**The ObjectDataProvider Gadget**

**What is a Gadget?**

During engagements, to achieve objectives such as arbitrary file writes or remote code execution through a deserialization attack, it is necessary to use a so-called gadget, or in some cases, a combination of gadgets called a gadget chain. A gadget is an object set up in a specific way so that it executes a desired set of actions upon deserialization, most importantly, in the context of attacks we want to carry out.

Note: Identifying gadgets (and vulnerable deserialization libraries) ourselves is outside of the scope of this module because it requires a lot of research, and so we will be relying on public findings and papers.

**ObjectDataProvider**

**What is ObjectDataProvider?**

Let's look at a well-known gadget for .NET, which can be used to execute arbitrary commands using the ObjectDataProvider class.

According to [Microsoft's documentation](https://learn.microsoft.com/en-us/dotnet/api/system.windows.data.objectdataprovider?view=windowsdesktop-7.0), the ObjectDataProvider class can be used to "wrap and create an object that can be used as a binding source". This description probably doesn't make a lot of sense, so let's elaborate a little bit. The ObjectDataProvider class is part of the [Windows Presentation Foundation](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0), which is a .NET framework for developing graphic user interfaces (GUIs) with XAML (Microsoft's variant of XML). Taking a look at an [example](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/how-to-make-data-available-for-binding-in-xaml?view=netframeworkdesktop-4.8) that Mircosoft provides (listed below), the description starts to make a bit more sense. We can see that in this case, a new Person object is created with the constructor parameter "Joe", and that the Name property of the resulting object is accessed near the bottom of the document.

Code: xml

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:src="clr-namespace:SDKSample"

xmlns:system="clr-namespace:System;assembly=mscorlib"

SizeToContent="WidthAndHeight"

Title="Simple Data Binding Sample">

<Window.Resources>

<ObjectDataProvider x:Key="myDataSource" ObjectType="{x:Type src:Person}">

<ObjectDataProvider.ConstructorParameters>

<system:String>Joe</system:String>

</ObjectDataProvider.ConstructorParameters>

</ObjectDataProvider>

<SNIP>

</Window.Resources>

<Border Margin="25" BorderBrush="Aqua" BorderThickness="3" Padding="8">

<DockPanel Width="200" Height="100">

<SNIP>

<TextBlock Text="{Binding Source={StaticResource myDataSource}, Path=Name}"/>

</DockPanel>

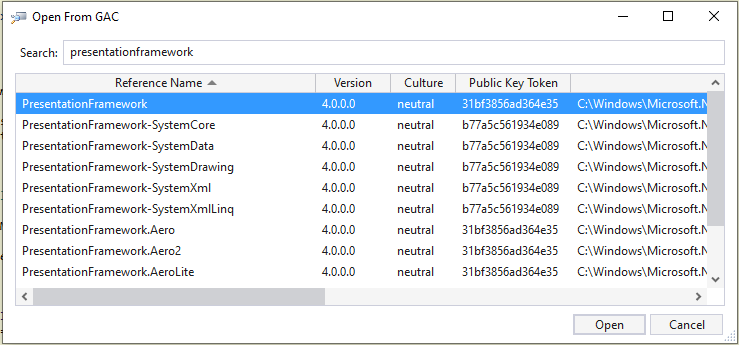
</Border>

</Window>

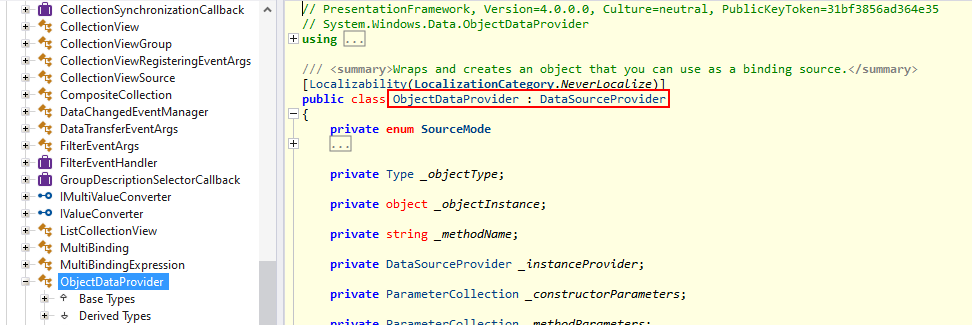
Most importantly, we notice that the object was created without any function calls! When we deserialize an object in .NET we can't execute any functions, so the fact that ObjectDataProvider does so automatically is very interesting for us as attackers.

