

5 20 54.9k

[EntArch.zip](#)[Download 100% FREE Spire Office APIs](#)

In this article I am going to show how to use Authorization and Authentication using a WCF service in Enterprise Architecting standards. This article is of Advanced WCF concepts. I am using an error driven approach for better experience with the problems and the solutions.

The core aspects we cover here are:

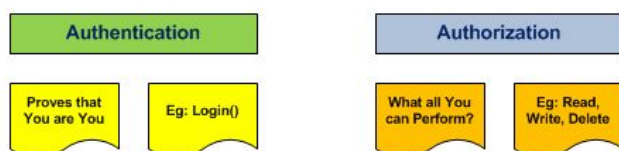
- WCF
- ASP.NET Authentication Service
- Custom Authentication
- HTTP Cookies
- Authorization PrincipalPermission Attribute
- Thread CurrentPrincipal
- Message Interceptors

You will be wondering what the above are. In a quick summary the following are the activities involved.

1. Create a WCF Service Application
2. Add an AuthenticationService.svc reusing ASP.NET Authentication Service
3. Create User Validator class
4. Enable Custom Authentication in Global.asax
5. Return Cookie if valid User
6. Modify service configuration
7. Try accessing the Authentication Service in Browser
8. Create a UtilityService.svc with one method named GetData(int)
9. Decorate the GetData(int) with PrincipalPermission attribute for Authorized Access only
10. Decorate UtilityService class withAspNetCompatibilityRequirements attribute
11. Set Utility Service constructor to set CurrentPrincipal from cookie
12. Create the client application and add references to both services
13. Create Authentication Service instance and invoke the Login() method
14. Receive the cookie and store it
15. Create a UtilityService instance and invoke GetData()
16. Attach the Cookie to the UtilityService client
17. Test the application and ensure the proper functioning
18. Move the cookie attaching code to Interceptors in the Client Application
19. Move the identity setting code to Interceptors in the Service Application
20. Modify the service-side code to include Role instead of Name
21. Use Encrypted Ticket for storing User Name and Roles
22. Retest the Application

## Authentication and Authorization

Before starting, we should have a clear understanding of Authentication and Authorization:



So to simplify, Authentication proves who the user is using his/her credentials or certificates.

Authorization deals with the rights the user has. For example, the Admin user will have Read, Write, Delete privileges but an ordinary Clerk will have Read permissions.

In WCF we are using the Membership infrastructure of ASP.NET to implement Authentication and Authorization.

Now we can start with the steps above.

**Step 2:** Add an AuthenticationService.svc reusing ASP.NET Authentication Service.

Add a new WCF Service and name it AuthenticationService.svc. Delete the associated files since we are going to expose the ASP.NET Authentication Service.

1. Delete AuthenticationService.cs
2. Delete IAuthenticationService.cs

Replace the contents of AuthenticationService.svc with the following:

```
<%@ ServiceHost
    Language="C#"
    Service="System.Web.ApplicationServices.AuthenticationService"
    Factory="System.Web.ApplicationServices.ApplicationServicesHostFactory" %>
```

Here we are exposing System.Web.ApplicationServices.AuthenticationService through the svc file. The associated namespace is referred to our project by default.

The Authentication Service supports the following methods:

- Login()
- Logout()
- ValidateUser()

Each of these methods work with the Membership Provider to perform the functionalities. We can also customize these aspects by providing our own database of user credentials and controlling the cookie creations.

**Step 3:** Create User Validator class

The Authentication Service will be receiving User Name and Password. We need to validate this with a Custom Provider. (We are not using ASP.NET Membership provider here.)

The following is the class definition of it. For the time being, we are hardcoding a user name as tom and password as chicago12.

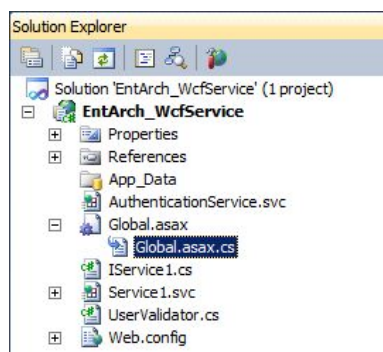
```
public class UserValidator
{
    public bool IsUserValid(string userName, string password)
    {
        bool result = (userName == "tom") && (password == "chicago12");

        return result;
    }
}
```

(Later in real projects you can change this single class to connect to a database to validate the user name and password)

**Step 4:** Enable Custom Authentication in Global.asax

Add a new item Web > Global Application Class into the project.



Replace the Application\_Start event as follows:

```
protected void Application_Start(object sender, EventArgs e)
{
```



And the following event handler:

```
void AuthenticationService_Authenticating(object sender, System.Web.ApplicationServices.AuthenticatingEventArgs e)
{
    e.Authenticated = new UserValidator().IsValid(e.UserName, e.Password);

    e.AuthenticationIsComplete = true;
}
```

The above method extracts the User Name and Password from the Custom Credential object of the Authentication Event Argument. Then it validates the User Name and Password with our UserValidator class.

The property Authenticated represents true / false if a user is valid or not respectively.

