Table of contents

[Introduction and goal](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#Introduction and Goal)

[WCF FAQ Part 1 and 2](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#WCF FAQ Part 1 and 2 series)

[What are the core security features that WCF addresses?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#What are the core security features that WCF addresses)

[What is transport level and message level security?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#What is transport level and message level security)

[For which bindings are transport, message, and mixed mode supported?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#For which bindings are transport, message and mixed mode supported)

[What are the scenarios, advantages, and disadvantages of transport VS message security?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#So what are the scenarios, advantages and disadvantages of transport VS message security)

[Can you explain a simple example of how to implement transport security?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#Can you explain a simple example of how to implement transport security)

[Can you show a simple example of message level security?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#Can you show a simple example of message level security)

[What is the difference between BasicHttpBinding and WsHttpBinding?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#What is the difference between BasicHttpBinding and WsHttpBinding)

[Can you show the security differences between BasicHttpBinding and WsHttpBinding?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#Can you show the security differences between BasicHttpBinding VS WsHttpBinding)

[When should we use WsHttp as compared to BasicHttp?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#When should we use WsHttp as compared to BasicHttp)

[How can we enable Windows authentication on WCF using ‘BasicHttpBinding’?](http://www.codeproject.com/Articles/36732/WCF-FAQ-Part-security-related-FAQ#How can we enable windows authentication on WCF using ‘BasicHttpBinding’)

Introduction and goal

In this article, we will start with transport and message security understanding. We will then see simple code samples of how to implement transport and message security using WsHTTP bindings. We will also see the differences between ‘BasicHttpBinding’ and ‘WsHttpBinding’ with the help of a simple code. WCF security is a huge topic by itself, but I am sure with this article you will get a quick start of how to go about WCF security.

I have collected around 400 FAQ questions and answers on WCF, WPF, WWF, SharePoint, Design Patterns, UML, etc. Feel free to download these FAQ PDFs from my site: <http://www.questpond.com/>.

Other FAQ Articles

WCF FAQ [Part 1](http://www.codeproject.com/KB/aspnet/WCF.aspx): This is a 20 question FAQ for beginners which explains the basic concepts of WCF like end points, contracts, and bindings. It also discusses about the various hosting methodologies of WCF services. The article finally talks about bindings and one way operations in WCF.

WCF FAQ [Part 2](http://www.codeproject.com/KB/aspnet/WCFPart2.aspx): This FAQ covers 10 questions which talk about concepts like duplex contracts, hosting WCF on different protocols, MSMQ bindings, transaction isolation levels, and two way communication. The article finally talks about two queues: volatile and dead letter queue.

LINQ FAQ Part 1: [LINQNewbie.aspx](http://www.codeproject.com/KB/linq/LINQNewbie.aspx)

LINQ FAQ Part 2: [LINQFAQPart2](http://www.codeproject.com/KB/linq/LINQFAQPart2.aspx)

LINQ FAQ Part 3: [LINQ FAQ Part 3](http://www.codeproject.com/KB/linq/LINQFAQPart3.aspx)

What are the core security features that WCF addresses?

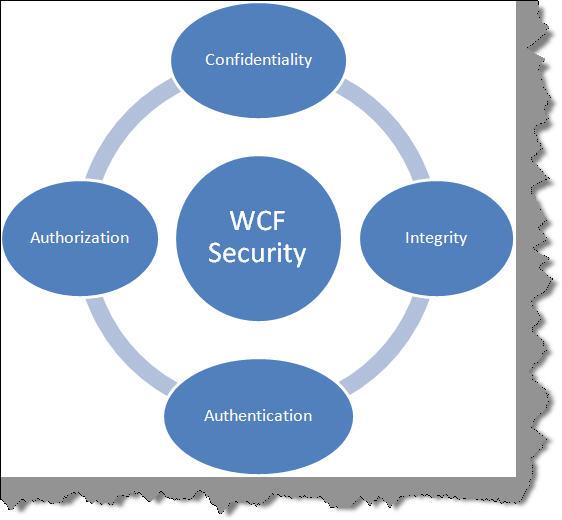
There are four core security features that WCF addresses:

Confidentiality: This feature ensures that information does not go in to the wrong hands when it travels from the sender to the receiver.

Integrity: This feature ensures that the receiver of the message gets the same information that the sender sent without any data tampering.

Authentication: This feature verifies who the sender is and who the receiver is.

Authorization: This feature verifies whether the user is authorized to perform the action they are requesting from the application.



What is transport level and message level security?

When we talk about WCF security, there are two aspects. The first is the data and the second is the medium on which the data travels, i.e., the protocol. WCF has the ability to apply security at the transport level (i.e., protocol level) and also at message level (i.e., data).

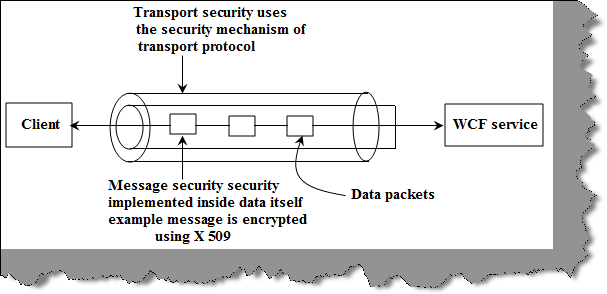


Figure: Transport and Message level security

Transport level security happens at the channel level. Transport level security is the easiest to implement as it happens at the communication level. WCF uses transport protocols like TCP, HTTP, MSMQ, etc., and each of these protocols have their own security mechanisms. One of the common implementations of transport level security is HTTPS. HTTPS is implemented over HTTP protocols with SSL providing the security mechanism. No coding change is required, it’s more about using the existing security mechanism provided by the protocol.

