A complete guide to create secure WCF REST API with custom Basic Authentication

WCF REST API services are still being used by many developers for client server connectivity for data and messaging. This blog is a complete guide on creating a WCF Rest service from scratch and Adding security to the service using Basic Authentication. Then we'll learn how to encrypt the basic authentication information which would be sent over the network using SSL. The main sections of this guide are:

- Creating a WCF REST API service
- Hosting a WCF REST service in IIS from Visual studio (on local machine)
- Deploying a WCF REST service on IIS (Local machine)
- Adding security to the Service by using Basic Authentication
- Securing basic authentication credentials using SSL over Http i.e. (Https)
    - Creating a certificate and Enabling IIS website to use Https
    - Setting up WCF REST service to use SSL (Https)

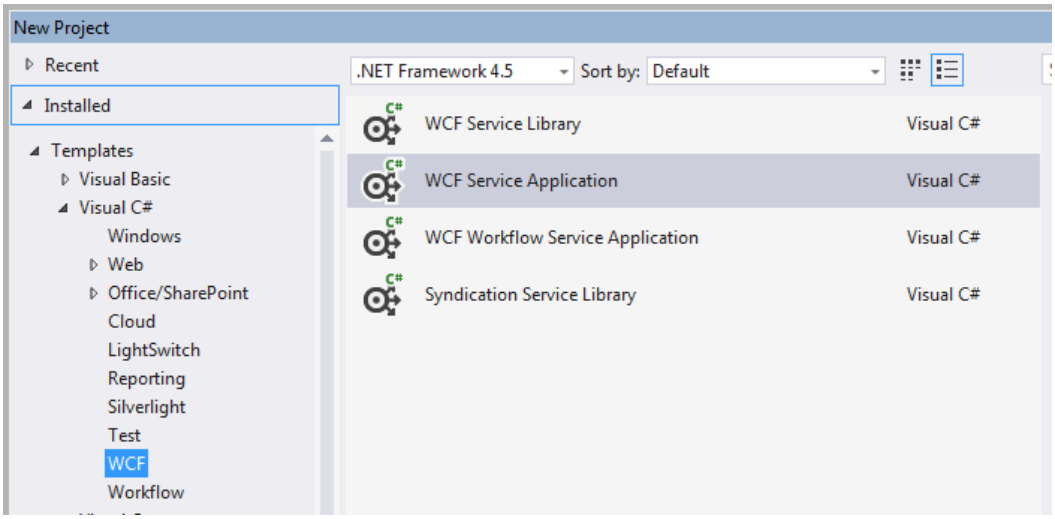
Pre-requisite:

- Basic knowledge of Visual studio/WCF basics.
- Visual studio version > 2008
- .Net framework > 3.5 installed

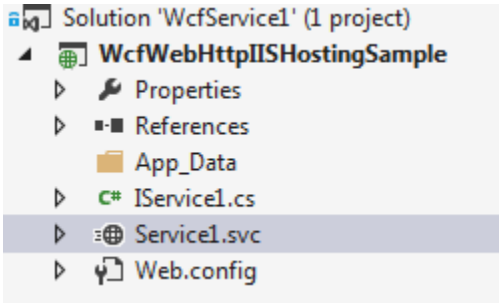
Creating a WCF REST API service

To get started quickly we'll be using the default template of WCF service library provided in Visual studio 2013 which I'm going to use in this guide. Follow the steps below:

1. Launch visual studio 2013(choose **"Run as Administrator"**, we'll see later why?)
2. From Menu **File** -> **New** -> **Project.** or click on Start Page to **start** a **New Project.**



3. Let's name it as `WcfWebHttpIISHostingSample` Now you'll see couple of files already added to the Wcf Service project.



4. Delete **IService1.cs** and **Service1.svc** file as we'll be creating new files and use our code to host the service via **ServiceHostFactory** class.
5. Add a new interface ITestService by right click on project and Add new item -> select Interface and rename it to **ITestService**. Copy below code and add it in newly created interface.

```
namespace WcfWebHttpIISHostingSample
{
    [ServiceContract]
    public interface ITestService
    {
        [WebInvoke(Method = "GET", UriTemplate = "/Data/{data}")]
        string GetData(string data);
    }
}
```

   In above interface we have added **WebInvoke** attribute though we can also use **WebGet** it would not make any difference. I'll stick with **WebInvoke** as it support POST, PUT, DELETE http verbs as well.
6. Add a new class TestService which will implement the above declared interface.

```
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Web;

namespace WcfWebHttpIISHostingSample
{
    [ServiceContract]
    public interface ITestService
    {
        [WebInvoke(Method = "GET", UriTemplate = "/Data/{data}")]
        string GetData(string data);
    }
}
```

7. We have defined a service contract, a Rest method with a sample definition. Now we have to define its end points. To add end point simply copy below settings and paste it into your configuration file(web.config) of newly created project file under service.model tag.

