WCF Security Features

# Features of WCF Membership Provider

WCF Membership providers allow applications to use the standard ASP.Net Membership API over a WCF channel. Some features include:

* Exposes a web service based API for common application security scenarios
* Expose any ASP.Net membership or role provider as a WCF web service
* ASP.Net compatible membership and role provider client via web service
* Use standard membership providers on client
* Built-in account activation and deactivation for use with “confirm your account” scenarios
* Rename user! This functionality is not possible with out-of-box .Net Membership provider
* Use any WCF bidirectional native protocols (WSE, WS-\*, TCP, JSON, etc.)
* Fat-calls to reduce number of cross-service boundary calls for common pathways

## Overview

This system contains two custom ASP.Net membership providers. The WCF client provider allows your web application to call a properly configured WCF service. The WCF service uses its own ASP.Net provider to provide security functionality. In a typical distributed solution your application will use ScrimpNet’s WCF provider as a client. Then you will configure the service to use the ASP.Net membership provider to your current operating environment.

When you want some advanced functionality, your application can directly call the ScrimpNet Security API. For quickest integration, it is recommended that you use the WCF providers. By keeping to the well documented ASP.Net security you will be able to have greater configuration and testability possibilities. Under the covers, the API uses the membership provider API where possible.

The library uses simple WCF end-points and traditional configuration. If your environment demands it, you can morph our code into custom behaviors or other advanced patterns.

Distributed functionality in any system is geometrically more complicated than simple systems. The more layers the more complexity. The more layers between the source call to the destination the more configuration is required.

The web service ScrimpNet.Security API strives to be as simple as possible but it hasn’t been able to completely free the developer from some setup tasks needed to get the moving parts working together.

# Installing WCF Membership Provider

To setup a functional distributed membership provider you need to:

1. Obtain WCF Membership client and server libraries
2. Install and secure (optional) the web service
3. Connect your web (or forms) application to WCF membership client

## Software Libraries

You can get the libraries from various places including: <http://scrimpnet.codeplex.com>. Here’s a list of some of the things you get:

|  |  |
| --- | --- |
| ScrimpNet.Core | Common library functions and referenced by all ScrimpNet libraries. This library does not have references to any other project and can be used in any project (not only Security) |
| ScrimpNet.BaseModelContracts | At ScrimpNet we have common functionality that is shared for WCF message request-response patterns including error handling, message passing, and end-to-end tracing. These base patterns are not part of the Security API and can be reused across different types of projects. |
| ScrimpNet.Security.Core | Contains the bulk of the security logic that implements the ScrimpNet.Security API. |
| ScrimpNet.Security.Client | Wraps WCF Security API for the client application. For the most part it is simply a wrapper for the WCF end-point. Your application should reference this library when not using a DotNet security provider. |
| ScrimpNet.Security.Contracts | WCF contracts for exchanging messages between your application and the ScrimpNet Security Server |
| ScrimpNet.Security.DotNet | Custom WCFMembershipProvider and WCFRoleProvider. These classes hook your application to the WCF Membership end-point using standard ASP.Net provider configuration.  Custom SQLMembership and SQLRole providers. SQL providers are standard .Net providers with a little configuration enhancement. Provides a good example on how to extend an existing provider. This example uses the custom providers but your application should work fine using out-of-the-box providers (SQL, AD, or Oracle) |
| ScrimpNet.Security.WCFHost | Sample hosting application with annotated sample configuration file. |
| ScrimpNet.Secuirty.DotNetTests | Tests to exercise the API and providers. You can use these tests to verify your server installation. These tests are extensive enough to support custom .Net providers to ensure a high degree of compatibility with ASP.Net security controls. |

## Installing Server

1. Set up the ASP.Net membership provider of your choice. This documentation assumes you will be using the default ASP.Net SQL provider but you can use any other provider. The ScrimpNet.Security server will expose the capabilities of your provider. [[1]](#footnote-1)
2. Create a WCF Host project. The easiest way to get started is to begin with the sample. If you are new to WCF the sample should be adequate. If you are a WCF pro, you’ll probably want to tweak things to match your environment.
3. Create a .svc end-point that points to ScrimpNet.Security.Core.WcfCoreSecurity.

