Implementing Message Layer Security with X.509 Certificates in WSE 3.0

Context

You are implementing brokered authentication in an application deployed on a Windows platform with security implemented at the message layer. A Web service using Web Services Enhancements (WSE) 3.0 is processing requests from clients. Clients and services use X.509 certificates with a standards-based security token that is portable across organizations and security boundaries. The solution must be able to provide a comprehensive set of security features that includes mutual authentication, data origin authentication, and data confidentiality.

Objectives

The objectives of this pattern are to:

- Secure a message exchange between two parties using brokered authentication with X.509 certificates.
- Combine an implementation of mutual authentication with the Data Origin Authentication and Data Confidentiality design patterns to provide a baseline that you can use to add more security requirements, such as replay protection and message validation.
- Demonstrate the implementation of the WSE 3.0 **mutualCertificate10Security** policy assertion and discuss when to use the **mutualCertificate11Security** policy assertion.
- Demonstrate an implementation of a custom WSE 3.0 **X509SecurityTokenManager** that allows you to associate additional data, such as roles with a client certificate.

Content

This pattern consists of the following sections:

- **Implementation Strategy**. This section provides a high-level description of the strategy to implement brokered authentication using X.509 certificates.
- **Implementation Approach**. This section describes the steps required to implement this pattern:
  - General setup
  - Configure the client
  - Configure the service

- **Resulting Context**. This section outlines the benefits, liabilities, and security considerations related to this pattern.
- **Extensions**. This section discusses how to extend the base pattern to implement role-based authorization.

Note: The code examples in this pattern are also available as executable QuickStarts on the Web Service Security community workspace.

Implementation Strategy

Use the **mutualCertificate10Security** policy assertion in WSE 3.0 to enable message signing and encryption using X.509 certificates. WSE 3.0 policy accesses the client's private key, which is used to sign the message. The service's public key, which is in its X.509 certificate, then encrypts the message. The service decrypts the message using its private key and verifies the signature using the public key of the client. The public key is in the client's X.509 certificate, which is included with the message.

Unlike using Secure Sockets Layer (SSL) in which the client obtains the service's X.509 certificate at run time, message layer security using X.509 certificates in WSE 3.0 requires that the service's certificate is obtained out-of-band and installed in the client's local certificate store.

Note: This pattern uses the **mutualCertificate10Security** policy assertion, because it relies on WS-Security 1.0. However, if your environment fully supports WS-Security 1.1 extensions, you can use the **mutualCertificate11Security** policy assertion. The **mutualCertificate11Security** policy assertion provides better performance, because it performs less asymmetric cryptography operations, which are computationally intensive. It performs two asymmetric and two symmetric operations compared with four asymmetric operations for the **mutalCertificate10Security** policy assertion.

This pattern assumes that the client has already obtained the service's certificate out-of-band, so that it can access the service's certificate from a local certificate store. For more information about installing X.509 certificates in the local certificate store, see How to: Use the X.509 Certificate Management Tools.

Participants

Using message layer security with X.509 certificates in WSE 3.0 involves the following participants:

- **Client**. The client accesses the Web service, and provides credentials for authentication during the request to the Web service.
- **Service**. The service is the Web service that requires authentication of the client to make access control decisions.

Process

The "Process" section of Brokered Authentication: X.509 PKI in Chapter 1, "Authentication Patterns," describes how you can use a certificate for authentication. This pattern provides a more detailed description of that process within the context of the implementation. The steps provided are based on the behavior of the **mutualCertificate10Security** assertion. The steps are divided into the following two sections based on what happens on the client and then on the service:

- The client initializes and sends a message with X.509 certificate information.
- The service authenticates the client using the X.509 certificate and signature.

The Client Initializes and Sends a Message with X.509 Certificate Information

This part of the process has six steps:

1. The client retrieves the service's X.509 certificate.
2. The client retrieves its own certificate and private key.
3. The client attaches its X.509 certificate to a message.
4. The client signs the message using its private key.
5. The client encrypts the message using the service's public key.
6. The client sends the message to the service.

The steps are summarized in Figure 3.8.