**How does it work?**

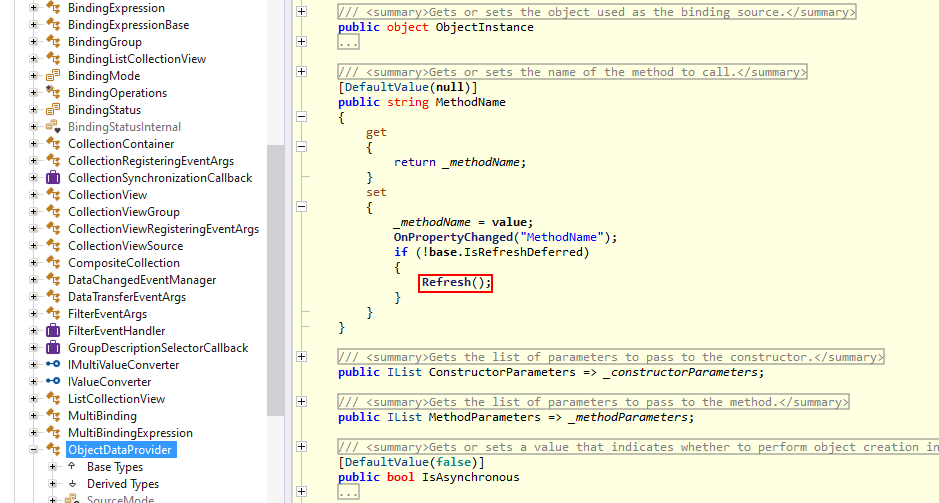
Let's take a look at why this is possible. We can open PresentationFramework.dll in ILSpy to look at what goes on behind the scenes. Select File > Open from GAC to open a library from the Global Assembly Cache, in this case, PresentationFramework.



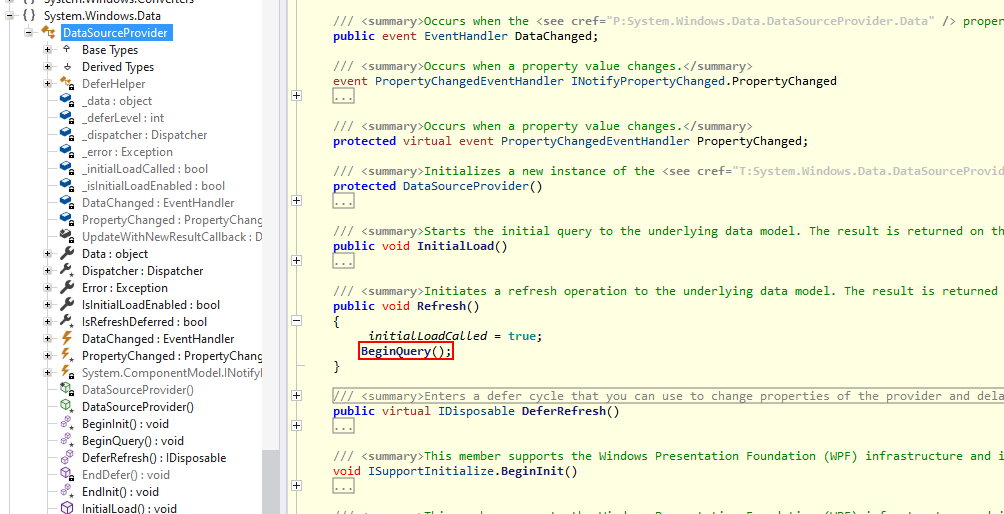
Navigating to System.Windows.Data and then ObjectProvider, we can open the decompiled source code, and the first thing we notice is that ObjectDataProvider inherits DataSourceProvider.