Message level security is implemented with message data itself. Due to this, it is independent of the protocol. One of the common ways of implementing message level security is by encrypting data using some standard encryption algorithm.

For which bindings are transport, message, and mixed mode supported?

Note: The below table is taken from the book Pro WCF: Practical Microsoft SOA Implementation -- Chris Peiris and Denis Mulder – Apress 2007.

Below is a table which shows which mode is supported for each binding. We are not discussing mixed mode. It’s nothing but a combination of transport and mixed modes. For instance, data encrypted and passed over WsHttp using HTTPS is a mixed mode security. Encryption is nothing but message security and HTTPS is a transport mode. In combination, they form mixed mode.

| Binding | Transport Mode? | Message Mode? | Mixed Mode? |
| --- | --- | --- | --- |
| BasicHttpBinding | Yes | Yes | Yes |
| WsHttpBinding | Yes | Yes | Yes |
| WsDualHttpBinding | No | Yes | No |
| NetTcpBinding | Yes | Yes | Yes |
| NetNamedPipeBinding | Yes | No | No |
| NetMsmqBinding | Yes | Yes | No |
| MsmqIntegrationBinding | Yes | No | No |

So what are the scenarios, advantages, and disadvantages of transport and message security?

|  | Transport | Message |
| --- | --- | --- |
| Scenarios when we should be using one of them | When there are no intermediate systems in between, this is the best methodology.  If it’s an intranet type of solution, this is the most recommended methodology. | When there are intermediate systems like one more WCF service through which message is routed, then message security is the way to go. |
| Advantages | Does not need any extra coding as protocol inherent security is used.  Performance is better as we can use hardware accelerators to enhance performance.  There is a lot of interoperability support and communicating clients do not need to understand WS security as it’s built in the protocol itself. | Provides end to end security as it’s not dependent on the protocol. Any intermediate hop in the network does not affect the application.  Supports a wide set of security options as it is not dependent on the protocol. We can also implement custom security. |
| Disadvantages | As it’s a protocol implemented security, it works only point to point.  As security is dependent on protocol, it has limited security support and is bound to the protocol security limitations. | Needs application refactoring to implement security.  As every message is encrypted and signed, there are performance issues.  Does not support interoperability with old ASMX webservices. |

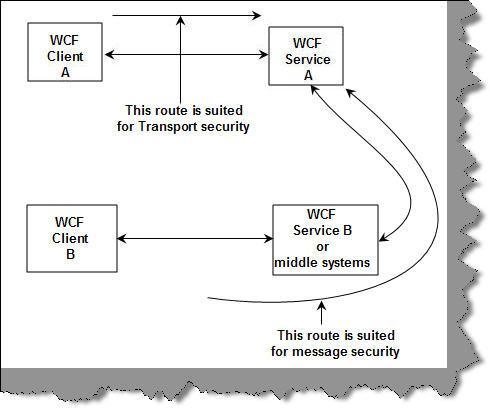


Figure: Route paths

Can you explain a simple example of how to implement transport security?

Let’s make a simple sample which will demonstrate how we can use transport security using WsHttp binding with HTTPS security.

Step 1: Create a simple service using a WCF project

The first step is to create a simple WCF project. So click on New Project and select WCF Service project. By default, a WCF project creates a default function GetData(). We will be using the same function for this sample.

Hide   Copy Code

public class Service1 : IService1

{

public string GetData(int value)

{

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

}

public CompositeType GetDataUsingDataContract(CompositeType composite)

{

if (composite.BoolValue)

{

composite.StringValue += "Suffix";

}

return composite;

}

}

Step 2: Enable transport level security in the web.config file of the service

The next step is to enable transport security in WsHttp binding. This is done using the Security XML tag as shown in the below code snippet.

Hide   Copy Code

<bindings>

<wsHttpBinding>

<binding name="TransportSecurity">

<security mode="Transport">

<transport clientCredentialType="None"/>

</security>

</binding>

</wsHttpBinding>

</bindings>

Step 3: Tie up the binding and specify HTTPS configuration

We need to now tie up the bindings with the end points. So use the bindingConfiguration tag to specify the binding name. We also need to specify the address where the service is hosted. Please note the HTTS in the address tag.

Change mexHttpBinding to mexHttpsBinding in the second end point.

Hide   Copy Code

<service name="WCFWSHttps.Service1" behaviorConfiguration="WCFWSHttps.Service1Behavior">

<!-- Service Endpoints -->

<endpoint address="https://localhost/WCFWSHttps/Service1.svc"

binding="wsHttpBinding" bindingConfiguration="TransportSecurity"

contract="WCFWSHttps.IService1"/>

<endpoint address="mex" binding="mexHttpsBinding" contract="IMetadataExchange"/>

</service>

In serviceMetadata, we also need to change httpGetEnabled to httpsGetEnabled.