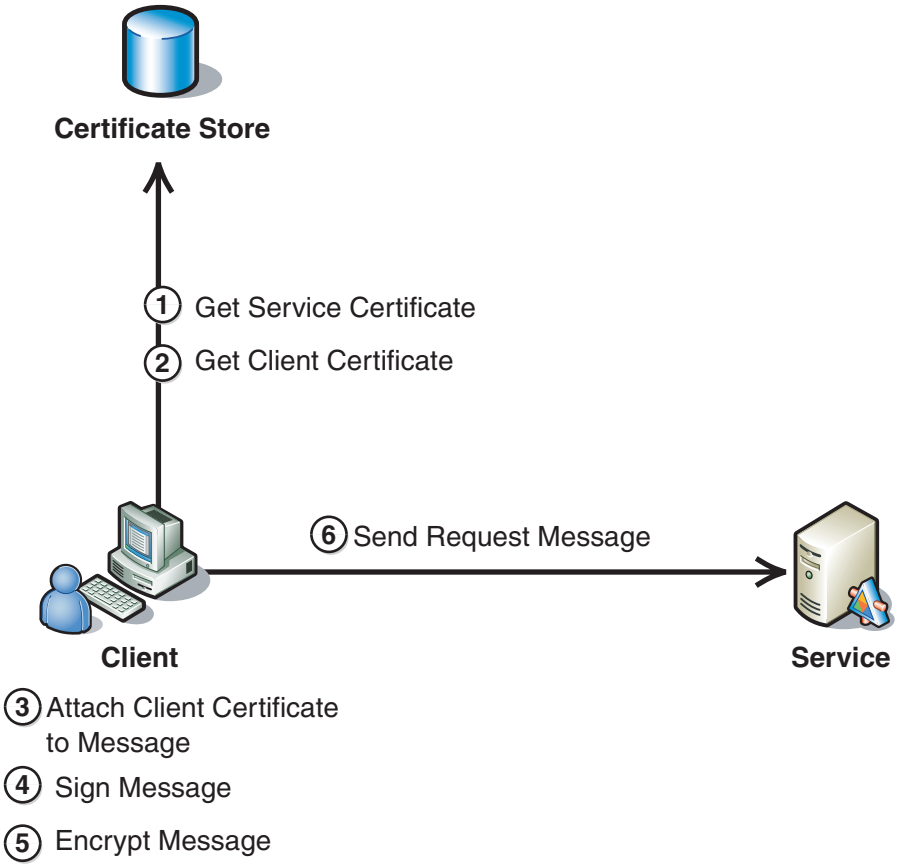

Figure 3.8
Initializing and sending a message with X.509 certificate information

Step One: The Client Retrieves the Service's Certificate

The client needs to access the X.509 certificate of the service to encrypt the request message. The WSE 3.0 policy assertion on the client is configured to retrieve the service's certificate from the client's local certificate store without the need for any additional code.

Step Two: The Client Retrieves Its own X.509 certificate and Private Key

The client accesses its X.509 certificate and private key. It uses the private key to sign the message and the X.509 certificate to provide the service with the public key and other information about the client for verification with the service.

Step Three: The Client Attaches Its X.509 Certificate to a Message

WSE 3.0 policy is configured to sign the message, and WSE 3.0 automatically attaches the client's certificate to the request message.

Step Four: The Client Signs the Message Using Its Private Key

The client uses its private key to sign the message. You can choose to sign one or more portions of the message, such as the address header or the message body. At a minimum, you should sign the message body, security, and addressing headers. A signature is created using a signature algorithm that computes a checksum value from the data to be signed and then encrypts the checksum value with the client's private key. When the signature is validated, the data used to create the signature is also validated to provide data origin authentication.

Step Five: The Client Encrypts the Message Using the Service's Public Key

You can encrypt message parts using a symmetric key that is encrypted with the public key from the service's X.509 certificate. At a minimum, ensure that the signature used to sign the encrypted data is itself encrypted to help protect it against offline attacks.

When you use WSE 3.0 policy to encrypt message data with X.509 certificates, the policy uses asymmetric encryption to encrypt a one-time symmetric key, which in turn encrypts the data. When message data is encrypted using the service's certificate information, WSE 3.0 also adds the certificate identifier to the message. If the certificate contains a subject key identifier, this is included to identify the certificate in the message. Otherwise, the policy uses the issuer name and certificate serial number instead. The service owns the certificate, which contains all the necessary information for it to access the appropriate private key and decrypt the symmetric key, which is then in turn used to decrypt the message.

Encrypting the request in this way protects sensitive data if the client is deceived into calling an illegitimate service. As the intended message recipient, only the correct Web service can decrypt the message with its private key.

Step Six: The Client Sends the Message to the Service

After the message is signed and encrypted, the client sends it to the service.