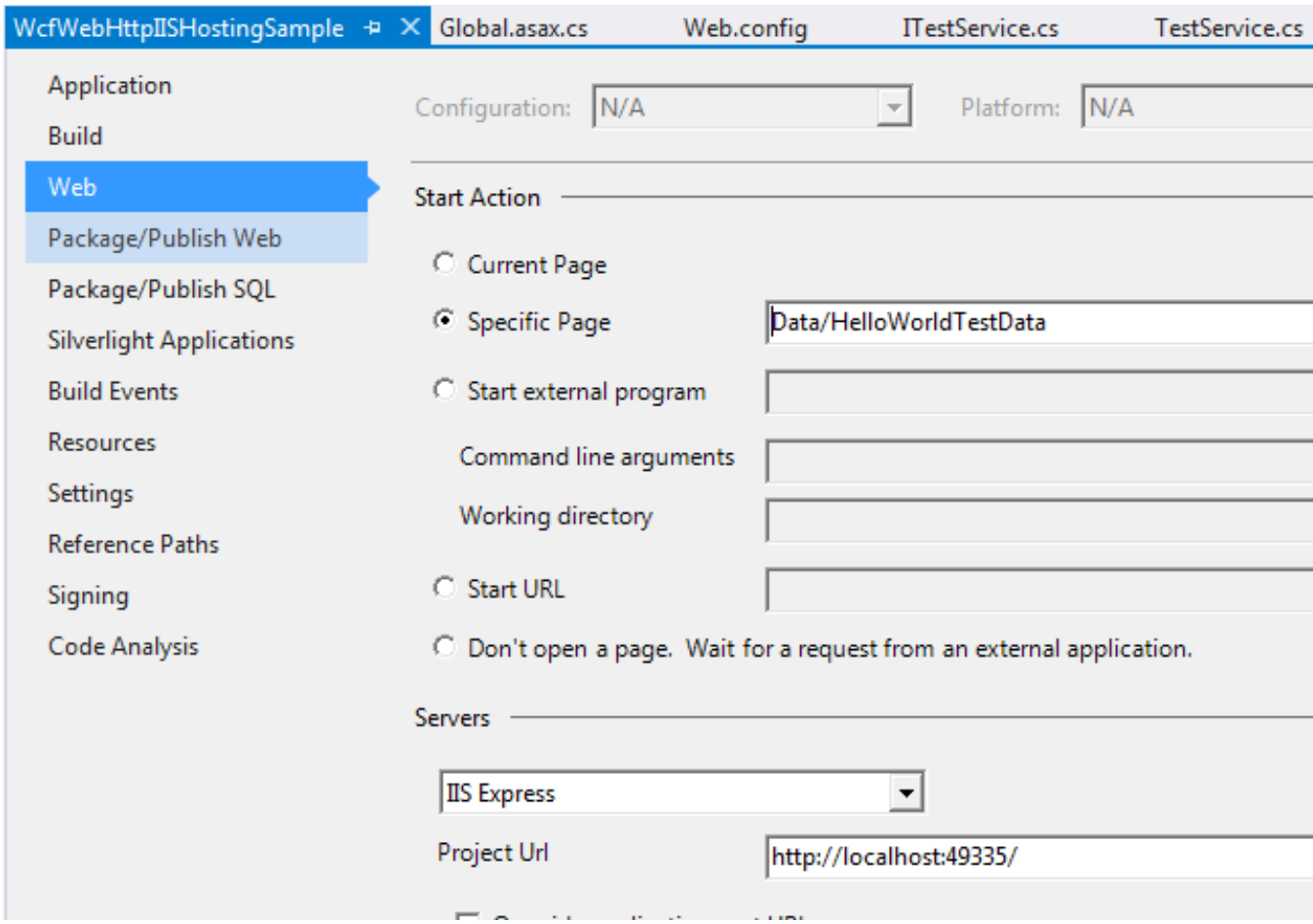
```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="ServiceBehavior">
        <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
        <serviceDebug includeExceptionDetailInFaults="false"/>
      </behavior>
```

```
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="webHttpServiceBehavior">
          <!-- Important this is the behavior that makes a normal WCF service to REST
based service-->
          <webHttp/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <services>
      <service name="WcfWebHttpIISHostingSample.TestService"
behaviorConfiguration="ServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost/WCFRestAuthentication/api/"/>
          </baseAddresses>
        </host>
        <endpoint binding="webHttpBinding"
contract="WcfWebHttpIISHostingSample.ITestService"
behaviorConfiguration="webHttpServiceBehavior"/>
      </service>
    </services>
</system.serviceModel>
```

8. Now we have an end point. So Next we'll add a service host factory that will host the service in IISExpress or Local development server at this moment.
9. Add new item Global.asax and add following code in Application_Start method. You can find it Under New Item -> Web -> Global application Handler.

```
        protected void Application_Start(object sender, EventArgs e)
        {
            RouteTable.Routes.Add(new ServiceRoute("", new WebServiceHostFactory(),
typeof(TestService)));
         }
```