Hide   Copy Code

<serviceBehaviors>

........

.........

<serviceMetadata httpsGetEnabled="true"/>

.........

.........

</serviceBehaviors>

Step 4: Make the web application HTTPS enabled

Now that we are done with the WCF service project creation and the necessary configuration changes are done, it’s time to compile the WCF service project and host it in an IIS application with HTTPS enabled.

We will be using ‘makecert.exe’ which is a free tool given by Microsoft to enable HTTPS for testing purposes. MakeCert (Makecert.exe) is a command-line tool that creates an X.509 certificate that is signed by a system test root key or by another specified key. The certificate binds a certificate name to the public part of the key pair. The certificate is saved to a file, a system certificate store, or both.

You can get it from C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\Bin or you can also get it from the Windows SDK.

You can type the below through your DOS prompt on “C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\Bin”. Please note “compaq-jzp37md0” is the server name so you need to replace it with your PC name.

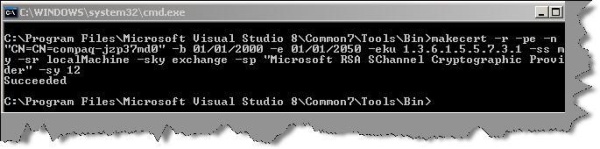
Hide   Copy Code

makecert -r -pe -n "CN= compaq-jzp37md0 " -b 01/01/2000

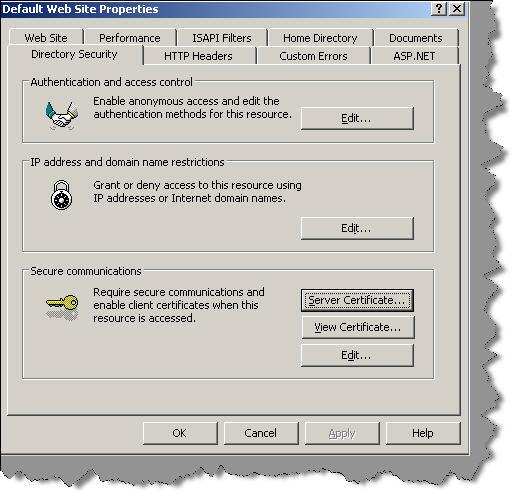
-e 01/01/2050 -eku 1.3.6.1.5.5.7.3.1 -ss my -sr localMachine -sky exchange

-sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12

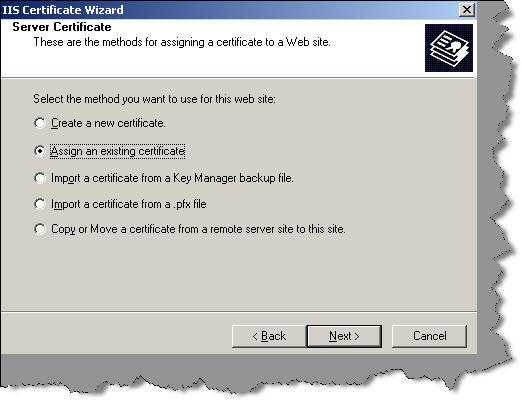
If you run the same through your command prompt, you should get a succeeded message as shown below:



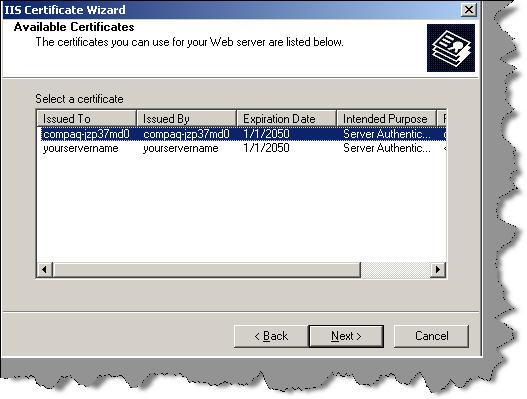
Now it’s time to assign this certificate to your IIS website. So go to IIS properties, click on the Directory Security tab, and you should see the Server Certificate tab.



Click on the Server Certificate tab and you will then be walked through an IIS certificate wizard. Click ‘Assign an existing certificate’ from the wizard.



You can see a list of certificates. The “compaq-jzp37md0” certificate is the one which we just created using ‘makecert.exe’.



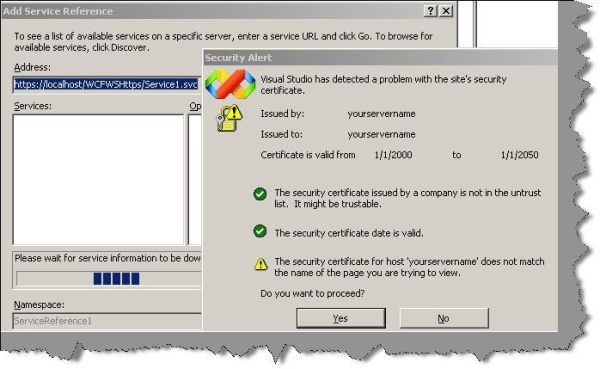
Now try to test the site without ‘https’ and you will get an error as shown below….That means your certificate is working.



Do not forget to enable IIS anonymous access.