The Service Authenticates a Client Using the X.509 Certificate and Signature

This part of the process has six steps:

1. The service validates the client's certificate.
2. The service verifies the certificate trust chain.
3. The service checks the certificate revocation status.
4. The service decrypts the message.
5. The service verifies the signature.
6. The service initializes and sends a response to the client (optional).

The steps are summarized in the Figure 3.9.

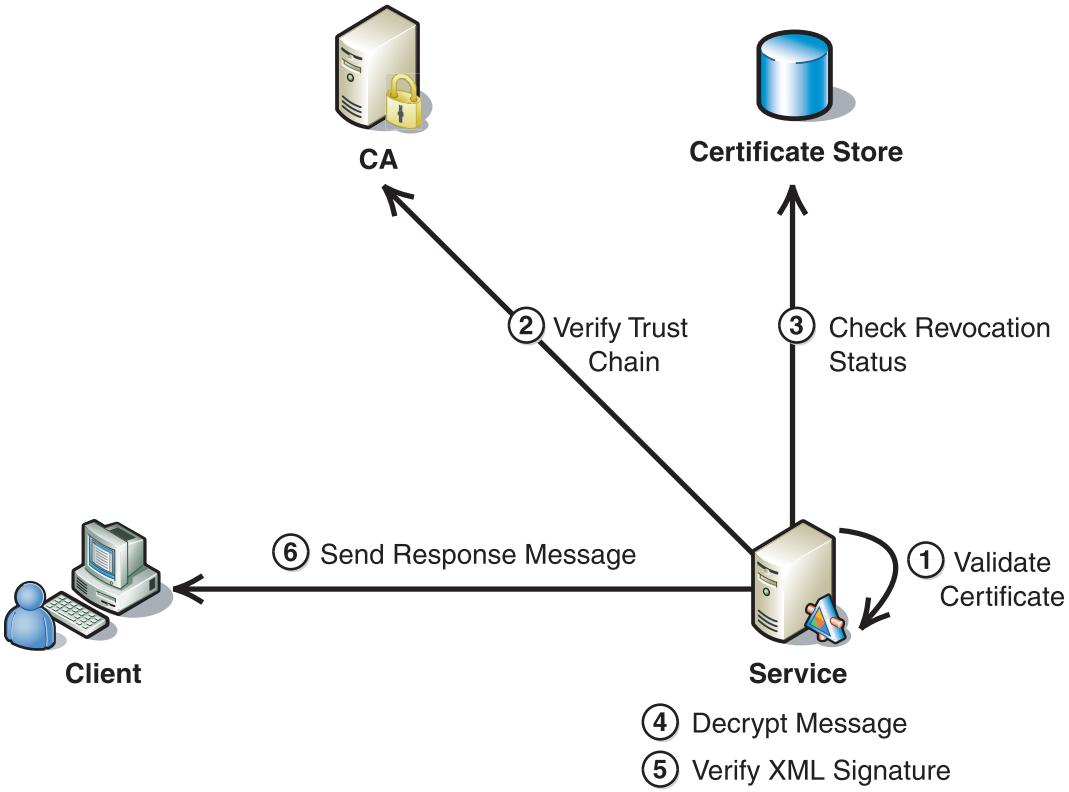

Figure 3.9
Authenticating a client using an X.509 certificate and signature

Step One: The Service Validates the Client's Certificate

WSE 3.0 validates the client's certificate attached to the request message. The certificate's validity period is checked to ensure that the service does not process a request that was secured with an expired X.509 certificate.

WSE 3.0 also verifies the integrity of the certificate's contents to ensure that it has not been tampered with after the certificate authority (CA) issued it. The integrity of the certificate's contents is verified using the signature of the issuing CA, which is also included in the certificate. If the certificate's contents cannot be validated against the issuer's signature, then the certificate has been tampered with and it is rejected as invalid. For more information about the contents of an X.509 certificate, see the X.509 Technical Supplement in Chapter 7, "Technical Supplements."

Step Two: The Service Verifies the Certificate Trust Chain

By default, WSE 3.0 verifies the trust chain of certificates, or requires that the client's certificate is installed in the **Trusted People** folder in the service's local certificate store. WSE 3.0 must be able to recognize an issuing CA as trusted to verify the certificate trust chain for the client's X.509 certificate. WSE 3.0 recognizes an issuing CA as trusted based on the X.509 certificate that endorses the client's certificate. WSE 3.0 recognizes the issuing CA's certificate as a trusted root for a certificate chain if the CA's X.509 certificate is installed in the machine certificate store in the **Trusted Root Certification Authorities** folder.