10. Now you'll be getting an error on **ServiceRoute** class in above code. To remove this error we have to add a new reference from Add reference -> Assemblies -> **System.Model.Activation**. And the error will be gone.
11. Now right click on Project and go to Properties. In properties window select Web and Add specific page details to map the service operation that we defined on our contract. Add Data/HelloWorldTestData in Specific page setting. Here Data is your path of operation that we have defined in WebInvoke attribute and "HelloWorldTestData" is your argument that the service method will receive as argument. See below:



12. Save All and Press F5.

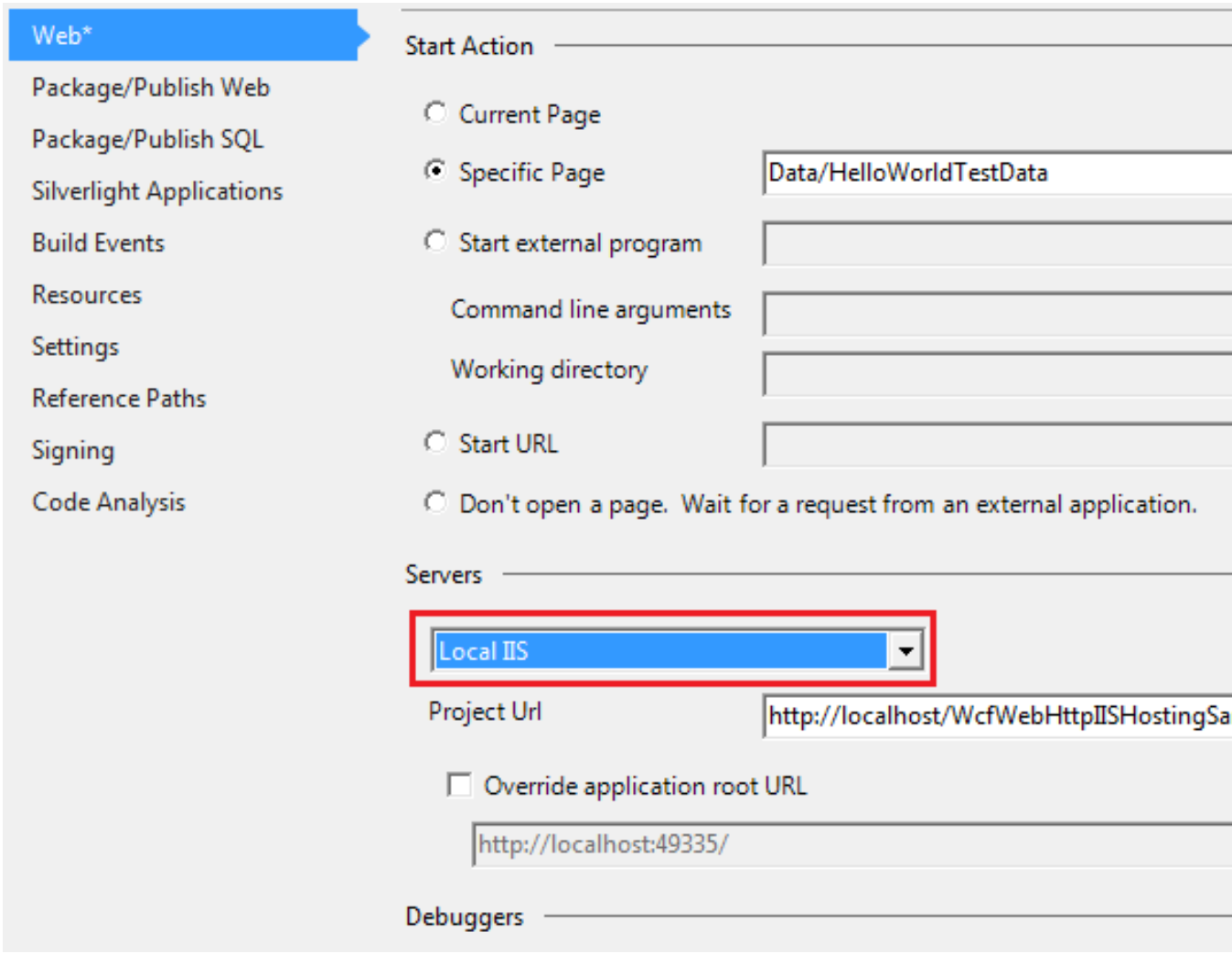
If you see a web page like this means you have successfully created a WCF REST service.

WCF REST service hosting in IIS

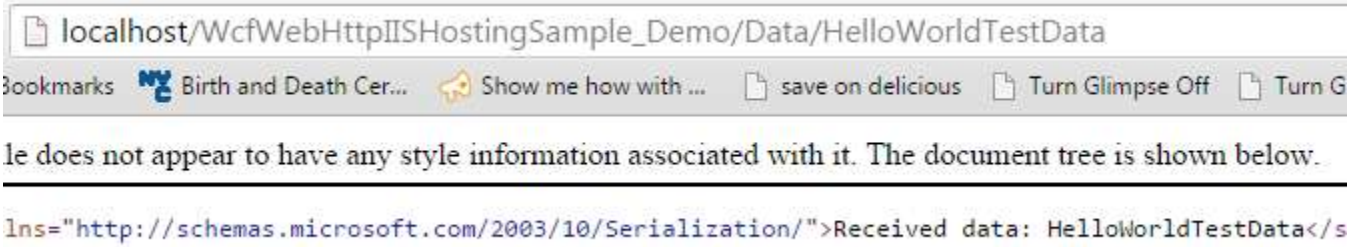
To Host the above service in IIS follow below steps:

1. Right click on the project and go to **Properties**. In properties window select **Web**.

2. Now under Servers on Web settings you'll see following details change the "IIS Express" to "IIS Server".



3. Now Click on **Create Virtual Directory.** Now if you have Visual Studio running As Administrator then you'll get a message **The Virtual directory was created successfully!** Otherwise you'll receive an error message and you need to launch again Visual studio as Administrator. I mentioned it in the start of this Guide.
4. Now press F5 and your website will be up and running on IIS server instead of your IIS express.



Deploying a WCF REST service on IIS Local machine (optional)

(Optional) You don't actually need to deploy the current WCF service because the visual studio already using IIS server for hosting as detailed in above step. This is just to let you know how to deploy a WCF REST service on IIS.

Follow below steps:

1. Right click on **Project** -> Goto **Publish**.
2. In Publish windows **open** the dropdown for profiles and select **New Profile**.
3. Select publish method as "Web Deploy". Enter server as "Localhost". Add site name as "**Default Web site/WcfWebHttpIISHostingSample_Demo2**". Click Validate connection it'll verify access rights and other information provided.
4. Once the Validate connection succeeds click publish. You'll get the success message in output window in Visual studio.
5. Test the service by entering in browser following url - http://localhost/WcfWebHttpIISHostingSample_Demo2/Data/TestingDeploymentSuccess

Adding Basic authentication

The REST services built and run over http protocol. A simple way to enable the authentication is to use **Basic Authentication**(i.e. user/pwd). Basic authentication can enabled over http protocol. Now here are the choices that we have:

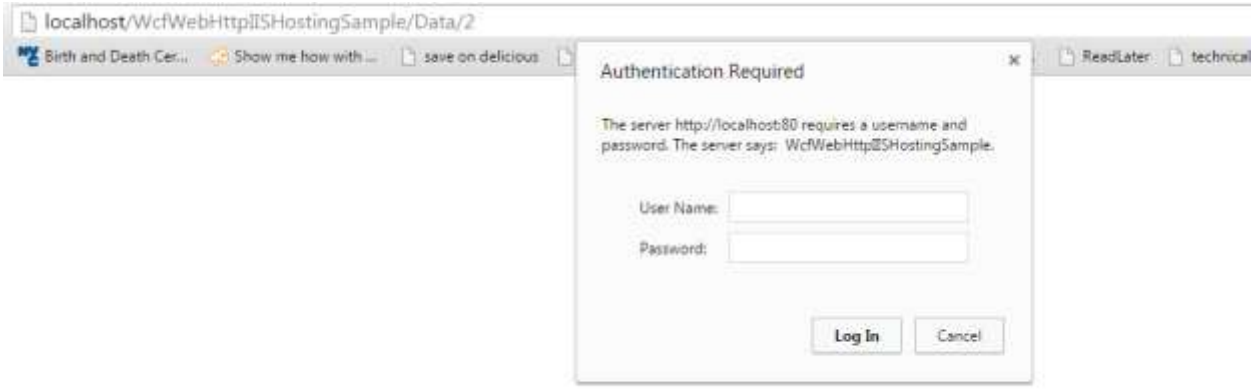
We can use simple transport level basic authentication by using Custom username and password **validator** as mentioned here. But author explicitly stated that *this will only work in **self hosted** environment*. Which is not our case as we are using IIS to host our service. Also there are other method available on the internet like by extending **ServiceAuthenticationManager** or using a custom **MessageInspector** and use it like a custom user/pwd validator but I was not really convinced with those methods because somehow they were not best candidate for our scenario. The challenge comes with IIS is that ***IIS does the authentication before WCF receives the request***. And Till now there's no out of the box support for Basic Authentication for WCF REST on IIS. So we have to go with our custom solution which by extending **ServiceAuthorizationManager** class and override method **CheckAccessCore** and use it in Service Behaviors as default Authorization manager. The configuration would look something like below after setting up a custom service Authorization manager.

```
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="true"/>
          <serviceAuthorization
            serviceAuthorizationManagerType
              =" WcfWebHttpIISHostingSample.RestAuthorizationManager,
WcfWebHttpIISHostingSample"/>
        </behavior>
      </serviceBehaviors>
```