Step 5: Consume the service in a web application

It’s time to consume the service application in ASP.NET web. So click on Add Service Reference and specify your service URL. You will be shown a warning box as in the below figure. When we used makecert.exe, we did not specify the host name as the service URL. So just let it go.



Step 6: Suppress the HTTPS errors

‘makecert.exe’ creates test certificates. In other words, it’s not signed by CA. So we need to suppress those errors in our ASP.NET client consumer. So we create a function called IgnoreCertificateErrorHandler which returns true even if there are errors. This function is attached as a callback to ServicePointManager.ServerCertificateValidationCallback.

In the same code, you can also see the service consuming code which calls the GetData function.

Hide   Shrink   Copy Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using WebApplicationConsumer.ServiceReference1;

using System.Net;

using System.Net.Security;

using System.Security.Cryptography.X509Certificates;

namespace WebApplicationConsumer

{

public partial class \_Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

ServicePointManager.ServerCertificateValidationCallback =

new RemoteCertificateValidationCallback(IgnoreCertificateErrorHandler);

Service1Client obj = new Service1Client();

Response.Write(obj.GetData(12));

}

public static bool IgnoreCertificateErrorHandler(object sender,

X509Certificate certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)

{

return true;

}

}

}

Step 7: Enjoy success

Now to the easiest step, compile your ASP.NET client and enjoy success.



Can you show a simple example of message level security?

Let’s do a simple example of message security using WsHTTP with X509 certificates.

Step 1: Create client and server certificates

Create two certificates, one for the server and the other for the client, using makecert.exe. You can get makecert.exe from the “C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\Bin” folder. You can go to the DOS prompt and run the below command:

Hide   Copy Code

makecert.exe -sr CurrentUser -ss My -a sha1 -n CN=WCfServer -sky exchange -pe

makecert.exe -sr CurrentUser -ss My -a sha1 -n CN=WcfClient -sky exchange -pe

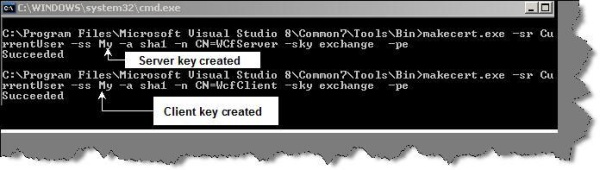
Below is a detailed explanation of the various attributes specified in makecert.exe.

| Attribute | Explanation |
| --- | --- |
| -sr | Specifies the Registry location of the certificate store. The SubjectCertStoreLocation argument must be either of the following:  currentUser - Specifies the Registry location HKEY\_CURRENT\_USER.  localMachine - Specifies the Registry location HKEY\_LOCAL\_MACHINE. |
| -ss | Specifies the name of the certificate store where the generated certificate is saved. |
| -a | Specifies the algorithm. Can be either MD5 or SHA1. |
| -n | Specifies a name for the certificate. This name must conform to the X.500 standard. The simplest method is to use the "CN=MyName" format. If the /n switch is not specified, the default name of the certificate is "Joe's Software Emporium". |
| -sky | Specifies the key type. Can be either exchange or signature. |
| -pe | This makes the key exportable. |

Note: Makecert.exe is a free tool provided by Microsoft which helps to create X.509 certificates signed by a system test root key or by another specified key. This is a test certificate and not a real one and should not be used for production purposes. For production, buy proper certificates from Thawte, Verisign, GeoTrust, etc.

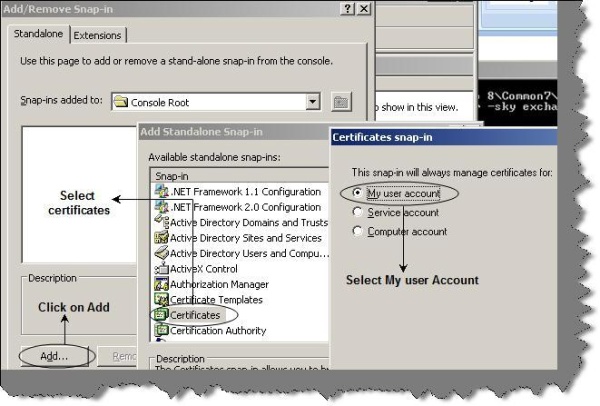
Currently we have specified that we want to create the client key with the ‘WcfClient’ name and server key with ‘WCFServer’. The certificates should be created for the current user and should be exportable.

Once you run the command, you should see the ‘Succeeded’ message as shown in the below figure. The figure shows keys created for both the server and client.

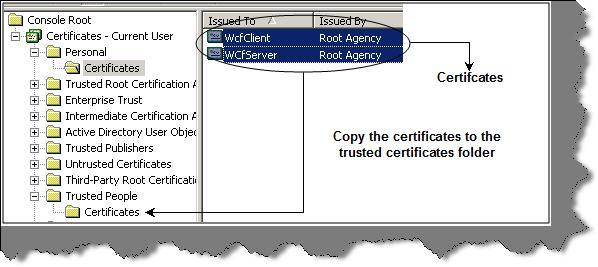


Step 2: Copy the certificates in trusted people certificates

Go to Start -> Run and type MMC and press Enter. You will be popped with the MMC console. Click on File -> Add/remove snap-in.