The high-level steps to install a certificate chain are as follows:

1. Export the certificate chain from the CA. This is dependant on the type of CA that issued the certificate.
2. Import the certificate chain into a local certificate store.

Note: For more information about managing certificates and trust chains, see the X.509 Technical Supplement in Chapter 7, "Technical Supplements."

Step Three: The Service Checks the Certificate Revocation Status

WSE 3.0 policy checks the revocation status of the certificate by verifying whether the certificate is on a certificate revocation list (CRL) that the CA publishes. You can obtain the CRL out-of-band by downloading it from a CA, and then importing it into a local certificate store where WSE 3.0 can access it. You can also check the revocation status of the certificate online. However, this approach relies on an online revocation service that the service must access to verify the certificate's revocation status. There is also a performance cost associated with checking the revocation status online. For this reason, you may want to consider downloading the CRL instead, if you can frequently update the cached CRL. By default, WSE 3.0 verifies the revocation status of X.509 certificates online.

Note: For more information about CRLs, see the X.509 Technical Supplement in Chapter 7, "Technical Supplements."

Step Four: The Service Decrypts the Message

By default, the **mutualCertificate10Security** assertion protects the message body by encrypting it. When WSE 3.0 receives an encrypted message, WSE 3.0 policy automatically decrypts it using the following steps:

1. WSE determines the value to identify the service's certificate — either the RFC3280 Subject Key Identifier, or the issuer name and serial number — that the client included in the message tells the service which certificate was used to encrypt the message. WSE 3.0 policy uses this value to determine which private key it must use to decrypt the message.
2. WSE decrypts the asymmetrically encrypted, one-time symmetric key that the client sent with the message, using the service's private key
3. WSE uses the symmetric key to decrypt the message data using a symmetric algorithm. By default, WSE 3.0 uses AES 256 for symmetric encryption.

Note: Service side policy alone does not stop a client from sending an unencrypted message. However, policy will reject a message at the server if it is not encrypted.

Step Five: The Service Verifies the Signature

WSE 3.0 verifies the client's signature on the incoming request message using the public key sent with the message. If the message data is signed, this step also validates the client as the message originator to provide data origin authentication.

Step Six: The Service Initializes and Sends a Response to the Client (Optional)

If the service returns a secure response to the client, the same process described in these steps is used for the response message between the service and the client, except that the roles of the client and the service reverse. However, unlike the request message, the service does not attach its X.509 certificate to the response message, because the client already has a copy of it.

Instead, WSE 3.0 policy adds a reference to the service's certificate in the response message. The service initiates and sends the response, signs it with the service's private key and encrypts it with a symmetric key that is encrypted with the client's X.509 certificate public key. The client processes the response in the same manner as the service processed the request: decrypt the symmetric key with the client's private key, and then decrypt the encrypted message parts with the symmetric key. Finally, the client verifies the service's signature with the service's X.509 certificate.

Implementation Approach

This section describes how to implement this pattern. The section is divided into three major tasks:

1. **General setup**. This task provides the required steps for both the client and the service.
2. **Configure the client**. This task provides the required steps to configure WSE 3.0 policy and the code on the client.
3. **Configure the service**. This task provides the required steps to configure WSE 3.0 policy and the code on the service.

Note: For the code examples included in this pattern, an ellipsis (...) is used where segments of code, such as class declarations and designer-generated code, have been omitted. You must name variables, methods, and return values and ensure that they are of the appropriate type for the client application.

General Setup

You must install WSE 3.0 on the computers that you use to develop WSE-enabled applications. After WSE 3.0 is installed, you must enable the client and the service to support WSE 3.0. You can achieve this by performing the following steps:

► **To enable a Visual Studio 2005 project to support WSE 3.0**

1. In Visual Studio 2005, right-click the application project, and then click **WSE Settings 3.0**.
2. On the **General** tab, select the **Enable this project for Web Services Enhancements** check box, and then click **OK**.

Both the client and service require access to their respective X.509 certificates from the local certificate stores on the host computers. The client also requires access to its private key, and the service requires access to its private key. Also, the client must be able to access the service's X.509 certificate from its local certificate store. Typically, the certificate for a trusted service is installed in the **Trusted People** folder in the local certificate store. For more information about how to install X.509 certificates in the local machine certificate store, see the Certificates How To.

