<https://kanithi.wordpress.com/2010/05/12/how-to-create-a-dynamic-rule-or-authorization-in-wcf-service/>

[Services Development](https://msdn.microsoft.com/en-us/library/ff921346.aspx) [Improving Web Services Security: Scenarios and Implementation Guidance for WCF](https://msdn.microsoft.com/en-us/library/ff650794.aspx)  [How-to Articles](https://msdn.microsoft.com/en-us/library/ff648360.aspx)

[How to: Perform Input Validation in WCF](https://msdn.microsoft.com/en-us/library/ff647875.aspx)

[How to: Perform Message Validation with Schema Validation in WCF](https://msdn.microsoft.com/en-us/library/ff647820.aspx)

[How to: Use basicHttpBinding with Windows Authentication and TransportCredentialOnly in WCF from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648505.aspx)

[How to: Use Certificate Authentication and Message Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648360.aspx)

[How to: Use Certificate Authentication and Transport Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff650785.aspx)

[How to: Use Delegation for Flowing the Original Caller Credentials to the Back End in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff650896.aspx)

[How to: Use Health Monitoring to Instrument a WCF Service for Security](https://msdn.microsoft.com/en-us/library/ff649765.aspx)

[How to: Use netTcpBinding with Windows Authentication and Message Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648534.aspx)

[How to: Use netTcpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff647180.aspx)

[How to: Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF](https://msdn.microsoft.com/en-us/library/ff648976.aspx)

[How to: Use the SQL Server Role Provider with Username Authentication in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff647040.aspx)

[How to: Use the SQL Server Role Provider with Windows Authentication in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648806.aspx)

[How to: Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF from Windows Forms](https://msdn.microsoft.com/en-us/library/ff647294.aspx)

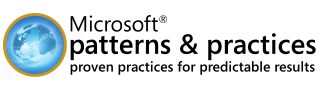
[How to: Use Username Authentication with Transport Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff649647.aspx)

[How to: Use wsHttpBinding with Username Authentication and TransportWithMessageCredentials in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648840.aspx)

[How to: Use wsHttpBinding with Windows Authentication and Message Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff650619.aspx)

[How to: Use wsHttpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms](https://msdn.microsoft.com/en-us/library/ff648431.aspx)

How to: Use Certificate Authentication and Message Security in WCF Calling from Windows Forms

[](https://msdn.microsoft.com/en-us/practices/default.aspx)

Applies to

* Microsoft Windows Communication Foundation (WCF) 3.5
* Microsoft Visual Studio 2008

Summary

This how-to article walks you through the process of using client certificates and message security to authenticate your users. The article shows you how to create and install client and service certificates during development, configure the WCF service and client to use the respective certificates, and test the service with a sample WCF client.

Contents

* [Objectives](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Objectives)
* [Overview](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Overview)
* [Summary of Steps](https://msdn.microsoft.com/en-us/library/ff648360.aspx#SummaryOfSteps)
* [Step 1: Create a Sample WCF Service](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step1)
* [Step 2: Configure wsHttpBinding with Certificate Authentication and Message Security](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step2)
* [Step 3: Create and Install a Service Certificate](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step3)
* [Step 4: Configure the Service Certificate for the WCF Service](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step4)
* [Step 5: Create a Test Client](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step5)
* [Step 6: Add a WCF Service Reference to the Client](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step6)
* [Step 7: Create and Install the Client Certificate for Authentication](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step7)
* [Step 8: Configure the Client Certificate in the WCF Client Application](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step8)
* [Step 9: Test the Client and WCF Service](https://msdn.microsoft.com/en-us/library/ff648360.aspx#Step9)
* [Additional Resources](https://msdn.microsoft.com/en-us/library/ff648360.aspx#AdditionalResources)

Objectives

* Learn how to create and use a temporary certificate for authentication and message security.
* Learn where to store the temporary certificate.
* Learn how to troubleshoot common errors related to temporary certificates, authentication, and message security in WCF.

Overview

When developing a WCF service that uses X.509 certificates to provide client authentication and message security, it is necessary to work with temporary certificates. This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

* **Peer trust** validates the certificate directly.
* **Chain trust** validates the certificate against the issuer of a certificate known as a root authority.

This how-to article discusses the chain trust option because it is the most commonly used approach in Business-to-Business (B2B) scenarios.

To use chain trust validation during development time, you create a self-signed root certificate authority (CA) and place it in the Trusted Root Certification Authority store of the client and service machines. The certificate used by the WCF client for client authentication and the WCF service for service authentication and message protection is then created and signed by the root self-signed certificate and installed in the LocalMachine store.