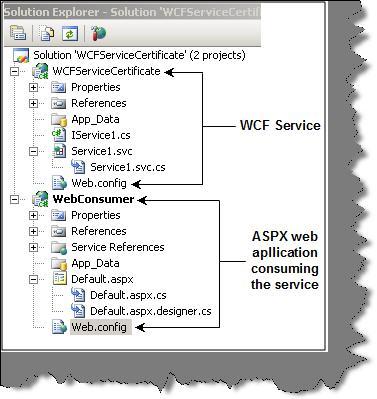


You will be popped up with an Add/Remove snap-in, click on the Add button, select Certificates, and select ‘My user Account’. You can see the certificates created for the client and server in the personal certificates folder. We need to copy those certificates in trusted people -> certificates folder.



Step 3: Specify the certification path and mode in the WCF service web.config file

Now that we have created both the certificates, we need to refer these certificates in our WCF project. So we have created two projects, one which has the WCF service and the other a web application which will consume the WCF service.



Let’s open the web.config file of the WCF service and enter two important things:

Where the certificate is stored, location, and how WCF application should find it. This is defined using the serviceCertificate tag as shown in the below snippet.

certificationvalidationmode defines how the client certificates will be authenticated.

|  |  |
| --- | --- |
| Certification validation mode | Description |
| Chain trust | In this situation, the client certificate is validated against the root certificate. |
| Peer trust | PeerTrust ensures that the public key portion of the certificate is in the Trusted People certificate folder on the client computer. |
| ChainORPeertrust | This is just an OR condition for both chain and peer. |

The above two points are clubbed together and entered in the web.config file of the WCF service.

Hide   Copy Code

<serviceCredentials>

<clientCertificate>

<authentication certificateValidationMode="PeerTrust"/>

</clientCertificate>

<serviceCertificate findValue="WCfServer"

storeLocation="CurrentUser"

storeName="My"

x509FindType="FindBySubjectName" />

</serviceCredentials>

Step 4: Define bindings

Now that we have defined our certificates and authentication type, we need to define that the authentication values will be sent through message using certificates. You can see we have defined WsHttpBinding with the message attribute specifying that the WCF client needs to send a certificate for validation.

Hide   Copy Code

<bindings>

<wsHttpBinding>

<binding name="wsHttpEndpointBinding">

<security>

<message clientCredentialType="Certificate" />

</security>

</binding>

</wsHttpBinding>

</bindings>

Step 5: Tie up the bindings with the end point

Once done, we need to tie up this binding with the end point. This is done by using the bindingConfiguration tag as shown in the below code snippet.

Hide   Copy Code

<endpoint address="" binding="wsHttpBinding" bindingConfiguration="wsHttpEndpointBinding"

contract="WCFServiceCertificate.IService1">

Step 6: Make your web application client for consuming the WCF service

That’s all we need from the WCF service perspective. So compile the WCF service and reference it in the ASP.NET web application using ‘Service Reference’. Below is the code snippet where we have referenced the service and called the ‘GetData’ function of the service.

Hide   Copy Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

>using WebConsumer.ServiceReference1;

namespace WebConsumer

{

public partial class \_Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

Service1Client obj = new Service1Client();

Response.Write(obj.GetData(12));

}

}

}

Now if you try to run the client, i.e., the web application as it is, you should get an error as shown below. The error clearly indicates you can use the WCF service until you do not provide the client certificate.



Step 7: Define certificates in the WCF client

So let’s start the process of defining certificates in the WCF client. The way we have defined the authentication certification mode and the path of the certificate, in the same way, we need to define it for the WCF client. You can see we have defined the authentication mode as ‘peertrust’ and we have specified the client certificate name as ‘WcfClient’.

Hide   Copy Code

<behaviors>

<endpointBehaviors>

<behavior name="CustomBehavior">

<clientCredentials>

<clientCertificate findValue="WcfClient" x509FindType="FindBySubjectName"

storeLocation="CurrentUser" storeName="My" />

<serviceCertificate>

<authentication certificateValidationMode="PeerTrust"/>

</serviceCertificate>

</clientCredentials>

</behavior>

</endpointBehaviors>

</behaviors>

Step 8: Tie up the behavior with the end point on the WCF client

We need to tie up the above defined behavior with the end point. You can see we have bound the behavior using the behaviorConfiguration property. We also need to specify that the DNS value will be WcfServer which is your server certificate name.

Hide   Copy Code

<client>

<endpoint address="http://localhost:1387/Service1.svc" binding="wsHttpBinding"

bindingConfiguration="WSHttpBinding\_IService1" contract="ServiceReference1.IService1"

name="WSHttpBinding\_IService1"

>behaviorConfiguration="CustomBehavior">

Step 9: Enjoy your hard work

Once we are done, you can run the ASP.NET web and you should see the below display.



What is the difference between BasicHttpBinding and WsHttpBinding?

If we want to summarize in one sentence the difference between ‘WsHttpBinding’ and ‘BasicHttpBinding’, ‘WsHttpBinding’ supports the ‘WS-\*’ specification. WS-\* specifications are nothing but standards to extend web service capabilities.