**Step 5:** Return a Cookie if valid user.

Now we need to send back a Cookie if the user is valid. For this add the following code in the above method:

```
if (e.Authenticated)
{
    // Create Cookie
    HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName);
    cookie.Value = e.UserName;

    // Attach Cookie to Operation Context header
    HttpResponseMessageProperty response = new HttpResponseMessageProperty();
    response.Headers[HttpRequestHeader.SetCookie] = cookie.Name + "=" + cookie.Value;
    OperationContext.Current.OutgoingMessageProperties
        [HttpRequestHeader.SetCookie] = response;
}
```

The above code performs the following:

1. Create a cookie with default name (.ASPXAUTH)
2. Set value to the User Name
3. Add the cookie to the WCF Operation Context message property

**Step 6:** Modify the service configuration.

Now we need to modify the web.config file to include the following:

1. Enable Authentication Service
2. Enable ASP.NET Compatibility

Replace the contents of web.config with the following:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.web.extensions>
    <scripting>
      <webServices>
        <authenticationService enabled="true"/>
      </webServices>
    </scripting>
  </system.web.extensions>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
  </system.webServer>
</configuration>
```

**Step 7:** Try accessing the Authentication Service in the Browser.

CUS

C# Corner Annual Conference 2017 Recap

following page then you are on the right track. (Otherwise, please check the steps above for any corrections / use the attached service code.)



ASK A QUESTION

CONTRIBUTE

## AuthenticationService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil

```
svcutil.exe http://localhost:54332/AuthenticationService.svc?wsdl
```

**Step 8, 9, 10:** Create a UtilityService.svc with one method named GetData(int).

Add a new WCF service named UtilityService.svc into our service application. Delete the existing methods inside the resulting classes and interfaces.

Add the following contents to the interface and class respectively.

```
// IUtilityService.cs
[ServiceContract]
public interface IUtilityService
{
    [OperationContract]
    string GetData(int i);
}

// UtilityService.svc
using System.ServiceModel;
using System.Security.Permissions;
using System.ServiceModel.Activation;

namespace EntArch_WcfService
{
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
    public class UtilityService : IUtilityService
    {
        [PrincipalPermission(SecurityAction.Demand, Name = "tom")]
        public string GetData(int i)
        {
            string result = "You Entered: " + i.ToString();
            return result;
        }
    }
}
```

The [PrincipalPermission] ensures that only user with name tom is authorized to access the method GetData(). In the real world applications we will be using the Role property to authorize a group of users.

The [AspNetCompatibilityRequirements] ensures that the service can be run in ASP.NET compatibility mode as we are using this mode for our Authentication infrastructure needs.

Authorization can be applied in 2 ways:

- Imperative: Example: if (Roles.IsUserInRole())
- Declarative: PrincipalPermission

**Step 11:** Set the Utility Service constructor to set the CurrentPrincipal from the cookie.

The WCF Authentication Service had sent a cookie to the user. This cookie will be returned back to the Utility Service (and any new services in the future) by the client. We need to retrieve the cookie and extract the user name out of it. This user name has to be set to the current thread principal. In real projects, you need to set the Roles as well.

The following are the activities involved in this step:

1. Retrieve Cookie from Operation Context Income Message Properties
2. Extract the User Name from the Cookie
3. Create a class CustomIdentity implementing IIdentity
4. Set the Thread Current Principal to the CustomIdentity instance

Modify the constructor of UtilityService as following:

```
public UtilityService()
{
```

CUS

C# Corner Annual Conference 2017 Recap



```
OperationContext.Current.IncomingMessageProperties[HttpRequestMessageProperty.Name]
string cookie = messageProperty.Headers.Get("Set-Cookie");
```

ASK A QUESTION

CONTRIBUTE

```
// Set User Name from cookie
if (nameValue.Length >= 2)
    userName = nameValue[1];

// Set Thread Principal to User Name
CustomIdentity customIdentity = new CustomIdentity();
GenericPrincipal threadCurrentPrincipal = new GenericPrincipal(customIdentity, new string[] { });
customIdentity.IsAuthenticated = true;
customIdentity.Name = userName;
System.Threading.Thread.CurrentPrincipal = threadCurrentPrincipal;
}
```

Following is the class definition for CustomIdentity. (You need to place it inside the service project.)

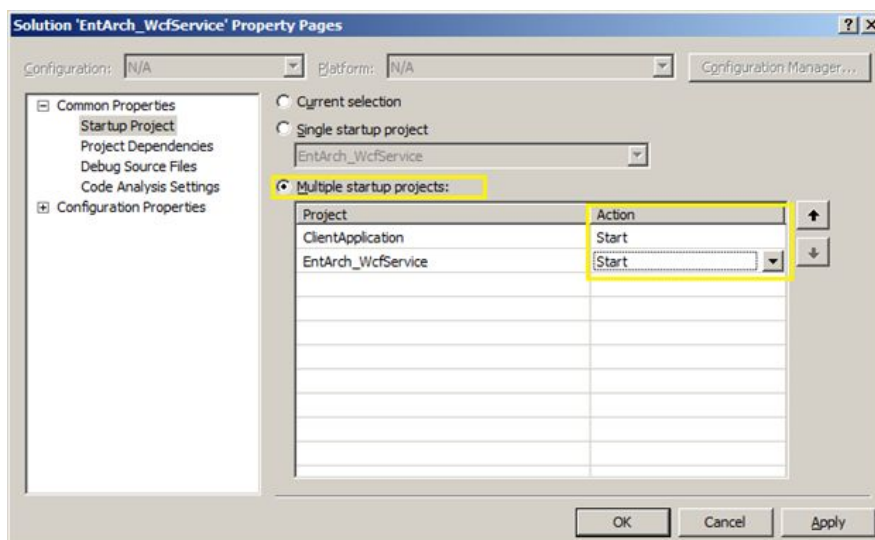
```
class CustomIdentity : IIdentity
{
    public string AuthenticationType
    {
        get;
        set;
    }