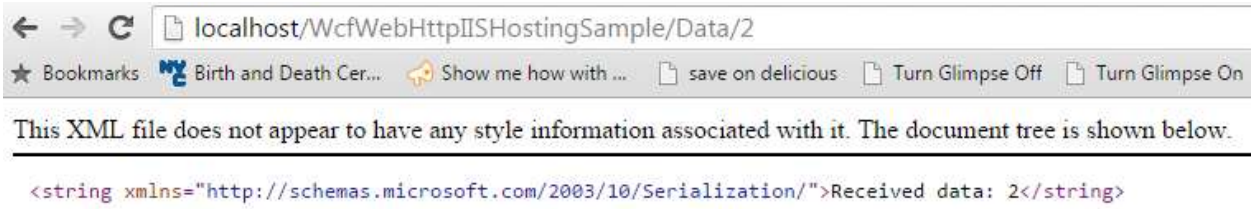
Now I have to extend the class **ServiceAuthorizationManager** with a custom class **RestAuthorizationManager** which I named it to make sense with the context.

```
   public class RestAuthorizationManager : ServiceAuthorizationManager
    {
        /// <summary>
        /// Method source sample taken from here: http://bit.ly/1hUa1LR
        /// </summary>

        protected override bool CheckAccessCore(OperationContext operationContext)
        {
            //Extract the Authorization header, and parse out the credentials converting
the Base64 string:
            var authHeader =
WebOperationContext.Current.IncomingRequest.Headers["Authorization"];
            if ((authHeader != null) && (authHeader != string.Empty))
            {
                var svcCredentials = System.Text.ASCIIEncoding.ASCII
                        .GetString(Convert.FromBase64String(authHeader.Substring(6)))
                        .Split(':');
                var user = new { Name = svcCredentials[0], Password = svcCredentials[1]
};
                if ((user.Name == "testuser" && user.Password == "testpassword"))
                {
                    //User is authrized and originating call will proceed
                    return true;
                }
                else
                {
                    //not authorized
                    return false;
                }
            }
            else
            {
                //No authorization header was provided, so challenge the client to
provide before proceeding:
                WebOperationContext.Current.OutgoingResponse.Headers.Add("WWW-
Authenticate: Basic realm=\"MyWCFService\"");
                //Throw an exception with the associated HTTP status code equivalent to
HTTP status 401
                throw new WebFaultException(HttpStatusCode.Unauthorized);
            }
        }

    }
```

Now your application is ready to challenge any request with Basic Authentication. To test it you can use any client like Fiddler or Chrome's post man plugin and see what outcome you get. Let's simply make a request from normal browser I'll get a challenge for basic authentication prompting user/password.

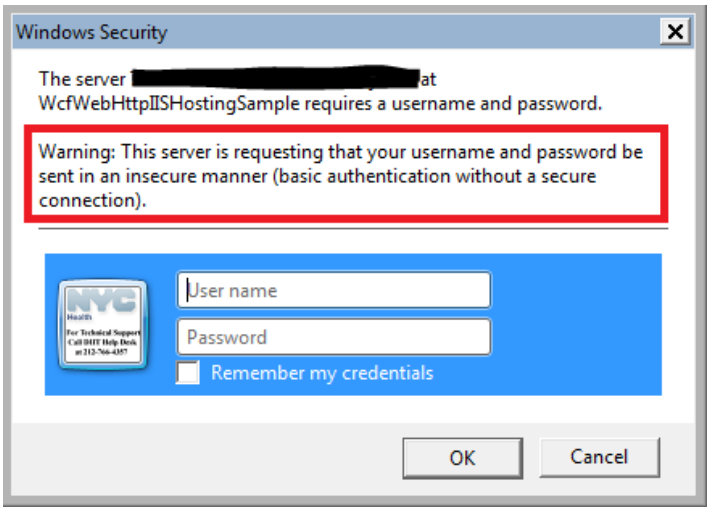


Enter the username - **testuser** and password – **testpassword** which we have used to validate the values in our **RestAuthorizationManager** class.



Securing basic authentication credentials using SSL over Http i.e. (Https)

The Basic authentication information usually sent in plain text (encrypted by Base64 string which can be easily decrypted) over the network. Most of the browser issues a warning about this secure information being sent as plain text. If you try to launch the API call in IE browser you'll see a warning something like below:



Here comes the Transport layer security into picture. All communication being done over the network can be secured using the Transport layer security by using SSL(Secure Sockets Layer) certificates.
To apply SSL security first you need a certificate. In order to get a real certificate one can go to certificates providers like Thawte, digicert, Godaddy etc. and purchase a certificate. These providers (not