Note: You can use the WSE Certificates tool to view private key file properties and set access permissions for the account under which the client and service run.

For applications that use X.509 certificates, X.509 security must be configured for WSE 3.0.

► To configure WSE 3.0 X.509 security settings

1. In Visual Studio 2005, right-click the application project, and then click **WSE Settings 3.0**.
2. Click the **Security** tab, and then select the **allow test roots** check box if you are using a self-signed test certificate in a development or test environment. If you are configuring the application to run in a production environment, leave this check box cleared.
3. For **Revocation mode**, select the **Offline** option if you do not want to depend on accessing the certificate's revocation status online. If you select this option, you must be confident that you can update a local copy of the CRL in the local certificate store frequently enough to meet your requirements for certificate verification. If you want to allow the application to access the revocation status online, leave this option set at the default value **Online**, and then click **OK**.

After configuring the settings for X.509 certificate security with the WSE 3.0 Settings tool, they should appear in the application configuration file, as shown in the following XML example.

```
<configuration>
...
   <microsoft.web.services3>
      <security>
         <x509 verifyTrust="true" allowTestRoot="true" revocationMode="Offline"
verificationMode="TrustedPeopleOrChain"/>
      </security>
...
    </microsoft.web.services3>
...
</configuration>
```

Note: Usually, it is not necessary to modify this information directly because you can control these settings through the WSE 3.0 Settings tool.

The **allowTestRoot** attribute shown in the previous example determines whether the application allows test certificates. Test certificates are acceptable for development and test environments. However, for production environments you should only use certificates issued by a CA. This attribute is optional, and its value is **false** by default.

If you set the **verificationMode** attribute to **TrustedPeopleOrChain** to verify the signature on an incoming message, this setting requires that the message sender's X.509 certificate is located in the **Trusted People** folder of the verifying party's certificate store or that the certificate can be verified to a trusted CA through a certificate trust chain. For more information about configuring the behavior of X.509 security in WSE 3.0, see <x509> Element on MSDN.

Note: WSE 3.0 offers four different protection levels that determine how messages are secured using SOAP message security. Generally, you should use the **Sign, Encrypt, and Encrypt Signature** setting for best message protection. This setting encrypts the message body and the XML signature, which reduces the likelihood of a successful cryptographic guessing attack against the signature. For this reason, all the composite implementation patterns use this value as default. If you want to use this setting in new Web services you should change the **messageProtectionOrder** attribute to the following value in your security policy:

```
messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"
```

Configure the Client

After enabling the client application to support WSE 3.0 during General Setup, you must enable policy support. If your application does not currently have a policy cache file, you can add one for this purpose, and enable policy support by performing the following steps.

► **To add policy support to a WSE 3.0-enabled Visual Studio 2005 project**

1. In Visual Studio 2005, right-click the application project, and then click **WSE Settings 3.0**.
2. On the **Policy** tab, select the **Enable Policy** check box. Selecting this setting adds a policy cache file with the default name wse3policyCache.config.
3. Under **Edit Application Policy**, click **Add**, and then type a policy friendly name for the new application policy, such as "x509."
4. Click **OK** to start the WSE Security Settings Wizard, and then click **Next**.
5. On the **Authentication Settings** page, the wizard provides a choice to secure a service or a client. Select **secure a client application** to configure the client. The wizard also provides a choice of authentication methods in the same step. Select **Certificate**, and then click **Next**.
6. On the **Client Certificate** page, select the client certificate for the client. Unless your client application is impersonating a Windows user, select **LocalMachine** for the Store Location.
7. Click **Select Certificate** to select the appropriate X.509 certificate for the client application, click **OK**, and then click **Next**.
8. On the **Message Protection** page, the wizard displays configuration options for message protection. By default, the **Enable WS-Security 1.1 Extensions** check box is selected. Clear this check box to use the **mutualCertificate10Security** assertion. Leave it selected if you want to use the **mutualCertificate11Security** assertion. For **Protection Order**, select **Sign, Encrypt, Encrypt Signature**, and then click **Next**.
9. On the **Server Certificate** page, select **LocalMachine** for the Store Location, click **Select Certificate**, select the appropriate X.509 certificate for the service, click **OK**, and then click **Next**.
10. On the **Create Security Settings** page, review your settings, and then click **Finish**.

