB0011: BinaryFormatter serialization is obsolete and should not be used. See https://aka.ms/binaryformatter for more information.
```

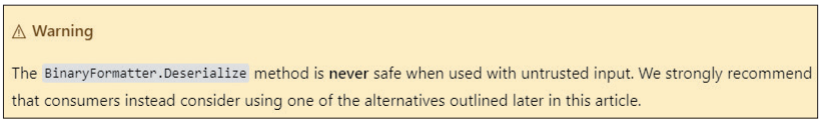
Der Compiler will nicht: Binäre Serialisierung funktioniert so nicht (Bild 3)



Der Typ muss als serialisierbar gekennzeichnet sein (Bild 4)



Mit dem Attribut [Serializable] funktioniert es (Bild 5)



Warnung, dass der BinaryFormatter nicht sicher ist (Bild 6)

sind es Sicherheitsbedenken, die in den letzten Jahren aufgekommen sind, und auch Sicherheitslücken wie die von log4j haben ein Bewusstsein dafür geschaffen [2].

Da der BinaryFormatter unter anderem die Typinformation persistiert, kann ein Angreifen potenziell Daten an unsere Anwendungen senden, durch die andere Typen geladen werden, die dann einen potenziellen Angriffsvektor zulassen. Dementsprechend hat der BinaryFormatter eine potenzielle Sicherheitslücke, wenn Daten von extern kommen. Dazu warnt Microsoft auch recht markant (Bild 6).

Das Ganze geht so weit, dass es sogar Projekte gibt, die zeigen, wie man den BinaryFormatter in die Irre führen kann – wie zum Beispiel das Projekt unter [3].

Das Problem sind vor allem die alten Datentypen wie *DataSet* oder dergleichen. Die sind nämlich ebenfalls alle im Standard als serialisierbar gekennzeichnet. Deswegen hilft das Attribut nur bedingt. Ein Angreifer kann unter Umständen recht witzige Objekte in unserer Anwendung instanzieren lassen.

Der Microsoft-Weg

Wie will Microsoft das Problem nun lösen? Ein Blick in die Dokumentation oder auf GitHub liefert die Antwort: Das Issue unter [4] beschreibt, was Microsoft mit .NET 8 bereits vorhat.

Mit .NET 8 sind also die unterliegenden Typen bereits alle obsolet. Gräbt man im Internet weiter nach der Strategie, die Microsoft hier verfolgt, so stößt man auf die BinaryFormatter Obsoletion Strategy [5].

Hier gibt es einen kompletten Zeitplan bis hin zu .NET 9 (Release 2024):

.NET 8 (Nov. 2023)

- All first-party dotnet org code bases complete migration away from BinaryFormatter
- BinaryFormatter disabled by default across all project types
- All not-yet-obsolete BinaryFormatter APIs marked obsolete as warning
- Additional legacy serialization infrastructure marked obsolete as warning
- No new [Serializable] types introduced (all target frameworks)

.NET 9 (Nov. 2024)

- Remainder of legacy serialization infrastructure marked obsolete as warning
- BinaryFormatter infrastructure removed from .NET
- Back-compat switches also removed

Mit .NET 9 will Microsoft also die gesamte Infrastruktur entfernen – hört, hört! Wer also bis heute immer alle Warnungen unterdrückt hat und das Thema von seinem Schreibtisch geschoben hat, sollte jetzt aufwachen: Es kommt eventuell bei Altprojekten einiges an Arbeit auf Sie zu.

Fazit

Sie verwenden keine klassischen Serialiserer mehr? Dann können Sie nun aufatmen und zufrieden sagen: Job done.

Sie arbeiten aktiv an einer Altanwendung, die vielleicht gerade auf .NET migriert wurde? Achtung: Es kommt die nächste Baustelle auf Sie zu:

„In .NET 9.0, the entirety of the BinaryFormatter infrastructure will be removed from the product."

Microsoft macht recht klar, dass mit .NET 9 tatsächlich Schluss ist mit der alten Serialisierungsinfrastruktur. Auch alle Zulieferer und Dritthersteller sind in den letzten Monaten und Jahren bereits aktiv geworden und verweigern auch immer mehr die Zusammenarbeit mit dem BinaryFormatter. Unterm Strich: Weg mit dem alten Zeug. ■

[1] BinaryFormatter security vulnerabilities, www.dotnetpro.de/SL2312NETirol1
[2] Kritische Schwachstelle in Java-Bibliothek Log4j, www.dotnetpro.de/SL2312NETirol2
[3] Generating payloads that exploit unsafe .NET object deserialization, www.dotnetpro.de/SL2312NETirol3
[4] Legacy serialization infrastructure APIs marked obsolete, www.dotnetpro.de/SL2312NETirol4
[5] BinaryFormatter Obsoletion Strategy, www.dotnetpro.de/SL2312NETirol5

Christian Giesswein
studierte Wirtschaftsinformatik in Wien und entwickelt von klein auf Software mit .NET und C#. In Tirol hat er das Unternehmen Giesswein Software-Solutions gegründet, das sich auf Individualsoftware und Consulting spezialisiert.
christian@software.tirol

dnpCode A2312NETirol