    public bool IsAuthenticated
    {
        get;
        set;
    }

    public string Name
    {
        get;
        set;
    }
}
```

**Step 12:** Create the client application and add references to both services.

Now we are ready to create the client application to consume both the services. Add a new Windows Forms Application to our existing solution and name it as ClientApplication. Make sure you made the Solution Properties > Start Up Projects to start both our Server and Client Applications as shown below.



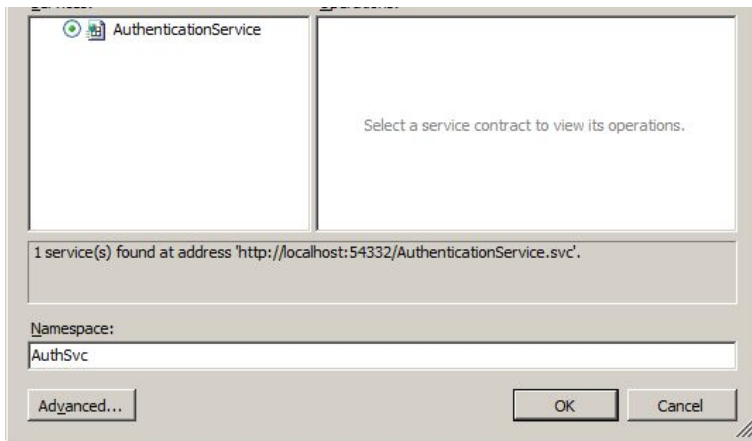
Now Add Service References to the following services:

1. AuthenticationService.svc
2. UtilityService.svc



ASK A QUESTION

CONTRIBUTE



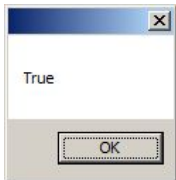
**Step 13:** Create an Authentication Service instance and invoke the Login() method.

Now we are ready to test our Authentication Service. Create a button on the Main Form and name it as Authenticate. On the button click create an instance of the client and invoke the Login() method with the right credentials. You should get a result of true.

```
private void AuthenticateButton_Click(object sender, EventArgs e)
{
    AuthSvc.AuthenticationServiceClient client =
        new AuthSvc.AuthenticationServiceClient();
    bool result = client.Login("tom", "chicago12", string.Empty, true);

    MessageBox.Show(result.ToString());
}
```

On running the client application and clicking the button you will receive the following output:



**Step 14:** Receive the cookie and store it.

The above code just checks the result of Login(). But we need to actually have the returned cookie in case of successful validation. For this we need to replace the above code with the following.

```
AuthSvc.AuthenticationServiceClient client = new AuthSvc.AuthenticationServiceClient();

using (new OperationContextScope(client.InnerChannel))
{
    bool result = client.Login("tom", "chicago12", string.Empty);
    var responseMessageProperty = (HttpResponseMessageProperty) OperationContext.Current.IncomingMessageProperties
        [HttpResponseMessageProperty.Name];