Scrolling down a bit to look at the MethodName field, we notice that the Refresh method is called when the value is set. This is important, because when an object is deserialized in C#, an empty instance is created and the properties are then set one by one, so this Refresh function will end up being called upon deserialization.



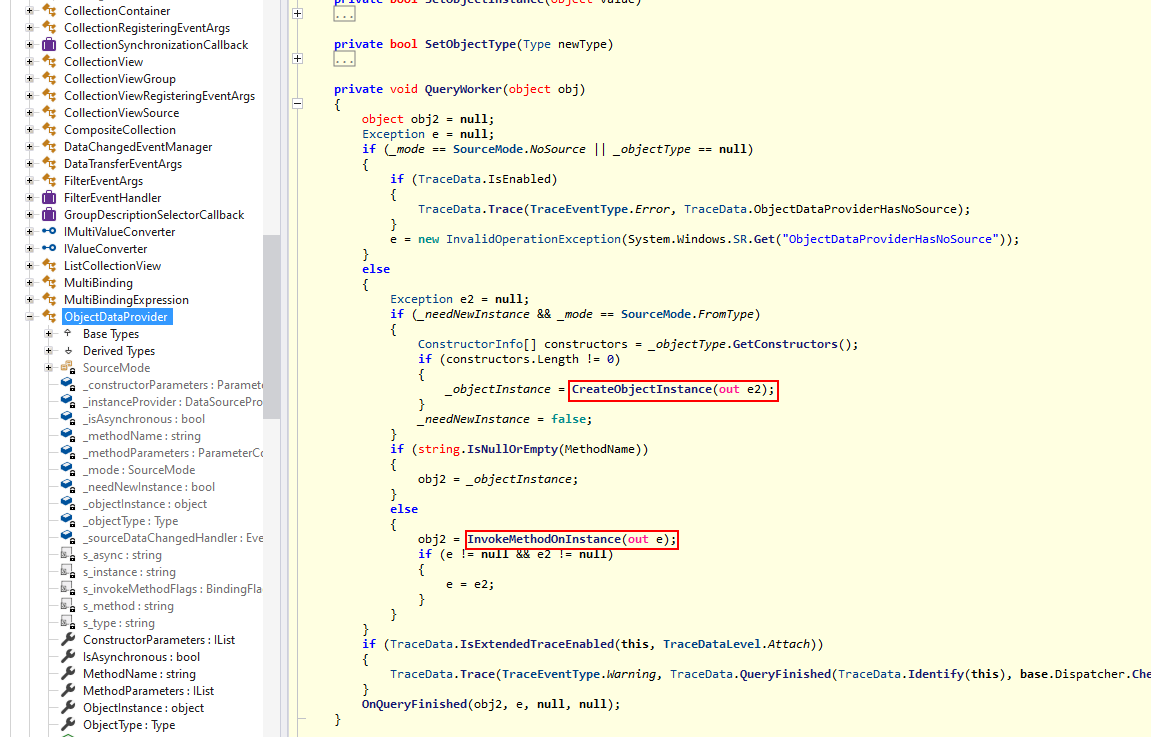
Refresh is a method defined in DataSourceProvider, and we can see that it simply calls the BeginQuery method.



BeginQuery is an empty method in DataSourceProvider, but it is overridden in ObjectDataProvider, so this is where the execution flow continues. Inside the implementation of BeginQuery, we can see that the QueryWorker function is called.



Finally, we end up in the QueryWorker function in ObjectDataProvider, and we can see that an object instance is created, and additionally that a method is invoked if the MethodName parameter is defined.