You will use makecert.exe to create a certificate to act as your root CA. You will then use your root CA certificate to sign additional certificates for your WCF service and client. Finally, you will configure the WCF client and service to use your temporary certificate.

Summary of Steps

* Step 1: Create a Sample WCF Service
* Step 2: Configure wsHttpBinding with Certificate Authentication and Message Security
* Step 3: Create and Install a Service Certificate
* Step 4: Configure the Service Certificate for the WCF Service
* Step 5: Create a Test Client
* Step 6: Add a WCF Service Reference to the Client
* Step 7: Create and Install the Client Certificate for Authentication
* Step 8: Configure the Client Certificate in the WCF Client Application
* Step 9: Test the Client and WCF Service

Step 1: Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the **File** menu, click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path** (e.g., http://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Browse to your WCF service (i.e., http://localhost/WCFTestService/Service.svc).

You should see details of your WCF service.

Step 2: Configure wsHttpBinding with Certificate Authentication and Message Security

In this step, you configure the WCF service to use certificate authentication and message security.

1. Right-click the Web.config file of the WCF service and then choose the **Edit WCF Configuration** option.
2. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name]** and set the **Name** attribute to **wsHttpEndpoint**.

By default, the name will be empty because it is an optional attribute.

1. Click the **Identity** tab and then delete the **Dns** attribute value.
2. In the Configuration Editor, select the Bindings folder.
3. In the **Bindings** section, choose **New Binding Configuration**.
4. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
5. Click **OK**.
6. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
7. Click the **Security** tab.
8. Make sure that the **Mode** attribute is set to **Message**, which is the default setting.
9. Set the **MessageClientCredentialType** to **Certificate** by selecting this option from the drop-down list.
10. In the **Configuration** section, select the **wsHttpEndpoint** node.
11. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by selecting this option from the drop-down list.

This associates the binding configuration setting with the binding.

1. In the Configuration Editor, on the **File** menu, select **Save**.
2. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:
3. <!--<identity>
4. <dns value="" />
5. </identity>-->
6. In Visual Studio, verify your configuration. The configuration should look as follows:
7. …
8. <bindings>
9. <wsHttpBinding>
10. <binding name="wsHttpEndpointBinding">
11. <security>
12. <message clientCredentialType="Certificate" />
13. </security>
14. </binding>
15. </wsHttpBinding>
16. </bindings>
17. <services>
18. <service behaviorConfiguration="ServiceBehavior" name="Service">
19. <endpoint address="" binding="wsHttpBinding"
20. bindingConfiguration="wsHttpEndpointBinding"
21. name="wsHttpEndpoint" contract="IService">
22. <!--<identity>
23. <dns value="" />
24. </identity>-->
25. </endpoint>
26. <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
27. </service>
28. </services>
29. …

Step 3: Create and Install a Service Certificate

In this step, you create a temporary service certificate and install it in the local store. This certificate will be used for service authentication and to encrypt the message, thereby protecting any other sensitive data.

Creating and installing the certificate is outside the scope of this How To article. For detailed steps on how to do this, see “How To - Create and Install Temporary Certificates in WCF for Message Security During Development.”

|  |
| --- |
| **Ff648360.note(en-us,PandP.10).gifNote:** |
| If you are running on Microsoft Windows XP, give the certificate permissions for the ASPNET identity instead of the NT Authority\Network Service identity because the Internet Information Services (IIS) process runs under the ASPNET account. The temporary certificate should be used for development and testing purposes only. For actual production deployment, you will need to obtain a valid certificate from a certificate authority (CA). |

Step 4: Configure the Service Certificate for the WCF Service

In this step, you configure the WCF service to use the temporary certificate you created in the previous step.

1. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** node.
2. Click **Add**.
3. In the **Service Behavior Element Extensions** dialog box, select the **serviceCredentials** option and then click **Add**.
4. Expand the **serviceCredentials** node and then select the **serviceCertificate** node.
5. Set the **FindValue** attribute to the name of the service certificate that you have created; for example, "CN=tempCertServer".
6. Leave the default settings for **StoreLocation** and **StoreName**.
7. In the Configuration Editor, on the **File** menu, click **Save**.
8. In Visual Studio, verify your configuration. The configuration should look as follows.
9. ...
10. <behaviors>
11. <serviceBehaviors>
12. <behavior name="ServiceBehavior">
13. <serviceMetadata httpGetEnabled="true" />
14. <serviceDebug includeExceptionDetailInFaults="false" />
15. <serviceCredentials>
16. <serviceCertificate findValue="CN=tempCertServer" />
17. </serviceCredentials>
18. </behavior>
19. </serviceBehaviors>
20. </behaviors>
21. ...

Step 5: Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 6: Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your Client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF Service (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the **Web reference name** field, change ServiceReference1 to **WCFTestService**.
4. Click **Add Reference**.