    if (result)
    {
        string cookie = responseMessageProperty.Headers.Get("Set-Cookie");
        MessageBox.Show(result.ToString() + Environment.NewLine + cookie);
    }
}
```

The code creates a new OperationContextScope and invokes the Login() method. After invocation the cookie is extracted from the response header.

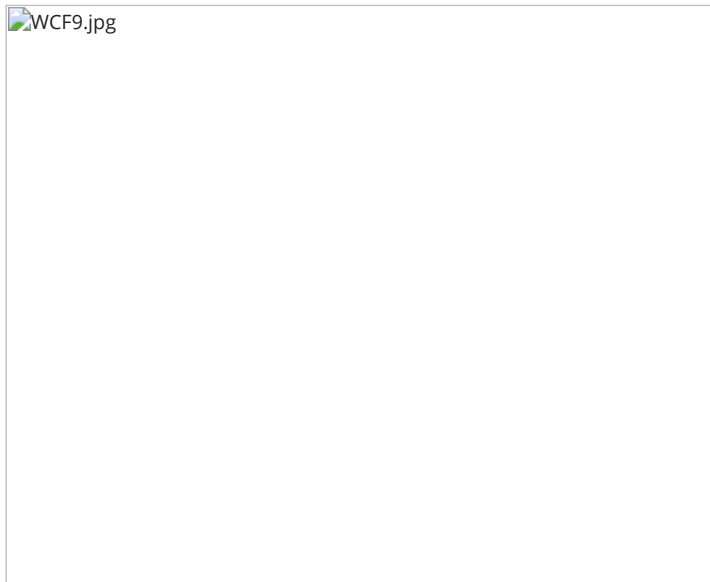
On running the client application again and clicking the button you should be getting the following output.



The output displays the result as well as the cookie information. You can see that the cookie name is .ASPXAUTH and the value is tom.

**Step 15:** Create UtilityService instance and invoke GetData() by attaching the stored cookie.

Inside the client application we can add a reference to the second service (UtilityService.svc).

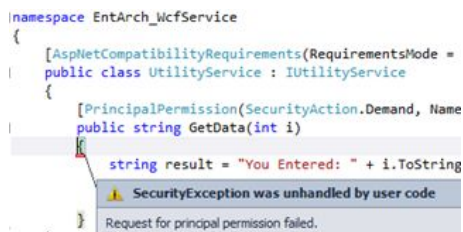


Now add a new button with text as Invoke Utility. On the button click handler, add the following code:

```
private void InvokeUtility_Click(object sender, EventArgs e)
{
    UtilSvc.UtilityServiceClient utilClient = new UtilSvc.UtilityServiceClient();
    string result = utilClient.GetData(10);
    MessageBox.Show(result);
}
```

On executing the application and clicking the button you will receive the following error:

"Request for principal permission failed."



The reason for the error is that the current thread user name is not tom. This error can be solved by doing the next step.

**Step 16:** Attach the Cookie to UtilityService client.

In order to solve the above error, we need to attach the cookie from the Authentication Service to the current Operation Context. We need to extract the variable cookie out of the Authentication Method so that it can be used in the Utility Invocation method.

CUS

C# Corner Annual Conference 2017 Recap



```
AuthSvc.AuthenticationServiceClient client = new AuthSvc.AuthenticationServiceClient();
```

ASK A QUESTION

CONTRIBUTE

```
bool result = client.ValidateUser("tom", "chicago12", string.Empty);
var responseMessageProperty = (HttpResponseMessageProperty)OperationContext.Current
    .OutputMessage.Properties["Set-Cookie"];
if (result)
{
    cookie = responseMessageProperty.Headers.Get("Set-Cookie");
    MessageBox.Show(result.ToString() + Environment.NewLine + cookie);
}
}
```

Now replace the contents of InvokeUtility\_Click with the following:

```
private void InvokeUtility_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(cookie))
    {
        MessageBox.Show("Please click Authenticate first.");
        return;
    }

    UtilSvc.UtilityServiceClient utilClient = new UtilSvc.UtilityServiceClient();

    using (new OperationContextScope(utilClient.InnerChannel))
    {
        HttpRequestMessageProperty request = new HttpRequestMessageProperty();
        request.Headers[HttpResponseHeader.SetCookie] = cookie;
        OperationContext.Current.OutgoingMessageProperties
            [HttpRequestMessageProperty.Name] = request;

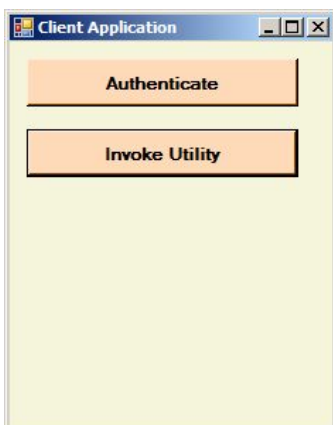
        string result = utilClient.GetData(10);
        MessageBox.Show(result);
    }
}
```

The method now performs the following activities:

1. Ensure the cookie field is has a value (Valid Authentication Service Call needed)
2. Create a Http Request Message Property set to cookie field
3. Attaches the Message Property instance to the Operation Context

**Step 17:** Test the application and ensure proper functioning.

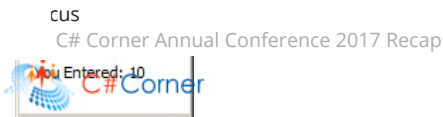
Now we are ready with the Authentication and Authorization infrastructure. Please run the application and test the features in 2 steps:



1. Click the Authenticate button and wait for the response
2. Click the Invoke Utility button

You will be getting the following Message Box on successful Authorization.





ASK A QUESTION

CONTRIBUTE

If you can see the above result.. Great!! You are done with Authentication and Authorization.

**Step 18:** Move the cookie attaching code to Interceptors in the Client Application.

Now we need to implement Interceptors in the client side.

### Why do we need Interceptors?

This is because the above code of attaching a cookie to the Operation Context seems to be a tedious job every time we need to do a service call. We can move these tasks to the background using WCF Interceptors.

MSDN: WCF Data Services enables an application to intercept request messages so that you can add custom logic to an operation. You can use this custom logic to validate data in incoming messages. You can also use it to further restrict the scope of a query request, such as to insert a custom authorization policy on a per request basis.

The following are the activities involved in this step.

1. Add Interceptor Behavior inside Client web.config
2. Create the Interceptor Behavior class
3. Create the Message Inspector class
4. Move the code from Form to Message Inspector class

To start with that, add the following code into the Client Application app.config file. Make sure you add them just before the `</system.serviceModel>` tag.

