… a demonstration of securing a simple WCF service using certificates.

### Background

A consistent set of fundamental security concepts apply in any distributed messaging system. The following concerns offer increasing message protection:

* Authentication: Identifying the message sender. Mutual authentication involves authenticating both the sender and the message receiver, to prevent possible man-in-the-middle attacks.
* Authorization: After authenticating the message sender, authorization determines what system features and functionality they are entitled to execute.
* Integrity: Messages should be digitally signed to ensure they have not been altered between sender and receiver.
* Confidentiality: Sensitive messages or specific message parts should be encrypted to ensure they cannot be openly viewed on the wire.

The first step to securing a WCF service is defining the security policy. Once the policy is defined, the configuration for authentication, authorization, and message protection is achieved declaratively or through the runtime object model. The choice of binding determines what security features are supported. OmnyxDPS uses NetTcpBinding (a binary encoded TPC protocol) for all bindings except for streaming client which uses http.

There are five basic WCF security modes:

* None: Turns security off.
* Transport: Uses transport security for mutual authentication and message protection.
* Message: Uses message security for mutual authentication and message protection.
* Both: Allows for transport and message-level security (only MSMQ supports this).
* TransportWithMessageCredential: Credentials are passed with the message and message protection and server authentication are provided by the transport layer.
* TransportCredentialOnly: Client credentials are passed with the transport layer and no message protection is applied.

Each transport protocol (TCP, IPC, MSMQ, or HTTP) has their own mechanism for passing credentials and handling message protection. Message security supports passing credentials as part of the SOAP message using interoperable standards, and also makes it possible to protect the message independent of transport all the way through to the ultimate message receiver. Transport message protection is only good from point to point.

This demo will focus on a simple WCF client server application secured with certificates to authenticate via a message based WS HTTP binding.

## Setting up Certificates

#### Definitions

X.509 digital certificates are commonly used to authenticate clients and servers, encode messages, and digitally sign messages. A digital certificate is a part of a public key infrastructure (PKI), which is a system of digital certificates, certificate authorities, and other registration authorities that verify and authenticate the validity of each party involved in an electronic transaction through the use of public key cryptography. A certificate authority issues certificates and each certificate has a set of fields that contain data, such as subject (the entity to which the certificate is issued), validity dates, issuer, and public key.

Certificate Store: a physical store managed by the OS to store certificates locally.

Makecert : a Microsoft SDK tool to generate certificates for testing and development.

Makecert was used to generate and install certificates in the Certificate Store on the server. Certificates were configured to enable the export of the private key. A base certificate was self signed [bbcrevisit] and used to sign the client and server certificate.



The base certificate, client certificate and the public key of the server certificate were installed on the client.



## Setting up WCF for Security

### Negotiation

To support mutual authentication and message protection, services must provide credentials to the caller. When transport security is used, service credentials are negotiated through the transport protocol. Service credentials for message security can also be negotiated when Windows credentials are used; otherwise a service certificate must be specified in the <behaviors> section under <serviceCredentials>. In this case**, the caller must have access to the public key portion of the service certificate to encrypt messages sent to the service**. This can be specified out of band, or negotiated with an initial handshake.  
  
The default behavior for message security supports negotiation. That means that the service is dynamically asked for the correct token before any messages are exchanged. For Windows client credentials SPNego protocol is used, and for UserName, Certificate or Anonymous credentials, TLSNego protocol is used. Today these are not interoperable protocols, so it may be desirable to disable this negotiation.  
  
You can set negotiateServiceCredential to false in the <message> section to accomplish this.

When negotiation is disabled for Windows client credentials, a Kerberos domain must exist. For other credential types the client must have access to the service public key to encrypt messages.

### Protection Level

By default, all secure WCF bindings will encrypt and sign messages. The protection levels defined for WCF are None, Sign, and EncryptAndSign. **Encryption cannot be disabled for transport security.** For message security, we can configure the binding to sign the messages but not encrypt the messages.

Protection-level settings are controlled by the contract. They can be set for all operations in the service contract using the ServiceContractAttribute. The following example illustrates disabling encryption.

[ServiceContract(

…

**ProtectionLevel=ProtectionLevel.Sign**

)]

public interface IService

{

…

Message protection can be specified at the finer operation level using the OperationContractAttribute.

[ServiceContract

…

]

public interface IService

{

[OperationContract(**ProtectionLevel= ProtectionLevel.Sign**)]

string ISErviceOperation();

}

To summarize, to enable authentication without encryption in message based protection the net.TCP binding must be used.

### Secure Session

A feature of message security is the ability to establish a secure session to reduce the overhead of one-off key exchange and validation**. By default, secure sessions are enabled for message security**. A security context token (SCT) is generated through an initial exchange between caller and service. This token is used to authorize and secure subsequent message exchanges.  
  
If the caller plans to make several calls to a service, secure sessions are more efficient. For a single call, however, you can disable this feature by setting establish SecurityContext to false.

<wsHttpBinding>

<binding name="wsHttp">

<security mode="Message">

<message …

**establishSecurityContext="false"** />

## Demo App.config settings for Client and Server

### Server Settings

<system.serviceModel>

<bindings>

<wsHttpBinding>

<binding name="DogServiceBinding">

<security mode="Message">

<message clientCredentialType="Certificate" negotiateServiceCredential="false" />

</security>

</binding>

</wsHttpBinding>

</bindings>

<services>

<service behaviorConfiguration="DogServiceBehavior" name="WcfServiceLibrary2.Service1">

<endpoint address="http://10.116.1.194:444/Dogs/DogPoundSecure" binding="wsHttpBinding" bindingConfiguration="DogServiceBinding" contract="WcfServiceLibrary2.IService1" />

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="DogServiceBehavior">

<serviceCredentials>

<clientCertificate>

<authentication certificateValidationMode="PeerTrust" />

</clientCertificate>

<serviceCertificate findValue="CN=DogServer" storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectDistinguishedName" />

<issuedTokenAuthentication allowUntrustedRsaIssuers="true" />

</serviceCredentials>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

### ClientSettings

<system.serviceModel>

<bindings>

<wsHttpBinding>

<binding name="DogServiceBinding">

<security mode="Message">

<message clientCredentialType="Certificate" negotiateServiceCredential="false" />

</security>

</binding>

</wsHttpBinding>

</bindings>

<services>

<service behaviorConfiguration="DogServiceBehavior" name="WcfServiceLibrary2.Service1">

<endpoint address="http://10.116.1.194:444/Dogs/DogPoundSecure" binding="wsHttpBinding" bindingConfiguration="DogServiceBinding" contract="WcfServiceLibrary2.IService1" />

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="DogServiceBehavior">

<serviceCredentials>

<clientCertificate>

<authentication certificateValidationMode="PeerTrust" />

</clientCertificate>

<serviceCertificate findValue="CN=DogServer" storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectDistinguishedName" />

<issuedTokenAuthentication allowUntrustedRsaIssuers="true" />

</serviceCredentials>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>