A reference to WCFTestService should appear beneath Web References in your Client project.

Step 7: Create and Install the Client Certificate for Authentication

In this step, you create a temporary client certificate by using the root CA created in Step 3 above, and install it in the local store. This certificate will be used for client authentication and to encrypt the message, thereby protecting any other sensitive data.

1. Copy the root CA certificate (RootCATest.cer) and private key file (RootCATest.pvk), created as part of Step 3, to the client machine.
2. Open a Visual Studio command prompt and browse to the location where you copied the root CA certificate and private key file.
3. Run the following command for creating a certificate signed by the root CA certificate:
4. makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=tempCert" -ic RootCATest.cer -sr CurrentUser -ss my -sky signature -pe tempCert.cer
5. In the **Enter Private Key Password** dialog box, enter the password for the root CA private key file created as part of the Step 3 above, and then click **OK**.

For more information and detailed steps, see “How to: Create and Install Temporary Certificates in WCF for Message Security During Development.”

Step 8: Configure the Client Certificate in the WCF Client Application

In this step, you configure the WCF client to use the temporary certificate you created in the previous step.

1. In your test client, right-click the App.config file and then click **Edit WCF Configuration**.
2. In the Configuration Editor, expand the **Advanced** node, select **Endpoint Behaviors**, and then select **New Endpoint Behavior Configuration**.
3. Click **Add**.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **clientCredentials** and then click **Add**.
5. Expand the **clientCredentials** node, select the **clientCertificate** node, and then set the **FindValue** attribute to the subject name of the client certificate that you created and installed in Step 7; for example, "CN=tempCertClient".
6. Leave the default **StoreLocation** attribute set to **CurrentUser** as is.
7. In the Configuration Editor, expand the **Client** node, expand the **Endpoints** node, and then select the **wsHttpEndpoint** node.
8. Set the **BehaviorConfiguration** attribute to **NewBehavior** by choosing this option from the drop-down list.

This is the endpoint behavior you just created.

1. In the Configuration Editor, on the **File** menu, click **Save**.
2. In Visual Studio, verify your configuration. The configuration should look as follows.
3. <system.serviceModel>
4. <behaviors>
5. <endpointBehaviors>
6. <behavior name="NewBehavior">
7. <clientCredentials>
8. <clientCertificate findValue="CN=tempCertClient"/>
9. </clientCredentials>
10. </behavior>
11. </endpointBehaviors>
12. </behaviors>
13. ...
14. <client>
15. <endpoint address="http://<<service address>>"
16. behaviorConfiguration="NewBehavior" binding="wsHttpBinding"
17. bindingConfiguration="wsHttpEnpoint1" contract="ServiceReference1.IService"
18. name="wsHttpEndpoint">
19. <identity>
20. <certificate encodedValue="<<Encode Value>>" />
21. </identity>
22. </endpoint>
23. </client>
24. </system.serviceModel>

Step 9: Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your Client project, drag a button control onto your form.
2. Double-click the button control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:
4. private void button1\_Click(object sender, EventArgs e)
5. {
6. WCFTestService.ServiceClient myService = new
7. WCFTestService.ServiceClient();
8. MessageBox.Show(myService.GetData(123));
9. myService.Close();
10. }
11. Right-click the Client project and then click **Set as Startup Project**.
12. Run the Client application by pressing F5 or CTRL+F5.

When you click the button on the form, the message “You entered: 123” should appear.

Additional Resources

* For more information on how to work with temporary certificates, see [How to: Create Temporary Certificates for Use During Development](https://msdn.microsoft.com/en-us/library/ms733813.aspx).
* For more information on how to view certificates using the Microsoft Management Console (MMC) snap in, see [How to: View Certificates with the MMC Snap-in](https://msdn.microsoft.com/en-us/library/ms788967.aspx).
* For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see [Differences Between Service Certificate Validation Done by Internet Explorer and WCF](https://msdn.microsoft.com/en-us/library/aa702599.aspx).
* For more information on differences in certificate validation between protocols, see [Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security](https://msdn.microsoft.com/en-us/library/aa702579.aspx).

# WCF Service with custom username password authentication



[**Fayaz Soomro**](https://www.codeproject.com/script/Membership/View.aspx?mid=1270397), 25 Jul 2010 [CPOL](http://www.codeproject.com/info/cpol10.aspx)

|  |  |
| --- | --- |
| https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-fill-md.png  https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.91 (61 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti) |  | |

This article describes custom username password authentication without the need of certificate installation on the client side.