```
<!-- Interceptors-->
<behaviors>
  <endpointBehaviors>
    <behavior name="InterceptorBehaviour">
      <interceptorBehaviorExtension />
    </behavior>
  </endpointBehaviors>
</behaviors>
<extensions>
  <behaviorExtensions>
    <add name="interceptorBehaviorExtension" type="ClientApplication.InterceptorBehaviorExtension,
      ClientApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  </behaviorExtensions>
</extensions>
```

Change the Behavior Configuration element of the Utility Service section as shown below.

```
<client>
  <endpoint address="http://localhost:54332/AuthenticationService.svc"
    binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_AuthenticationService"
    contract="AuthSvc.AuthenticationService" name="BasicHttpBinding_AuthenticationService" />
  <endpoint address="http://localhost:54332/UtilityService.svc"
    binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IUtilityService"
    contract="UtilSvc.IUtilityService" name="BasicHttpBinding_IUtilityService"
    behaviorConfiguration="InterceptorBehaviour"/>
</client>
```

Now we need to create the Behavior Extension and Interceptor classes inside the Client Application.

// Behavior Extension Class - Attaches Cookie Inspector to the client behavior

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Description;
using System.ServiceModel.Configuration;

namespace ClientApplication
{
  public class InterceptorBehaviorExtension : BehaviorExtensionElement, IEndpointBehavior
  {
    public override System.Type BehaviorType
    {

```

CUS

C# Corner Annual Conference 2017 Recap



protected override object CreateBehavior()

ASK A QUESTION

CONTRIBUTE

```

-
public void AddBindingParameters(ServiceEndpoint endpoint, System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
{
}

public void ApplyClientBehavior(ServiceEndpoint endpoint, System.ServiceModel.Dispatcher.ClientRuntime clientRuntime)
{
    clientRuntime.MessageInspectors.Add(new CookieMessageInspector());
}

public void ApplyDispatchBehavior(ServiceEndpoint endpoint, System.ServiceModel.Dispatcher.EndpointDispatcher endpointDispatcher)
{
}

public void Validate(ServiceEndpoint endpoint)
{
}
}

// Cookie Inspector Class
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Dispatcher;
using System.ServiceModel.Channels;
using System.Net;
using System.Windows.Forms;

namespace ClientApplication
{
    public class CookieMessageInspector : IClientMessageInspector
    {
        private string cookie;

        public CookieMessageInspector()
        {
        }

        public void AfterReceiveReply(ref System.ServiceModel.Channels.Message reply,
            object correlationState)
        {
        }

        public object BeforeSendRequest(ref System.ServiceModel.Channels.Message request,
            System.ServiceModel.IClientChannel channel)
        {
            if (string.IsNullOrEmpty(Globals.Cookie))
            {
                MessageBox.Show("No Cookie Information found! Please Authenticate.");
                return null;
            }

            HttpRequestMessageProperty requestMessageProperty = new HttpRequestMessageProperty();
            requestMessageProperty.Headers[HttpResponseHeader.SetCookie] = Globals.Cookie;
            request.Properties[HttpRequestMessageProperty.Name] = requestMessageProperty;

            return null;
        }
    }
}

```

**ApplyClientBehavior:** This method enables us to attach client-side behaviors to the WCF calls. We are attaching a Cookie Inspector (CookieMessageInspector) in this method.

**CookieMessageInspector:** This class takes care of attaching the cookie to outgoing message properties (through BeforeSendRequest method). The cookie is saved in the Globals.Cookie property after the Authentication Service is called.

**Step 19:** Move the identity setting code to Interceptors in the Service Application.

CUS

C# Corner Annual Conference 2017 Recap



Extracting Cookie from Current Operation Context

ASK A QUESTION

CONTRIBUTE

- Setting user name to the Current Thread

This code for one utility seems to be fine. But in the case of dozens of Utility services the same code needs to be duplicated. We can solve this by using background interceptors in the service side. For this we can use the same BehaviorExtension and Inspector classes with slight modifications.

Place the following files in the WCF Service Application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Description;
using System.ServiceModel.Configuration;
using System.ServiceModel.Dispatcher;

namespace EntArch_WcfService
{
    public class InterceptorBehaviorExtension : BehaviorExtensionElement, IServiceBehavior
    {
        public override Type BehaviorType
        {
            get { return typeof(InterceptorBehaviorExtension); }
        }

        protected override object CreateBehavior()
        {
            return new InterceptorBehaviorExtension();
        }

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

        public void ApplyDispatchBehavior(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher dispatcher in serviceHostBase.ChannelDispatchers)
            {
                foreach (var endpoint in dispatcher.Endpoints)
                {
                    endpoint.DispatchRuntime.MessageInspectors.Add(new IdentityMessageInspector());
                }
            }
        }

        public void Validate(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase)
        {
        }
    }
}