Below is a detailed comparison table between both the entities from security, compatibility, reliability, and SOAP version perspective.

| Criteria | BasicHttpBinding | WsHttpBinding |
| --- | --- | --- |
| Security support | This supports the old ASMX style, i.e., WS-BasicProfile 1.1. | This exposes web services using WS-\* specifications. |
| Compatibility | This is aimed for clients who do not have .NET 3.0 installed and it supports wider ranges of clients. Many clients like Windows 2000 still do not run .NET 3.0. So an older version of .NET can consume this service. | As it is built using WS-\* specifications, it does not support wider ranges of clients and it cannot be consumed by older .NET versions less than version 3. |
| SOAP version | SOAP 1.1 | SOAP 1.2 and WS-Addressing specification. |
| Reliable messaging | Not supported. In other words, if a client fires two or three calls, you really do not know they will return back in the same order. | Supported as it supports WS-\* specifications. |
| Default security options | By default, there is no security provided for messages when the client calls happen. In other words, data is sent as plain text. | As WsHttBinding supports WS-\*, it has WS-Security enabled by default. So the data is not sent as plain text. |
| Security options | None  Windows – default authentication  Basic  Certificate | None  Transport  Message  Transport with message credentials |

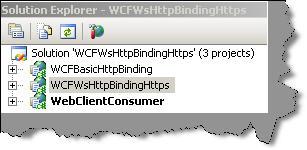
One of the biggest differences you must have noticed is the security aspect. By default, 'BasicHttpBinding' sends data in plain text while ‘WsHttpBinding’ sends in an encrypted and secured manner. To demonstrate this, let’s make two services, one using ‘BasicHttpBinding’ and the other using ‘WsHttpBinding’, and then let’s see the security aspect in a more detailed manner.

We will do a small sample to see how ‘BasicHttpBinding’ sends data in plain text format and how ‘WsHttpBinding’ encrypts data.

Note: By default, security is not enabled on ‘BasicHttpBinding’ for interoperability purposes. In other words it is like our old web service, i.e., ASMX. But that does not mean we cannot enable security in ‘BasicHttpBinding’. Sometime back I had written an article on how to enable security on ‘BasicHttpBinding’: [WCFBasicHttpBinding.aspx](http://www.codeproject.com/KB/WCF/WCFBasicHttpBinding.aspx).

Can you show the security differences between BasicHttpBinding and WsHttpBinding?

In order to understand the security differences between both these entities, we will do a small project. In this project, we will create two WCF services: one using ‘BasicHttpBinding’ and the second using ‘WsHttpBinding’.



Step 1: Let’s first create a simple service using ‘BasicHttpBinding’. For that we just create a simple WCF project and then modify the ‘ServiceModel’ element as shown below. You can see in the ‘endpoint’ tag, we have specified ‘basicHttpBinding’ as the protocol.

Hide   Shrink   Copy Code

<system.serviceModel>

<services>

<service name="WCFBasicHttpBinding.Service1"

behaviorConfiguration="WCFBasicHttpBinding.Service1Behavior">

<!-- Service Endpoints -->

<endpoint address="" binding="basicHttpBinding"

contract="WCFBasicHttpBinding.IService1">

<!--

Upon deployment, the following identity element should be removed or replaced to reflect the

identity under which the deployed service runs. If removed, WCF will infer an appropriate identity

automatically.

-->

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

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

contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="WCFBasicHttpBinding.Service1Behavior">

<!-- To avoid disclosing metadata information, set the value below to false

and remove the metadata endpoint above before deployment -->

<serviceMetadata httpGetEnabled="true"/>

<!-- To receive exception details in faults for debugging purposes,

set the value below to true. Set to false before deployment

to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="false"/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

Step 2: We need to create one more service using ‘WsHttpBinding’. For that you do not need to do anything special. By default, a WCF project is created using ‘WsHttpBinding’. Below is how the Web.config file looks like. You can see how the endpoint tag is using ‘wsHttpBinding’.

Hide   Shrink   Copy Code

<system.serviceModel>

<services>

<service name="WCFWsHttpBindingHttps.Service1"

behaviorConfiguration="WCFWsHttpBindingHttps.Service1Behavior">

<!-- Service Endpoints -->

<endpoint address="" binding="wsHttpBinding"

contract="WCFWsHttpBindingHttps.IService1">

<!--

Upon deployment, the following identity element should be removed or replaced to reflect the

identity under which the deployed service runs. If removed, WCF will infer an appropriate identity

automatically.

-->

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

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

contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="WCFWsHttpBindingHttps.Service1Behavior">

<!-- To avoid disclosing metadata information, set the value below to false

and remove the metadata endpoint above before deployment -->

<serviceMetadata httpGetEnabled="true"/>

<!-- To receive exception details in faults for debugging purposes, set the value below to true.

Set to false before deployment to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="false"/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

Step 3: We will not be creating any new methods in both the services. We will just use the default code created by the WCF template. Both these services will have a GetData function which returns a string. GetData is a default function created in WCF projects.