* [**Download source and executable - 251 KB**](https://www.codeproject.com/KB/WCF/CustomUserNamePassAuth2/CustomUsernamePasswordAuth.zip)

## **Tools used**

1. [Pluralsight Self-Cert Tool](http://www.pluralsight.com/community/cfs-file.ashx/__key/CommunityServer.Blogs.Components.WeblogFiles/keith.SampleCode/SelfCert.zip). This tool is provided by Pluralsight to create and install certificates.
2. [WinHttpCertCfg.exe](http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en). Windows HTTP Services Certificate Configuration Tool is a command line tool to grant specific users read right access on a certificate's private key file.
3. Environment: Visual Studio 2010, and IIS7 or above.

## **Contents**

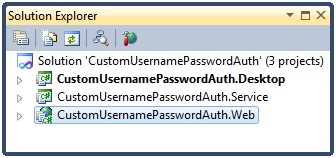
1. [Introduction](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-1)
2. [Creating the service](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-2)
3. [Configuring the service](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-3)
4. [Configuring IIS and publishing the website](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-4)
5. [Installing the server side certificate](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-5)
6. [Setting up the client](https://www.codeproject.com/Articles/96028/WCF-Service-with-custom-username-password-authenti#h-6)

## **Introduction**

Windows Communication Foundation comes with a rich set of security features such as transport level message and transport with message; each security type has its own advantages and overheads as well. My application has lots of diverse clients used to connect with the service, and they have to be authenticated from the database, so the best possible solution is message level security using custom username - password authentication. After digging in to the net, I found pieces of information, and with some effort, I implemented a concrete solution which I am hoping is helpful for others.

## **Creating the service**

The solution is created using VS2010, and contains three projects: the WCF Service, the website, and the desktop application which is the client application.



The WCF Service just contains a function GetServertime():

Hide   Copy Code

[ServiceContract]

public interface IService1

{

[OperationContract]

string GetServertime();

}

[ServiceBehavior(IncludeExceptionDetailInFaults = true)]

public class Service1 : IService1

{

public string GetServertime()

{

return DateTime.Now.ToString();

}

}

We create a class and name it UserNamePassValidator. We the implement this code in it:

Hide   Copy Code

using System;

using System.ServiceModel;

namespace CustomUsernamePasswordAuth.Service

{

class UserNamePassValidator :

System.IdentityModel.Selectors.UserNamePasswordValidator

{

public override void Validate(string userName, string password)

{

if( userName==null || password==null)

{

throw new ArgumentNullException();

}

if (!(userName == "fayaz" && password == "soomro") )

{

throw new FaultException("Incorrect Username or Password");

}

}

}

}

This class must be derived from System.IdentityModel.Selectors.UserNamePasswordValidator and override the Validate method. And to validate the user, use any data source; in this example, we will use a hard coded value.

## **Creating the web application**

Add a reference to the service in the web application. Add a text file and rename it to UserNamePassService.svc, and add the following line of code:

Hide   Copy Code

<%@ ServiceHost Language="C#" Debug="true"

Service="CustomUsernamePasswordAuth.Service.Service1" %>

## **Configuring the Web Service**

Modify the web.config and add following lines in it.

Add a service behavior and name it Behavior1. Enable the service meta data by adding <serviceMetadata httpGetEnabled="true"/> so that when we add a service reference into the client application, it fetched the information and creates the proxy classes for us. And the essential part is the service certificate. Certificate creation will be covered in a later section, but now, we have to remember the certificate settings. FindValye="MyWebSite" will be the subject for the certificate CN=MyWebSite, and you can change this value to your domain name or project name.

Set the usernamepasswordvalidation mode to custom, and customUsernameapsswordValidator has to be specify the custom validation class and namespace.

Hide   Copy Code

<system.serviceModel>

<behaviors>

<serviceBehaviors>

<behavior name="Behavior1">

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

<serviceCredentials>

<serviceCertificate findValue="MyWebSite"

storeLocation="LocalMachine"

storeName="My"

x509FindType="FindBySubjectName" />

<userNameAuthentication userNamePasswordValidationMode="Custom"

customUserNamePasswordValidatorType="CustomUsernamePasswordAuth.

Service.UserNamePassValidator, CustomUsernamePasswordAuth.Service" />

</serviceCredentials>

Set up the binding configuration as shown below. Name it Binding1 and set the security mode as Message and clientCredentialType as "username".

Hide   Copy Code

<bindings>

<wsHttpBinding>

<binding name="Binding1">

<security mode="Message">

<message clientCredentialType="UserName"/>

</security>

</binding>

</wsHttpBinding>

</bindings>

Now we will set up the service endpoint. There are two endpoints: wsHttp endpoint, and Mex end point for metadata exchange. The base address is http://localhost/. The fully qualified service address will be http://localhost/UserNamePassService.svc.