Going back to the [documentation](https://learn.microsoft.com/en-us/dotnet/api/system.windows.data.objectdataprovider?view=windowsdesktop-7.0) again, we can see that ObjectDataProvider has the following fields (among others):

* ObjectType: Used to set the type of object to create an instance of
* MethodName: Set the name of a method to call when creating the object
* MethodParameters: The list of parameters to be passed to the method

So using just these three fields, we should be able to create an instance of an arbitrary object, and call an arbitrary method with arbitrary parameters, all without invoking a single method. We can test this out ourselves with a short C# program, which uses ObjectDataProvider to create an instance of System.Diagnostics.Process and invokes the Start method with parameters to launch the calculator application.

Note: Don't worry about actually compiling/running the program below, we will get to exploit development in the following sections.

Code: csharp

using System.Windows.Data;

namespace ODPExample

{

internal class Program

{

static void Main(string[] args)

{

ObjectDataProvider odp = new ObjectDataProvider();

odp.ObjectType = typeof(System.Diagnostics.Process);

odp.MethodParameters.Add("C:\\Windows\\System32\\cmd.exe");

odp.MethodParameters.Add("/c calc.exe");

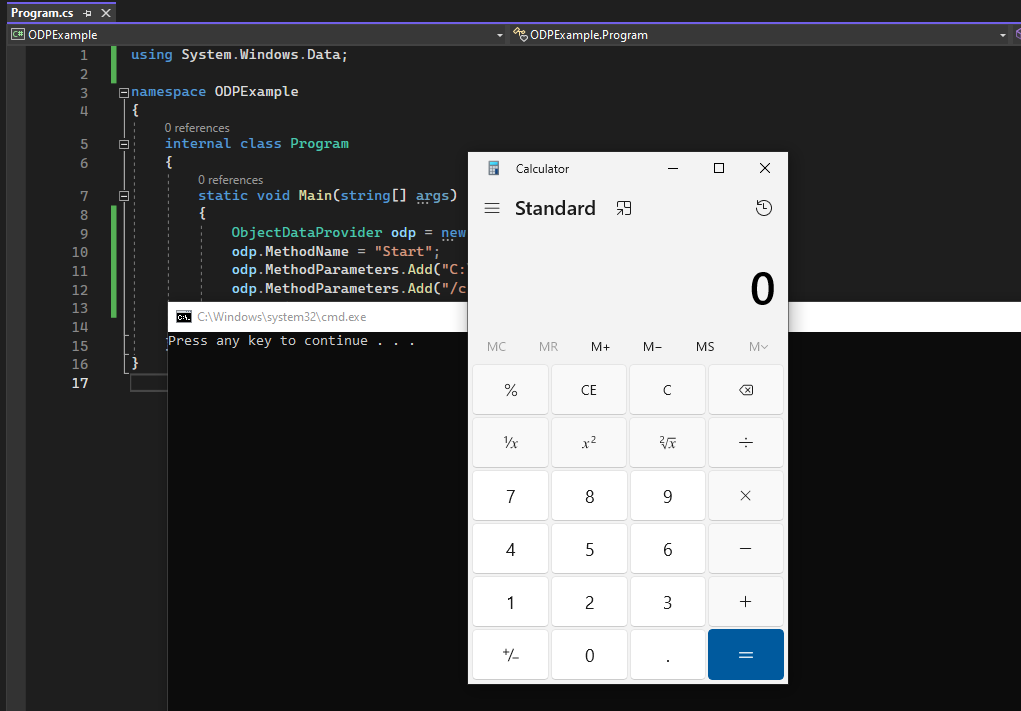
odp.MethodName = "Start";

}

}

}

Using ObjectDataProvider, we can execute arbitrary commands without invoking any methods directly/explicitly.



**Conclusion**

We now have a gadget that enables remote code execution upon deserialization. This is an important part of exploiting .NET deserialization vulnerabilities, because we will typically can not use serialized data to directly invoke methods.