Hide   Copy Code

public class Service1 : IService1

{

public string GetData(int value)

{

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

}

public CompositeType GetDataUsingDataContract(CompositeType composite)

{

if (composite.BoolValue)

{

composite.StringValue += "Suffix";

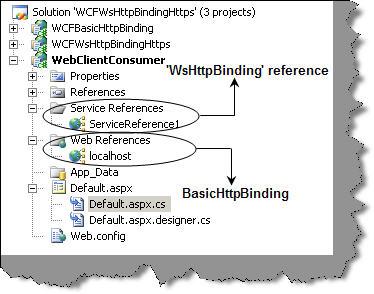
}

return composite;

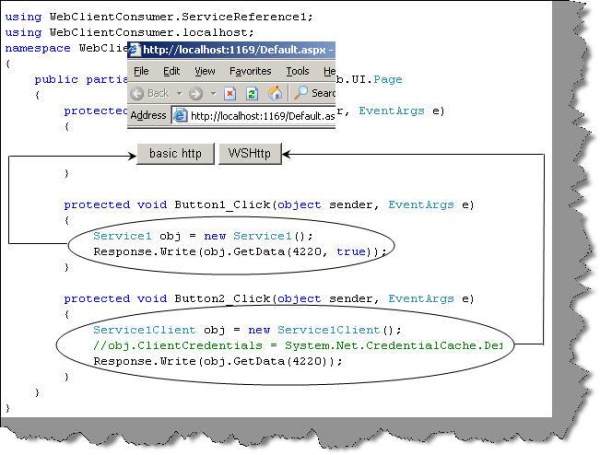
}

}

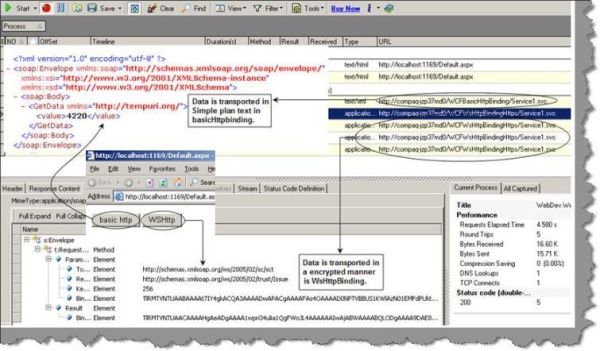
Step 4: Now that out services are created, we need to create a client which will consume this service. We create a simple web application and add two references: a service reference, i.e., ‘WsHttpBinding’, and a web reference, i.e., ‘BasicHttpBinding’. Please note, when you right click to add a reference, you need to use ‘Add service reference’ to add ‘WsHttpService’ and you need to add a web reference for ‘BasicHttpBinding’.



We will add two buttons on the default ASPX page. One button will call the HTTP service and the other will call the WsHTTP service. Below is how the GetData is called in the button clicks.



Step 5: Now that we are ready with the project, it is time to sniff and see how data is transported between the client and the service in both scenarios. Let’s download a simple HTTP data recorder from <http://www.ieinspector.com/httpanalyzer/download.html>. We will then click both the buttons one by one and record the data transfer using the tool. You can see the posted data is in simple plain XML format for the basic HTTP protocol and it’s in an encrypted format for the WsHTTP protocol.

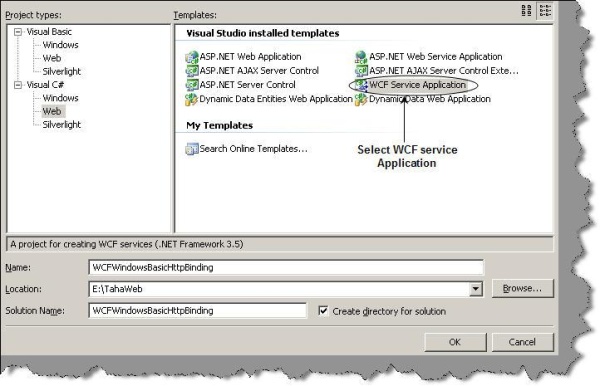


When should we use WsHttp as compared to BasicHttp?

If you are looking for backwards compatibility and to support lots of clients, then basic HTTP binding is the way to go, or else WsHttp is a great way to start if you are seeing your clients made in .NET 3.0 and above.

How can we enable Windows authentication on WCF using ‘BasicHttpBinding’?

Step 1: Create a WCF service application project as shown in the below figure.



Circle WCF service application -> Select this

By default, the WCF project creates a class file which has the GetData function. This function takes in number values and displays an explanatory sentence like ‘You entered 1 value’ when you enter ‘1’.

Hide   Copy Code

public class Service1 : IService1

{

public string GetData(int value)

{

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

}

}

Step 2: When we create a WCF service application, it also has a web.config file associated with it. So open the web.config file and ensure that the authentication mode is Windows.

Hide   Copy Code

<authentication mode="Windows" />

Step 3: The third step is to define the bindings and the transport type. To define the bindings, we need to enter a basicHttpBinding element inside the bindings XML tag. We also need to define clientCredentialType as Windows.

Hide   Copy Code

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpEndpointBinding">

<security mode="TransportCredentialOnly">

<transport clientCredentialType="Windows" />

</security>

</binding>

</basicHttpBinding>

</bindings>

<services>

.........

.........

</system.serviceModel>

Step 4: Now the bindings defined need to be associated with a service interface, ‘service1’. So we need to modify the services element as shown below. Note that we have defined an end point which has the binding association.

Hide   Copy Code

<system.serviceModel>

........

........

........