Hide   Copy Code

<services>

<service behaviorConfiguration="Behavior1"

name="CustomUsernamePasswordAuth.Service.Service1">

<endpoint address="" binding="wsHttpBinding"

bindingConfiguration="Binding1"

contract="CustomUsernamePasswordAuth.Service.IService1" />

<endpoint address="mex" binding="mexHttpBinding"

contract="IMetadataExchange" />

<host>

<baseAddresses>

<add baseAddress="http://localhost/" />

</baseAddresses>

</host>

</service>

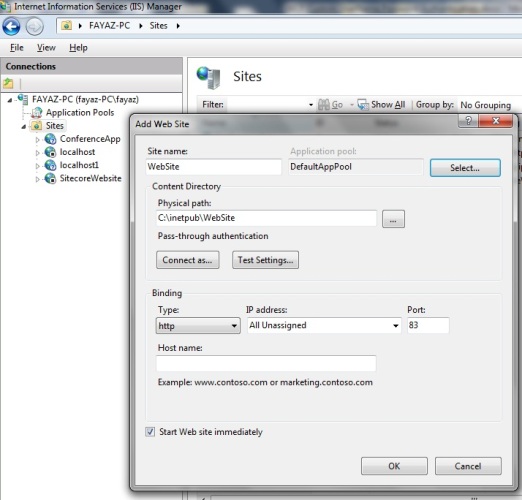
</services>

</system.serviceModel>

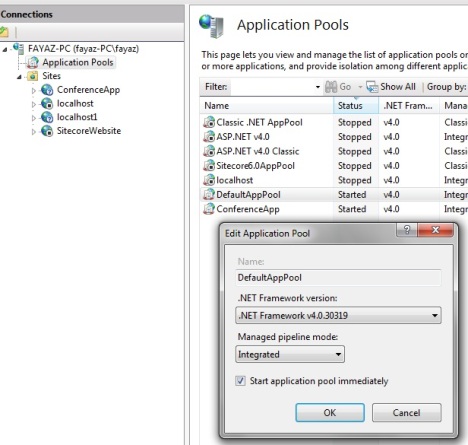
**Note**: if the website is going to be hosted on a specific port in IIS, as in this example, we have hosted the website in IIS on port 83, http://localhost:83/UserNamePassService.svc, we don't need to change the port in the configuration file and leave the baseAddress as "http://localhost/".

## **Creating the site in IIS 7**

Open IIS Manager. Right click Sites and Add Website. Name it as WebSite, set Application pool to DeafaultAppPool, and select the physical path and set port to 83. As shown below:

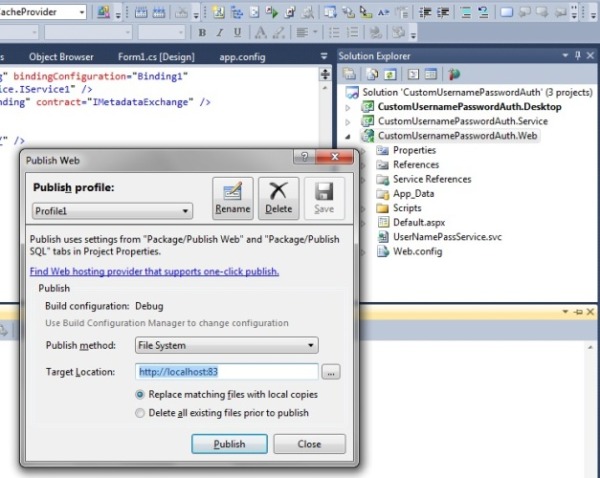


Set the DefaultAppPool Framework version to 4.0.



### **Publish Site to IIS**

Right click the website project in Solution Explorer and publish it. Select Publish method as File system, and Target location as http://localhost:83, as shown in the figure below:



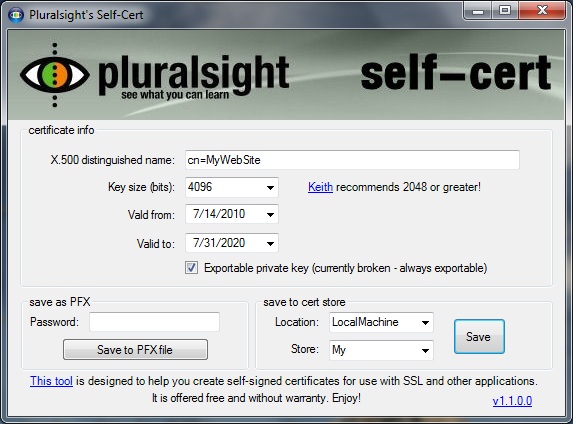
Browse the site. Open Internet Explorer and type http://localhost:83/UserNamePassService.svc. You will see the error that X.509 could not be found.