// IdentityMessageInspector.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Dispatcher;
using System.ServiceModel.Channels;
using System.Net;
using System.IO;
using System.ServiceModel;
using System.Security.Principal;

namespace EntArch_WcfService
{
    public class IdentityMessageInspector : IDispatchMessageInspector
    {
        public object AfterReceiveRequest(ref Message request, System.ServiceModel.IClientChannel channel, System.ServiceModel.InstanceContext instanceContext)
        {
        }
    }
}
```

CUS

C# Corner Annual Conference 2017 Recap



```
OperationContext.Current.IncomingMessageProperties[HttpRequestMessageProperty]
string cookie = messageProperty.Headers.Get("Set-Cookie");
```

ASK A QUESTION

CONTRIBUTE

```
// Set User Name from cookie
if (nameValue.Length >= 2)
    userName = nameValue[1];

// Set Thread Principal to User Name
CustomIdentity customIdentity = new CustomIdentity();
GenericPrincipal threadCurrentPrincipal = new GenericPrincipal(customIdentity, new string[] { });
customIdentity.IsAuthenticated = true;
customIdentity.Name = userName;
System.Threading.Thread.CurrentPrincipal = threadCurrentPrincipal;

return null;
}

public void BeforeSendReply(ref Message reply, object correlationState)
{
}
}
```

Now remove the code from the UtilityService constructor. The new constructor will look empty.

```
public UtilityService()
{
}
```

Now add the following code to the web.config file of the service just under <system.serviceModel>

```
<services>
  <service name="EntArch_WcfService.UtilityService" behaviorConfiguration="InterceptorBehavior"/>
</services>
<behaviors>
  <serviceBehaviors>
    <behavior>
      <serviceMetadata httpGetEnabled="true"/>
      <serviceDebug includeExceptionDetailInFaults="true"/>
    </behavior>
    <behavior name="InterceptorBehavior">
      <interceptorBehaviorExtension />
    </behavior>
  </serviceBehaviors>
</behaviors>
<extensions>
  <behaviorExtensions>
    <add name="interceptorBehaviorExtension" type="EntArch_WcfService.InterceptorBehaviorExtension, EntArch_WcfService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  </behaviorExtensions>
</extensions>
```

Please make sure that the namespaces are correct and build the project.

**Step 20:** Modify the service-side code to include Role instead of Name.

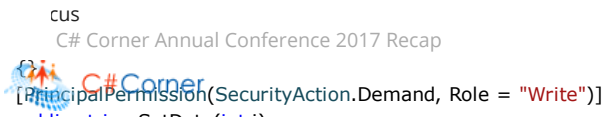
The current Authorization code is hardcoded with the user name Tom. But this is not feasible in the real world. We need to change the user name authorization to role-based authorization.

Here each validated user has certain roles associated with them. This role information will be fetched from the database during the Authentication process and passed along with the cookie to the user.

**Example:**

User	Roles
tom	Read
pike	Read, Write

The roles are kept as strings and we can separate methods based on the roles as given below:



To accomplish the above we need to modify the following.

1. Modify UserValidator.IsUserValid to return roles from database
2. Modify the AuthenticationService\_Authenticating.Authentication method to include roles
3. Concatenate the user name and roles in the cookie value as semicolon (;) separated
4. Modify the Identity Inspector to retrieve roles from the cookie value

The modifications are given below:

// UtilityService

```
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
public class UtilityService : IUtilityService
{
    public UtilityService()
    {
    }

    [PrincipalPermission(SecurityAction.Demand, Role = "Read")]
    public string GetData(int i)
    {
        string result = "You Entered: " + i.ToString();

        return result;
    }
}
```

// User Validator

```
public class UserValidator
{
    public bool IsUserValid(string userName, string password, out IList<string> roles)
    {
        roles = new List<string>();

        bool result = (userName == "tom") && (password == "chicago12");

        if (result) // If valid user return the Roles of user
            roles.Add("Read");

        return result;
    }
}

private string Concatenate(string userName, IList<string> roles)
{
    string result = userName + ";";

    foreach (string role in roles)
        result += role + ";";

    return result;
}
```

Please note that the **Read** role is hardcoded in the above code:

// Identity Message Inspector

```
public object AfterReceiveRequest(ref Message request, System.ServiceModel.IClientChannel channel, System.ServiceModel.InstanceContext instanceContext)
{
    // Extract Cookie (name=value) from messageproperty
    var messageProperty = (HttpRequestMessageProperty)
        OperationContext.Current.IncomingMessageProperties[HttpRequestMessageProperty.Name];
    string cookie = messageProperty.Headers.Get("Set-Cookie");
    string[] nameValue = cookie.Split('=', ',');
    string value
        = string.Empty;

    // Set User Name from cookie
    if (nameValue.Length >= 2)
        value = nameValue[1];
}
```

CUS

C# Corner Annual Conference 2017 Recap



Get Thread Principal to User Name

CustomIdentity customIdentity = new CustomIdentity();

ASK A QUESTION

CONTRIBUTE