<services>

<service behaviorConfiguration="WCFWindowsBasicHttpBinding.Service1Behavior"

name="WCFWindowsBasicHttpBinding.Service1">

<endpoint address="" binding="basicHttpBinding"

bindingConfiguration="BasicHttpEndpointBinding"

name="BasicHttpEndpoint" contract="WCFWindowsBasicHttpBinding.IService1">

<identity>

<dns value="localhost" />

</identity>

</endpoint>

</service>

</services>

.........

.........

.........

.........

</system.serviceModel>

Your <system.serviceModel> XML with the bindings and services is shown below:

Hide   Shrink   Copy Code

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpEndpointBinding">

<security mode="TransportCredentialOnly">

<transport clientCredentialType="Windows" />

</security>

</binding>

</basicHttpBinding>

</bindings>

<services>

<service behaviorConfiguration="WCFWindowsBasicHttpBinding.Service1Behavior"

name="WCFWindowsBasicHttpBinding.Service1">

<endpoint address="" binding="basicHttpBinding"

bindingConfiguration="BasicHttpEndpointBinding"

name="BasicHttpEndpoint" contract="WCFWindowsBasicHttpBinding.IService1">

<identity>

<dns value="localhost" />

</identity>

</endpoint>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="WCFWindowsBasicHttpBinding.Service1Behavior">

<!-- To avoid disclosing metadata information, set the value below to false and

remove the metadata endpoint above before deployment -->

<serviceMetadata httpGetEnabled="true"/>

<!-- To receive exception details in faults for debugging purposes, set the value below

to true. Set to false before deployment to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="false"/>

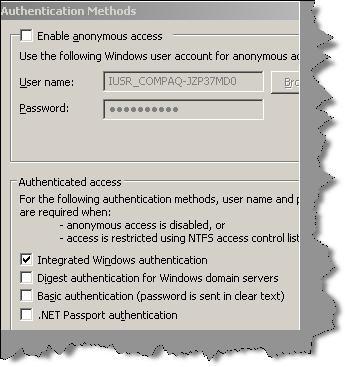
</behavior>

</serviceBehaviors>

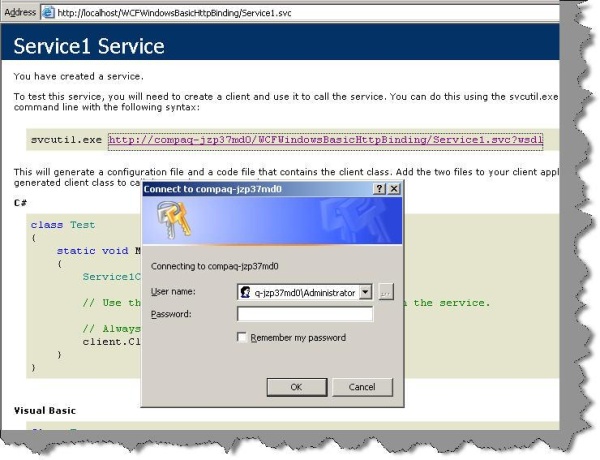
</behaviors>

</system.serviceModel>

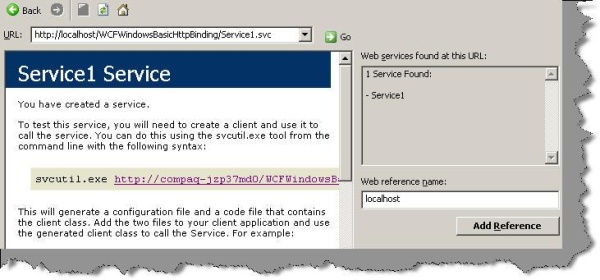
Step 5: Go to IIS properties and click on Security tab and ensure that anonymous access is disabled and only Windows authentication is enabled.



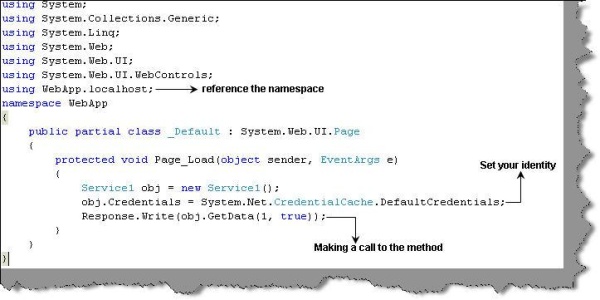
Step 6: We need to host our service in IIS. So make the directory as an IIS application so that your service can be hosted. Now if you try to browse the service, i.e., the SVC file, you will see that it pops up the authentication authorization security dialog box. So this service cannot be executed with Windows authentication.



Step 7: Let’s consume this WCF service. Add an ASP.NET web application and add a web reference. You will be shown a dialog box as shown below. Click on Add Reference so that a proxy is generated for the WCF service.



Step 8: Type in the following code snippet in your page load. Add the namespace reference and call the method GetData. The most important step to note is the credential supplied. DefaultCredentials passes the current Windows identity to the WCF service.



If you execute the service, you should get the following display:



You can try commenting the below code in your client, in other words we are not passing any credentials.

Hide   Copy Code

obj.Credentials = System.Net.CredentialCache.DefaultCredentials;

Now if you execute, you should get the below error stating that this is an unauthorized call.



License