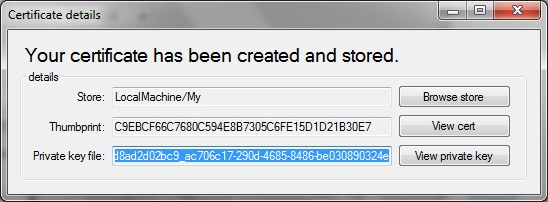
## **Installing the certificate**

Download the Pluralsight SelfCert from the link given at the beginning of the article. Run the tool as Administrator; otherwise, it will crash.

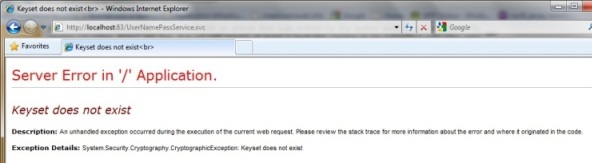
Configure the settings to install the certificate; refer the screen below.



After making the required changes, click the Save button and then you will see the screen below:



After the installation of the certificate, browse the site again, but this time, you should see a different error as shown in the screen below:



This error means that the default application pool does not have access rights to the certificate's private key, so now, we have to give read access to the default application pool to do this.

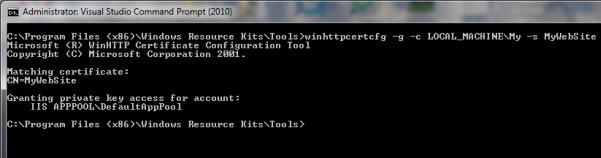
Download WinHttpCertCfg.exe from the link given at the beginning of the article. This tool is a command line tool. After installing the tool, run the following command on the command prompt as Administrator.

Hide   Copy Code

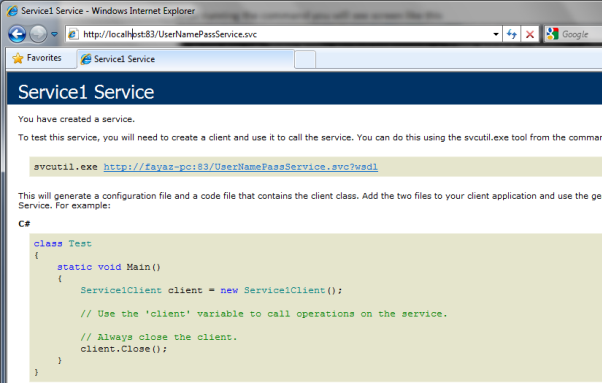
C:\Program Files (x86)\Windows Resource Kits\Tools>winhttpcertcfg

-g -c LOCAL\_MACHINE\My -s MyWebSite -a DefaultAppPool

After running the command, you will see the screen like this:

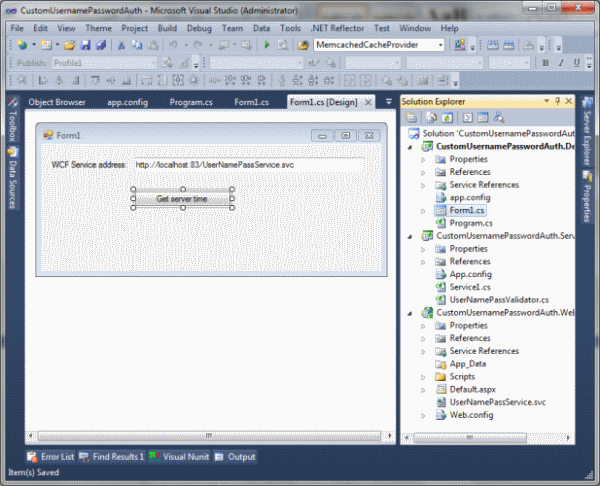


Now browse the site again. And verify the service, it should be up.

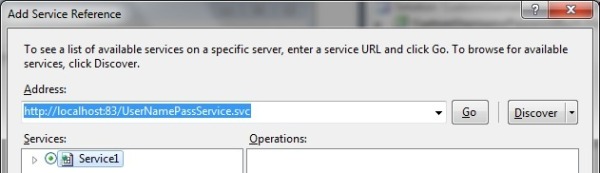


## **The final step is to create a client to consume the service**

The client application is the desktop application, and just contains the address textbox and the button to get the server time.