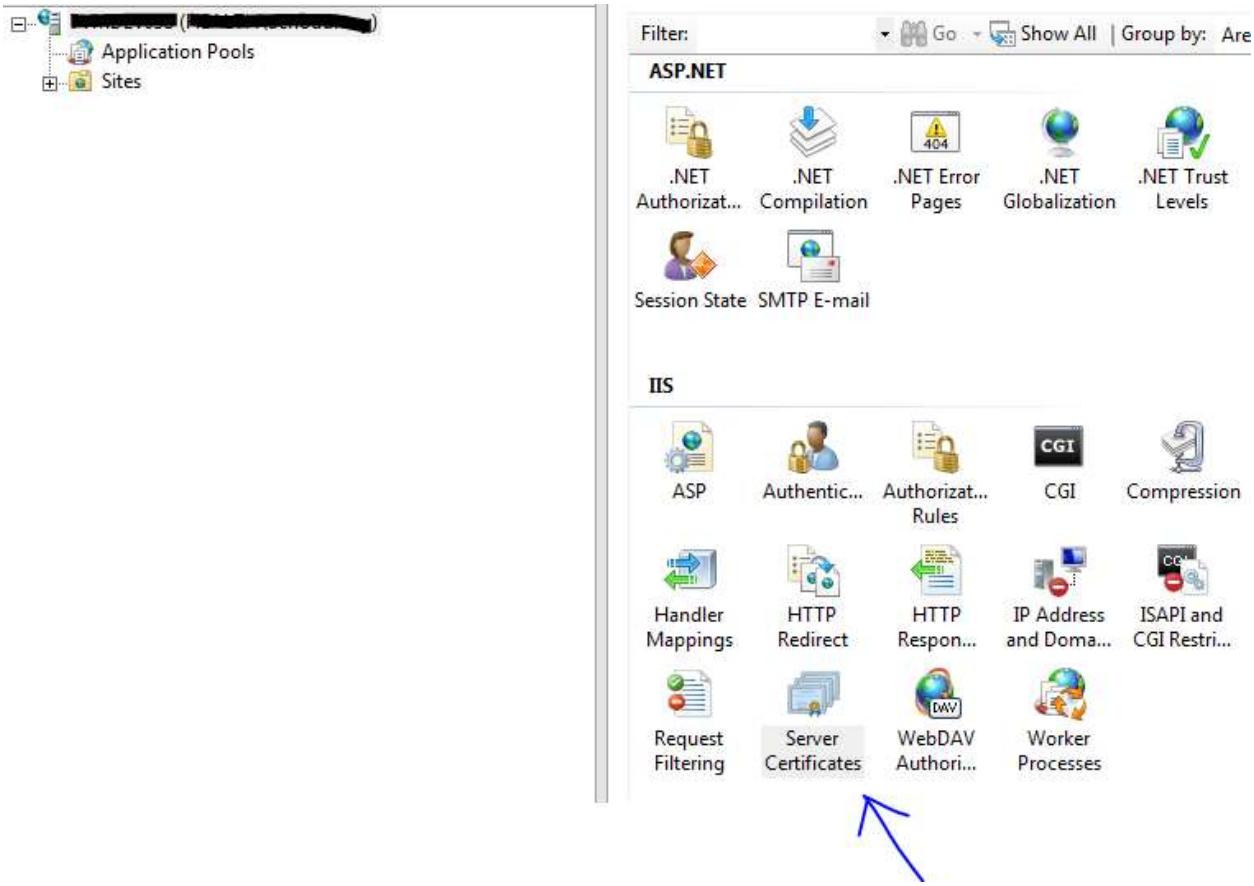
mentioning any specific provider but all in general) are trusted providers for issuing digital certificates to ensure that identity of certificate owner is verified.

But in development environment you don't need to purchase a SSL certificate in real. You can generate a local self-signed certificate from your machine. But the certificate would be only valid for your local use only. If you use the certificate over the network this would be invalidated. I'm not going into details so let's get started how to get a self-signed server certificate that you can use in development environment.

Creating a certificate and enabling the Https settings to use the certificate

You have two options either create a certificate using '[makecert]' command. [Here's] a nice detailed article with quite good explanation about certificates. Or you can use IIS 7.0 or greater to create your certificate.

1. Launch Internet Manager from control panel or goto Run -> type **'Inetmgr'** -> press enter.
2. Select the ROOT node on left side panel. Look for **`Server Certificates`** under **`IIS`** on right panel.