```

        customIdentity.Name = value;
        System.Threading.Thread.CurrentPrincipal = threadCurrentPrincipal;

        return null;
    }

    private string[] GetRoles(string value)
    {
        if (!string.IsNullOrEmpty(value))
        {
            List<string> roles = new List<string>();

            int ix = 0;
            foreach (string item in value.Split(';'))
            {
                if (ix > 0)
                {
                    if (item.Trim().Length > 0)
                        roles.Add(item);
                }

                ix++;
            }

            return roles.ToArray<string>();
        }

        return new string[0];
    }

    private string GetUserName(string value)
    {
        if (!string.IsNullOrEmpty(value))
        {
            foreach (string item in value.Split(';'))
            {
                return item;
            }
        }

        return string.Empty;
    }

```

Once the above codes has been modified and ready, you can run the application. Try changing the role in the PrincipalPermission attribute to Write and you will get an error on accessing the GetData() method.

#### Step 21: Create Cookie Encryption.

As of now our cookie contains information that is easily readable by HTTP Examining Utilities like Fiddler. This makes the cookie information prone to security threats.



In this step we are going to add encryption to the cookie value. The above image displays the value which we are going to encrypt (username;Role)

FormsAuthenticationTicket: The System.Web.Security namespace provides a convenient class for us. We can store the user name and roles information inside this class instance. The ticket class also provides expiry and encryption facilities.

The following code creates the ticket:

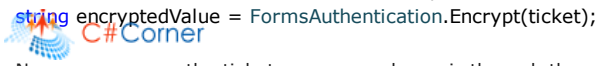
```

FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(
    1,
    e.UserName,
    DateTime.Now,
    DateTime.Now.AddHours(24),
    true,
    roles,
    FormsAuthentication.FormsCookiePath);

```

CUS

C# Corner Annual Conference 2017 Recap



ASK A QUESTION

CONTRIBUTE

```

HttpWebResponse response = new HttpWebResponse(response.Headers[HttpResponseHeader.SetCookie] = FormsAuthentication.FormsCookieName + "=" + encryptedValue;
OperationContext.Current.OutgoingMessageProperties[HttpResponseMessageProperty.Name] = response;

```

You need to specify the Cookie Name, Machine Key Properties inside the web.config as shown below:

```

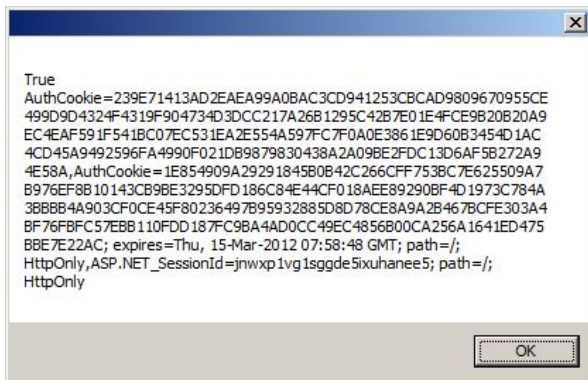
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <authentication mode="Forms">
      <forms slidingExpiration="true"
        name="AuthCookie"
        protection="All"
        timeout="20"/>
    </authentication>
    <machineKey
      decryption="AES"
      validation="SHA1"
      decryptionKey="1523F567EE75F7FB5AC0AC4D79E1D9F25430E3E2F1BCDD3370BCFC4EFC97A541"
      validationKey="33CBA563F26041EE5B5FE9581076C40618DCC1218F5F447634EDE8624508A129"
    />
  </system.web>

```

In the Identity Inspector we need to change the message property parsing code to reconstruct the Ticket.

// The updated code can be found in the attachment

On running the client application we can see that the message information is now encrypted.



## Step 22: Retest the Application.

Now perform the following activities:

1. Set the property of the client app.config to Copy Always.
2. Execute the Application
3. Click on the Authenticate button
4. Click on the Utility Invoke button

Upon performing the button clicks as specified in Step 17, the same results should be achieved.

CUS

C# Corner Annual Conference 2017 Recap



ASK A QUESTION

CONTRIBUTE

The GetData() method requires Read Role for the Authenticated User. You need to click the Authenticate button before invoking this method.

We can see the result as shown below:



This concludes our article on WCF Authentication using Cookies. I hope you enjoyed the learning curves and solutions.

## References

[MSDN - Authentication and Authorization](#)

[MSDN - PrincipalPermission](#)

[MSDN - ASP.NET Compatibility Mode](#)

[MSDN - Http Cookie](#)

[MSDN - WCF Interceptors](#)

[Cookie Inspector Example](#)

[MSDN - FormsAuthenticationTicket](#)

## Summary

In this article we have seen how to expose a WCF Service Application along with Authentication Service and Authorization according to Enterprise Architecting standards.

The following are the activities involved:

1. The client authenticates using the AuthenticationService.svc and receives a cookie.
2. The authenticated cookie is used in further communications.
3. The Authorization part is played by the PrincipalPermission attribute.
4. The WCF Interceptors do a good job by doing cookie attaching in the background.

You can note that the client needs to authenticate only once. Later the cookie received is used to access other services. Based on the real-time requirements we need to enable Ticket Expiry as well.

The attached source contains the projects we have explained. Please let me know of any clarifications or query you have.

In the next article we can see use of a Silverlight application to use the same service.

[Download 100% FREE Spire Office APIs](#)

asp.net

Custom Authentication

HTTP Cookies

WCF

Jean Paul *TOP 50*

Jean Paul is a Developer plus Architect working on Microsoft Technologies for the past 12 years. He is very much passionate in programming and his core skills are SharePoint, ASP.NET & C#. In the academic side ... [Read more](#)

<http://jeanpaulva.com/>

12

5.3m

4

[View Previous Comments](#)


5

20



CUS


C# Corner Annual Conference 2017 Recap




US: Don't forget to give your feedback, your help means that thanks is always...

ASK A QUESTION


CONTRIBUTE




NA 1 0




Good one :)  
Sibeesh Venu  
20 36.6k 3.6m




nice  
Vithal Wadje  
9 45k 15.9m




Nice article. Here is another option for securing WCF services using Custom Header encrypted tokens <http://wp.me/p3mRWu-CI>  
hoakim  
NA 3 0




easy understanding  
Gowtham Rajamanickam  
43 21.9k 1.9m




simle and awesome  
Gowtham Rajamanickam  
43 21.9k 1.9m




Thanks for nice article  
Santhakumar Munuswamy  
81 14.5k 412.8k



Thanks for the update KurianOfBorg! I will check on this soon.  
Jean Paul  
12 43.9k 5.3m



While this works, you're still resorting to creating a fake Identity on the server side and using a crude cookie to store the user name, not to mention all the code for manipulating HTTP headers. A better solution is to use the standard Forms Authentication cookie which is the same for both a regular Forms Authentication login page for the web interface well as the WCF Authentication Service. A native client can call the WCF Authentication Service to obtain the cookie, and any AJAX controls on the web site will automatically be authenticated if the user is logged into the site. This post shows how to transfer the principal from the cookie to the WCF service using a simple custom AuthorizationPolicy: <http://stackoverflow.com/a/11417296/274972>  
KurianOfBorg  
NA 1 0



Thank You Amit!  
Jean Paul  
12 43.9k 5.3m

0

0

reply

Apr 13, 2015

1

0

Reply

Apr 12, 2015

1

0

Reply

Apr 11, 2015

1

0

Reply

Apr 11, 2015

1

0

Reply

Apr 11, 2015

1

0

Reply

Feb 19, 2013

0

0

Reply

Feb 19, 2013

0

0

Reply

Jun 04, 2012

0


0

Reply

Comment Using


0 Comments

Sort by Oldest



Add a comment...

Facebook Comments Plugin



File APIs for .NET  
Aspose are the market leader of .NET APIs for file business formats – natively work with DOCX, XLSX, PPT, PDF, MSG, MPP, images formats and many more!

CUS

C# Corner Annual Conference 2017 Recap

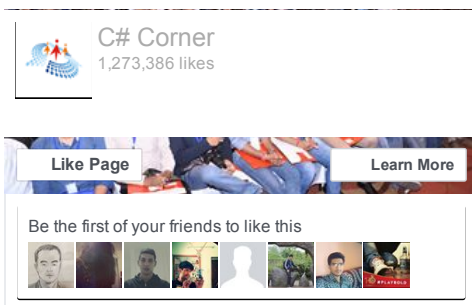
[ASK A QUESTION](#)[CONTRIBUTE](#)

## TRENDING UP

- 01 Basics And Overview Of ASP.NET MVC Core or MVC 6 In Visual Studio 2017
- 02 What's New In Angular 4.0
- 03 Singleton Design Pattern In C#
- 04 Factory Method Design Pattern In C#
- 05 Angular 4.0 Kickstarter
- 06 ASP.NET MVC - Passing Data From Controller To View
- 07 Insert Data By Stored Procedure In MVC 5.0 With Data First Approach
- 08 SPA Using Angular 2, ASP.NET Core 1.1 And Entity Framework Core - Part One
- 09 Steps To Perform CRUD Operations Using AngularJS And Stored Procedure In An ASP.NET MVC
- 10 \$q Service In AngularJS

[View All](#)

Follow @csharpcorner 90.8K followers



Philadelphia

New York

London

New Delhi

Mumbai

CUS

C# Corner Annual Conference 2017 Recap



A community of 2.3 million developers

[ASK A QUESTION](#)

[CONTRIBUTE](#)

[Learn ASP.NET MVC](#) [Learn ASP.NET Core](#) [Learn Node.js](#) [Learn Python](#) [Learn JavaScript](#) [Learn Xamarin](#) [More...](#)

[Home](#) [Events](#) [Consultants](#) [Jobs](#) [Career Advice](#) [Stories](#)

[About Us](#) [Contact Us](#) [Privacy Policy](#) [Terms](#) [Media Kit](#) [Sitemap](#) [Report a Bug](#) [FAQ](#)

©2017 C# Corner. All contents are copyright of their authors