<%@ ServiceHost Language="C#" Debug="true" Service="ScrimpNet.Security.Core.WcfCoreSecurity" %>

1. Configure WCF end-point. The simplest way to get started is to copy the <system.servicemodel> section from the web.config file of ScrimpNet.Security.WcfHost. Be sure your end-point is configured for ScrimpNet.Security.Contracts.IWcfSecurityService and the project references[[2]](#footnote-2):  
    ScrimpNet.Core  
    ScrimpNet.Security.Contracts  
    ScrimpNet.Security.Core  
    ScrimpNet.ServiceModel.BaseContracts
2. Web.config. Follow configuration steps for your chosen membership provider[[3]](#footnote-3). The underlying Security API uses the default provider as specified in <system.web>. If you use the ScrimpNet.SqlMembershipProvider, you can take advantage of:  
    \* Robust error messages to help trouble shoot configuration and setup issues  
    \* ScrimpNet’s meta token replacement features[[4]](#footnote-4)  
     
   ScrimpNet.Sql providers uses the default configuration parameters for ASP.Net SqlMembershipProvider. ScrimpNet.Sql providers can use tokens to specify application keys and connection string variants to support switching between environments. It is common to use {app} and {env}.  
     
    connectionStringName="ScrimpNet.Security.{env}"

applicationName="{app}" />

These settings tell the provider to choose the appropriate connection string based on environment and to use the default system application name (a common value for many applications and a requirement for ScrimpNet libraries) as the provider’s application name. Of course these {meta} tokens are optional and you can hardcode values.

1. web.config (appSetting). Choose and set ScrimpNet.Security.AuthenticationKey.

What is this?

An authentication key is sent by the caller to the server to prove the caller is allowed to access the ScrimpNet.Security server. It is a string value and can be any non-null value.

When to use This?

If your WCF Client is in a secure environment such as your production web servers, then you don’t need to supply any value.

If your WCF client is being called from a non-secure environment then you might provide a unique token, such as a GUID to each caller. ,(e.g. you are hosting the security API as a service or the client is on a public facing, unsecured site)

How is this used?

The security service verifies the GUID provided on each call against its internal value and returns an error if it doesn’t match.

What Is Your Authentication Key:   
 *(you’ll need this value when configuring the client)*

1. web.Config(appSetting). Choose and set ScrimpNet.Security.EncryptionKey

What is this?

This is a “password” that is used to encrypt data between client and the ScrimpNet.Security server. It is technically a symmetric key.

When to use this?

Use this key when you are sending data across an unsecure transport layer (“pipe”). For most installations you will not need to provide a value. If the client is unsecure (hosted on an ISP) and calling to the service behind a firewall then you might consider encryption.[[5]](#footnote-5)

How is this used?

If left blank, then no encryption occurs. If a value is supplied the client uses this as part of a key to encrypt data sent to the server and decrypt data received from the server.

What Is Your Encryption Key:   
 *(you’ll need this value when configuring the client)*

1. web.Config(appSetting). Set ScrimpNet.Application.Name

What is this?

A unique identifier for this service and used most frequently in logging.   
Recommended value: ‘ ScrimpNet.Security.Server’

When to use this?

It is a required field and referenced by all ScrimpNet libraries

How is this used?

You can use any non-null value. This value is available through the ‘{app}’ configuration {meta] token. Many organizations have a similar setting. If so, you can reference your value by using the {%app:} configuration {meta} token.

<add key="ScrimpNet.Application.Name" value="{%app:ProgramName%}"/>

<add key="ProgramName" value="Services"/>

Results in ScrimpNet.Application.Name having a value of ‘Services’

1. web.confg(appSetting). Set ScrimpNet.Application.Environment

What is this?

Describes where the server is functioning (e.g. developer’s workstation, QA, stage, production, hot-fix)

When to use this?