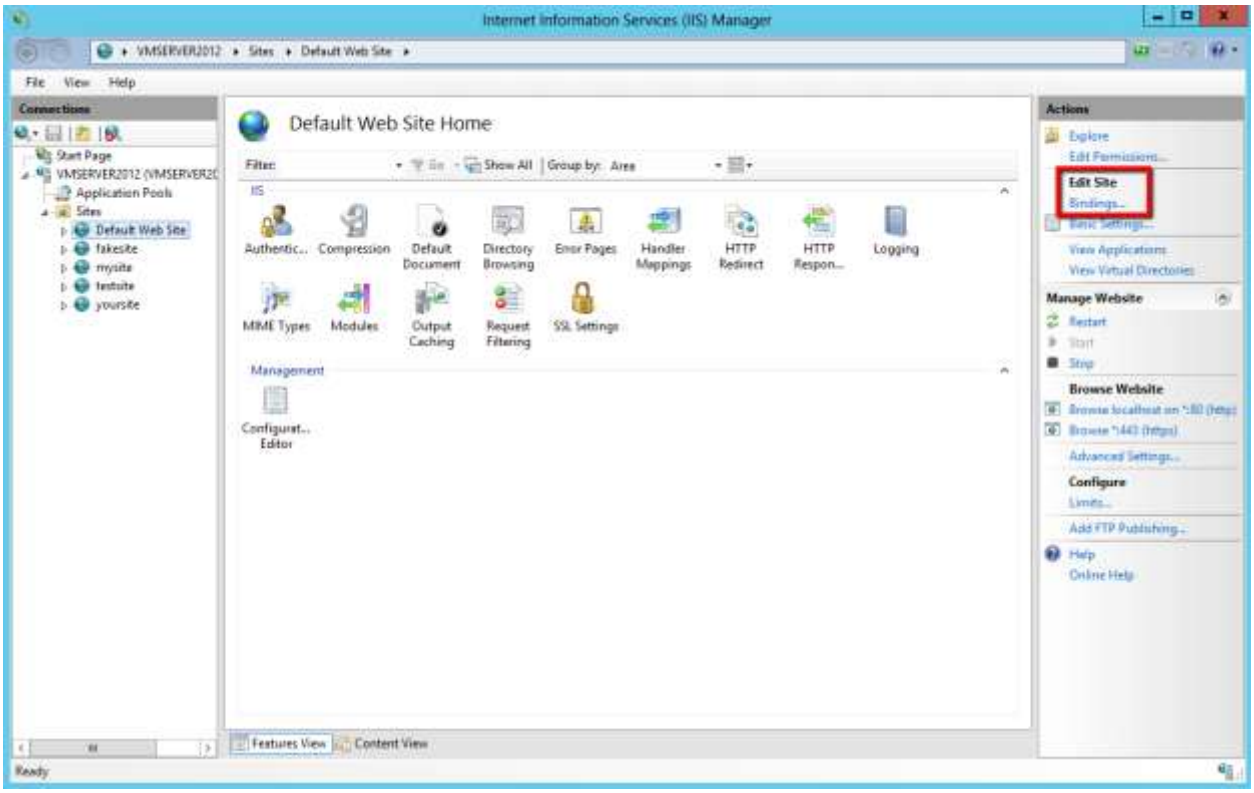


3. Double click on '**Server Certificates'** and select **'Create self-signed certificate'**.

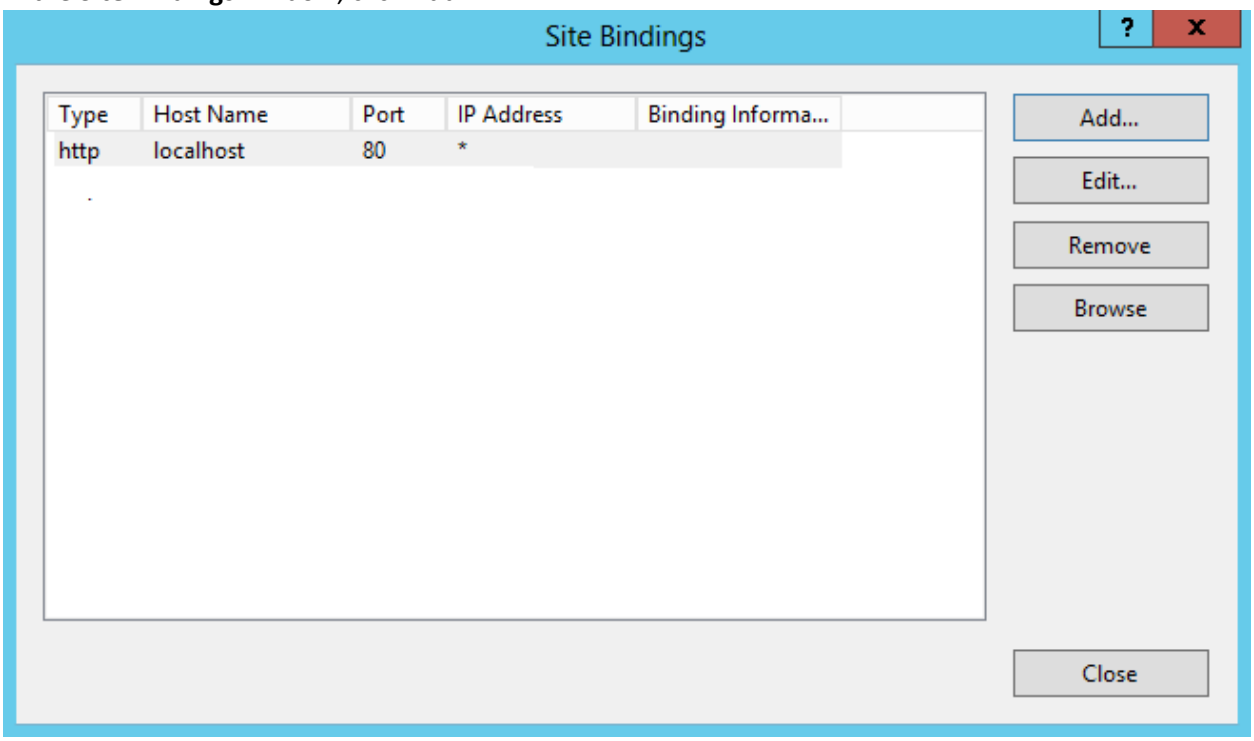
4. Now enter the name of certificate. This will be the friendly name of your certificate.