Now add the service reference to the project:



Add the code to the Button\_click event:

Hide   Copy Code

private void button1\_Click(object sender, EventArgs e)

{

string time = "";

*// Method 1: Create the client using the configuration file*

Service1Client c = new Service1Client();

c.ClientCredentials.UserName.UserName = "fayaz";

c.ClientCredentials.UserName.Password = "soomro";

c.ClientCredentials.ServiceCertificate.Authentication.CertificateValidationMode =

X509CertificateValidationMode.None;

time = c.GetServertime();

MessageBox.Show(time);

*// Method 2: Creating the client by creating endpoint and binding through coding*

var ServiceendPoint = new EndpointAddress(new Uri(txtServiceAddress.Text),

EndpointIdentity.CreateDnsIdentity("MyWebSite"));

var binding = new WSHttpBinding();

binding.Security.Mode = SecurityMode.Message;

binding.Security.Message.ClientCredentialType = MessageCredentialType.UserName;

var result = new Service1Client(binding, ServiceendPoint);

result.ClientCredentials.ServiceCertificate.Authentication.CertificateValidationMode =

X509CertificateValidationMode.None;

result.ClientCredentials.UserName.UserName = "fayaz";

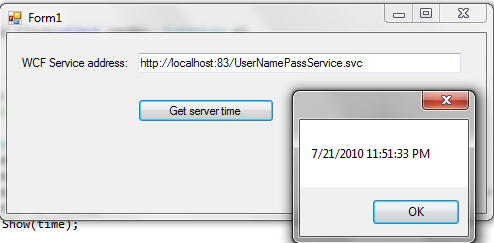
result.ClientCredentials.UserName.Password = "soomro";

time = result.GetServertime();

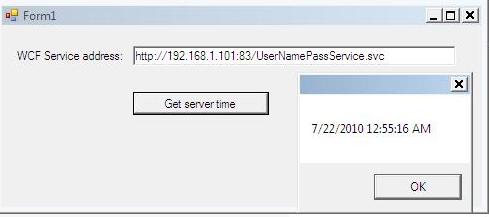
MessageBox.Show(time);

}

Run the application:



Running the client from another PC to make sure everything works fine:



## **Conclusion**

I 'm sure this project will be useful for developers who want to implement custom security. I tried my best to describe each step with a screenshot. I hope you've enjoyed this article. If you like this article, please let me know :). If you have any questions, please feel free to contact me at fayaziiui@gmail.com.

Implementing Dynamic Authorization for a WCF service using SQL providers

★★★★★

★★★★

★★★

★★

★

avatar of mohamed-m-malek[Mohamed M Malek](https://social.msdn.microsoft.com/profile/Mohamed+M+Malek)May 11, 2011[0](https://blogs.msdn.microsoft.com/momalek/2011/05/11/implementing-dynamic-authorization-for-a-wcf-service-using-sql-providers/#respond)

* [0](https://blogs.msdn.microsoft.com/momalek/2011/05/11/implementing-dynamic-authorization-for-a-wcf-service-using-sql-providers/)
* [0](https://blogs.msdn.microsoft.com/momalek/2011/05/11/implementing-dynamic-authorization-for-a-wcf-service-using-sql-providers/)

Using the ASP.NET SQL membership provider to authenticate and authorize calls to WCF services is not an uncommon scenario. But the problem is in the authorization part. Usually to authorize access to WCF service methods this is done using static hard coded attributes decorating the methods definition. So basically you would say that this method is accessible to anyone in the specific role. This is shown below.

        [PrincipalPermission(SecurityAction.Demand, Role = "Managers")]

        public string GetData(int value)

        {

            return string.Format("You entered: {0}", value);

        }

Now the more challenging scenario is when you need to configure the access rules in the runtime like from a database or configuration files rather than in the service code.

To implement this scenario you would need to implement a custom WCF service behaviour along with a custom service authorization manager.

First the behaviour goes like this.

public class SqlProviderSecurityBehavior : IServiceBehavior

{

    private ISqlProviderSecuritySettings settings;

    public SqlProviderSecurityBehavior(ISqlProviderSecuritySettings settings)

    {

        this.settings = settings;

    }

    #region IServiceBehavior Members

    public void AddBindingParameters(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase, System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints, System.ServiceModel.Channels.BindingParameterCollection bindingParameters)

    {

        if (this.settings.MembershipProvider != null)

        {

            ServiceCredentials cr = bindingParameters.Find<ServiceCredentials>();

            if (cr == null)

            {

                cr = new ServiceCredentials();

                bindingParameters.Add(cr);

            }

            // set membership provider

            SqlMembershipProvider sqlMembership = new SqlMembershipProvider();

            NameValueCollection config = new NameValueCollection();

            //if specifying the actual connection string, use reflection to set the values

            if (!string.IsNullOrEmpty(this.settings.MembershipProvider.ConnectionString))

            {

                Type t = typeof(SqlMembershipProvider);