It is a good discipline to define your operating environments. If you don’t define value, the ScrimpNet libraries will automatically assign the machine name of the hosting computer as the environment name. A warning will be written to the server’s LastChanceLog in the \bin folder.[[6]](#footnote-6)

How is this used?

The libraries use this primarily in logging operations such as filtering and grouping. This value is available through the ‘{env}’ configuration {meta} token. Many organizations have a similar setting. If so, you can reference your value by using the {%app:} configuration {meta} token.

<add key="ScrimpNet.Application.Environment" value="{%app:Environment%}"/>

<add key="Environment" value="QA-1"/>

Results in ScrimpNet.Application.Environment having a value of ‘QA-1’

That’s it. To expose your membership provider(s) as a WCF web service simply:

1. Create a service that references ScrimpNet.Security.Core.WcfService
2. Configure your membership provider using its standard configuration
3. Set up a couple of support configuration settings.
4. Test your service by viewing the service’s .svc file or connecting by your app. That’s next.

## Installing Client

1. Add references to ScrimpNet libraries in your project. These can be either project or binary references. If you are doing development, we suggest project references so you can tweak the code (or ‘groan’ fix it.) In production, you might want to deploy the project as binary references.  
    ScrimpNet.Core  
    ScrimpNet.Security.Contracts  
    ScrimpNet.Security.Core  
    ScrimpNet.ServiceModel.BaseContracts
2. Create membership/role provider in your <system.web> configuration of your web site.[[7]](#footnote-7)

<membership defaultProvider="SecurityServiceMembershipProvider">

<providers>

<clear/>

<add name="SecurityServiceMembershipProvider"

type="ScrimpNet.Security.WcfProviders.WcfMembershipProvider,ScrimpNet.Security.DotNet"

serviceUri="{%app:ScrimpNet.SecurityServer.Uri%}"

authenticationKey="{%app:ScrimpNet.Security.AuthenticationKey%}"

encryptionKey="{%app:ScrimpNet.Security.EncryptionKey%}"

/>

</providers>

</membership>

<roleManager enabled="true" defaultProvider="SecurityServiceRoleProvider">

<providers>

<clear/>

<add name="SecurityServiceRoleProvider"

type="ScrimpNet.Security.WcfProviders.WcfMembershipProvider,ScrimpNet.Security.DotNet"

serviceUri="{%app:ScrimpNet.Security.ServerUri%}"

/>

</providers>

</roleManager>

1. Add entries into your <appSettings>

<appSettings>

<add key="ScrimpNet.Security.ServerUri"   
 value="security service url (e.g. http://localhost:3679/SecurityService.svc)" />

<add key="ScrimpNet.Security.AuthenticationKey"  
 value="your authentication key" />

<add key="ScrimpNet.Security.EncryptionKey"  
 value="your encryption key" />

</appSettings>

That’s it! You application now can use the WCF membership and role providers like any other provider!

1. To create the SQL tables and procedures see: <http://www.4guysfromrolla.com/articles/040506-1.aspx> [↑](#footnote-ref-1)
2. The sample solution uses project references but binary and/or GAC references work fine [↑](#footnote-ref-2)
3. See <http://msdn.microsoft.com/en-us/library/ff648345.aspx> for details on configuring SQL and ActiveDirectory providers [↑](#footnote-ref-3)
4. Token replacement enables configuration settings to be looked up in other places in the .config file. Very helpful in centralizing settings (e.g. passwords, URLs) instead of repeating the same values. See appendix A at end of this document for a brief description. [↑](#footnote-ref-4)
5. The channel can be secured via WCF WS-\* protocols or SSL. EncryptionKey should be considered a weak and/or a temporary alternative to robust standards. [↑](#footnote-ref-5)
6. The only way to suppress the warning is to configure a value. [↑](#footnote-ref-6)
7. We are using ScrimpNet’s configuration token replacement syntax here. The tokens point to <appSettings> configuration entries so more entries are centralized and not scattered throughout the .config file. You can use actual values as well. [↑](#footnote-ref-7)