5. Click Ok and certificate that you have created would appear in parent window.

6. Now **select** the **website** you want to configure for HTTPS in IIS.



7. In the **Site Bindings** window, click **Add**.



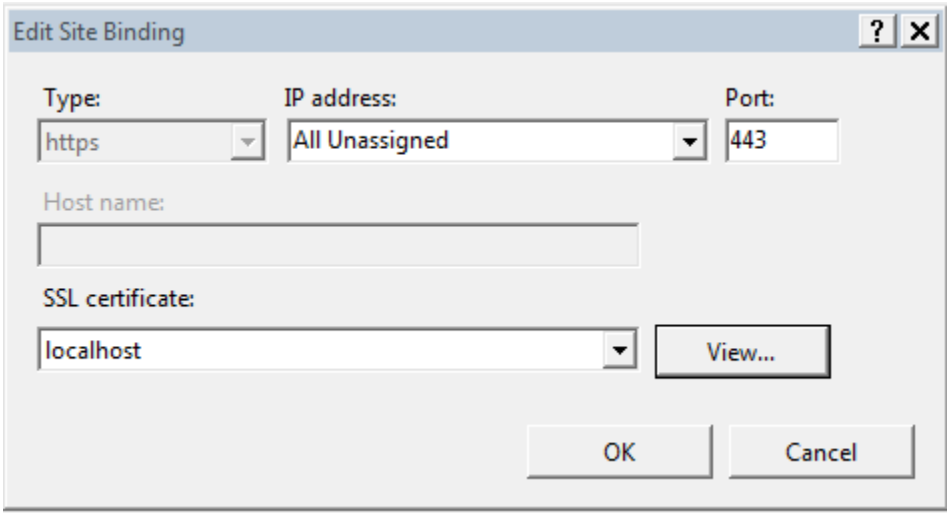
8. In the **Add Site Binding** window, enter the following information:

Type: In the drop-down list, select **https**.

IP address: In the drop-down list, select All Unassigned.

Port: Enter 443. The port for SSL traffic is usually port 443.

SSL certificate: In the drop-down list, select your recently imported SSL Certificate by its friendly name.



9. Click OK. Your SSL Certificate is now installed and the website is configured to accept secure connections.

Note: You may have to restart IIS or the server for it to recognize the new certificate (in some cases).

10. To test the setting open the site binding again and select Https binding and click on **'Browser'**. You might see below error.

11. To get rid of this error use the machine name exactly same as your certificate section "Issued to" says. E.g. If you open your certificate then you'll see **issued to** property and which should be your Machine name. If your machine is part of a domain then machine name would be like <machinename>.<xyz>.<domain> etc. so if you open it in your browser will fully qualified name of your machine then you won't be getting that error.

Setting up WCF REST service to use SSL (Https)

To enable the SSL over Http protocol follow below steps:

1. Add Protocol mapping for the webhttpbinding to use https.

```
<protocolMapping>
       <add binding="webHttpBinding" scheme="https"/>
</protocolMapping>
```

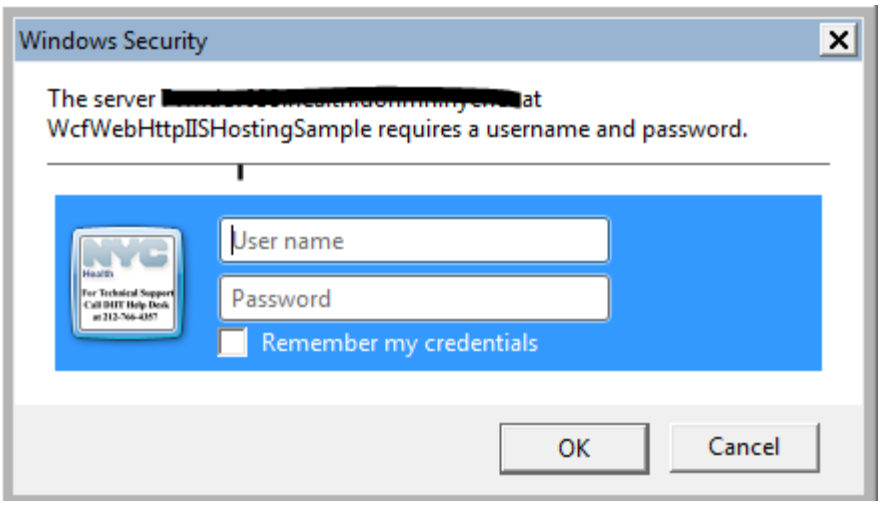
2. Under **binding** -> **webhttpbindings** enable security mode **Transport**.

```
<bindings>
      <webHttpBinding>
         <security mode="Transport" />
        </binding>
      </webHttpBinding>
 </bindings>
```

3. Add a host base address under services -> service.

```
<service name="WcfWebHttpIISHostingSample.TestService"
behaviorConfiguration="ServiceBehavior">
     <host>
      <baseAddresses>
        <add baseAddress="https://desktop-pc.mydomain/WcfWebHttpIISHostingSample/api/"/>
      </baseAddresses>
     </host>
       …
       …

</services>
```

4. Now rebuild the service and test it in the browser. You will not see the warning of in-secure server communication any more.



I hope this article will helped you. Feel free to like share. The complete sample used in this Guide can be found here on Github.