                FieldInfo fi = t.GetField("\_sqlConnectionString", BindingFlags.NonPublic |BindingFlags.Instance);

                fi.SetValue(sqlMembership, this.settings.MembershipProvider.ConnectionString);

                sqlMembership.ApplicationName = this.settings.MembershipProvider.ApplicationName;

            }

            else

            {

                //use Initialize method when specyfing connectionName… no reflection needed

                config.Add("connectionStringName", this.settings.MembershipProvider.ConnectionStringName);

                config.Add("applicationName", this.settings.MembershipProvider.ApplicationName);

                sqlMembership.Initialize("SqlMembershipProvider", config);

            }

            cr.UserNameAuthentication.UserNamePasswordValidationMode =UserNamePasswordValidationMode.MembershipProvider;

            cr.UserNameAuthentication.MembershipProvider = sqlMembership;

        }

        if (this.settings.RoleProvider != null)

        {

            // set role provider

            serviceHostBase.Authorization.PrincipalPermissionMode =PrincipalPermissionMode.UseAspNetRoles;

            SqlRoleProvider sqlRoleProvider = new SqlRoleProvider();

            NameValueCollection config = new NameValueCollection();

            //if specifying the actual connection string, use reflection to set the values

            if (!string.IsNullOrEmpty(this.settings.RoleProvider.ConnectionString))

            {

                Type t = typeof(SqlRoleProvider);

                FieldInfo fi = t.GetField("\_sqlConnectionString", BindingFlags.NonPublic |BindingFlags.Instance);

                fi.SetValue(sqlRoleProvider, this.settings.RoleProvider.ConnectionString);

                sqlRoleProvider.ApplicationName = this.settings.RoleProvider.ApplicationName;

            }

            else

            {

                //use Initialize method when specyfing connectionName… no reflection needed

                config.Add("connectionStringName", this.settings.RoleProvider.ConnectionStringName);

                config.Add("applicationName", this.settings.RoleProvider.ApplicationName);

                sqlRoleProvider.Initialize("SqlRoleProvider", config);

            }

            serviceHostBase.Authorization.RoleProvider = sqlRoleProvider;

            serviceHostBase.Authorization.ServiceAuthorizationManager = newSqlAuthorizationManager(sqlRoleProvider);

            // store as service extension

            RoleProviderServiceHostExtension ext = new RoleProviderServiceHostExtension(sqlRoleProvider);

            serviceHostBase.Extensions.Add(ext);

        }

    }

    public void ApplyDispatchBehavior(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase)

    {

    }

    public void Validate(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBaseserviceHostBase)

    {

    }

    #endregion

}

As you can see in this WCF behaviour it sets the service authorization manager to be our custom class that is implemented as follows.

public class SqlAuthorizationManager : ServiceAuthorizationManager

{

    private SqlRoleProvider sqlRoleProvider;

    public SqlAuthorizationManager(SqlRoleProvider \_sqlRoleProvider) : base()

    {

        sqlRoleProvider = \_sqlRoleProvider;

    }

    protected override bool CheckAccessCore(OperationContext operationContext)

    {

        bool baseResult = base.CheckAccessCore(operationContext);

        //For mex support (starting WCF service, etc.)

        //NOTE: Other than for service startup this will NOT be true because the WCF

        //configuration dictates that WindowsCredentials must be sent and Anonymous users

        //are NOT allowed.

        if (operationContext.ServiceSecurityContext.IsAnonymous) return true;

        //Extract the identity token of the current context user making the call to this service

        IIdentity Identity = operationContext.ServiceSecurityContext.PrimaryIdentity;

        //Prior to proceeding, throw an exception if the user has not been authenticated at all

        if (!Identity.IsAuthenticated)

        {

            throw new SecurityTokenValidationException("Authenticated user is required to call this service.");

        }

        string[] roles = sqlRoleProvider.GetRolesForUser(Identity.Name);

        if (roles.Length <= 0)

        {

            throw new System.ServiceModel.Security.SecurityAccessDeniedException("User is not part of the service account role.");

        }

        if (!roles.Contains("The role you need to check comes here or can be dynamic"))

        {

            throw new System.ServiceModel.Security.SecurityAccessDeniedException("User is not part of the service account role.");

        }

        //this is the custom authorization rules in a custom table of your choosing

        ASPProvidersEntities entities = new ASPProvidersEntities();

        var userrule = (from serviceAuth in entities.ServiceAuthorizations

                        where serviceAuth.ServiceContractName == operationContext.EndpointDispatcher.ContractName && serviceAuth.Username == Identity.Name

                        select serviceAuth).SingleOrDefault();

        if (userrule == null)

        {

            throw new System.ServiceModel.Security.SecurityAccessDeniedException("User is not authorized to call this service.");

        }

        return baseResult;

    }

}

Happy coding