After you complete these tasks, your client security policy should look similar to the following code example.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="x509">
    <mutualCertificate10Security establishSecurityContext="true"
renewExpiredSecurityContext="true" requireSignatureConfirmation="false"
messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"
requireDerivedKeys="false" ttlInSeconds="300">
      <clientToken>
        <!-- WSE2 QuickStart Client Certificate -->
        <x509 storeLocation="LocalMachine" storeName="My"
findValue="CN=WSE2QuickStartClient" findType=" FindBySubjectDistinguishedName"/>
      </clientToken>
      <serviceToken>
        <!-- WSE2 QuickStart Server Certificate -->
        <x509 storeLocation="LocalMachine" storeName="My"
findValue="CN=WSE2QuickStartServer" findType="FindBySubjectDistinguishedName" />
      </serviceToken>
      <protection>
        <request signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <response signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="false" />
      </protection>
    </mutualCertificate10Security>
    <requireActionHeader/>
  </policy>
</policies>
```

In the previous policy example, the **signatureConfirmation** attribute is set to **false**. If this value is set to **true**, compatibility with WS-Security 1.0 is lost.

The **<clientToken>** element contains values that specify the client's X.509 certificate for signing outbound request messages and decrypting inbound response messages from the service. The information specific to the X.509 certificate is contained in the **<x509>** element. Set the **findValue** attribute to the value used to locate the certificate within the local certificate store. This will be the subject distinguished name for certificates retrieved from the certificate store using the client name or the text encoded binary value for a certificate identifier, such as the certificate's SHA1 thumbprint.

If you use the WSE 3.0 default configuration that retrieves certificates from the certificate store according to the subject distinguished name, you may risk confusing the identity of the client if different CAs issue different certificates with the same subject distinguished name. To avoid this, consider using a certificate identifier to retrieve certificates from the certificate store. For more information about how to set the **findType** and **findValue** attributes for the **<x509>** element, see <x509> Element (Policy) in the WSE 3.0 documentation.

Note: The value that WSE 3.0 is configured to use to find the certificate in the certificate store is not directly connected to the identifier that WSE 3.0 uses to identify certificates in transit when messages are sent. For example, by default, WSE 3.0 retrieves certificates from the certificate store by subject distinguished name, but the **mutualCertificate10Security** assertion uses either the RFC3280 subject key identifier or the issuer name and serial to identify the certificate in the request message.

In this example, the certificate and corresponding private key that the client uses belong to an application, not a user. If you want to authenticate a user instead of a client application, use a smart client application and ensure that the user's certificate and private key are accessible from the local certificate store on the workstation where the smart client application is installed. To ensure that the private key and corresponding certificate are only accessible to the user of the smart client application, set the **Store Location** to **CurrentUser** in step 6 of the previous procedure when configuring security policy on the client. Also, set the access control list (ACL) on the private key file so that only the user that owns the certificate can access it.

You can use the WSE 3.0 Certificates tool to obtain certificate information, such as the subject distinguished name or subject key identifier. The WSE 3.0 Certificates tool displays the subject distinguished name in reverse order, but this is actually the correct order for the client name in a WSE 3.0 policy assertion. You can also use the Microsoft Management Console (MMC) snap-in to obtain the subject distinguished name or certificate thumbprint, but as the subject distinguished name is not reversed, you must reverse it yourself to use it in a WSE 3.0 policy assertion. For example, you must reverse a subject distinguished name obtained from the MMC snap-in such as "CN=bob, DC=Microsoft, DC=com" when you specify it in policy to read as "DC=com, DC=Microsoft, CN=bob."

If you configure your application to retrieve certificates from the certificate store using the certificate's SHA1 thumbprint by setting the **findType** value of the **<x509>** attribute to **FindByThumbprint**, WSE 3.0 requires you to set the **findValue** attribute to the hexadecimal encoded value for the certificate thumbprint, not the Base64 encoded value. You must use the Certificates MMC snap-in to obtain this value, and remove the spaces between each byte value. For example, the first few bytes of a certificate thumbprint copied from the Certificates MMC snap-in would be formatted as: "c6 74 47 da..." Remove the spaces when pasting this information into the **findValue** attribute so that it displays as: "c67447da..." You cannot obtain this value using the WSE 3.0 Certificates tool.

The **<serviceToken>** element contains information about the service's certificate, which encrypts request messages and verifies the signature on response messages. The **<serviceToken>** settings can be configured similarly to those of the **<clientToken>** described previously in this section.

For more information about configuring other settings for this policy assertion, see <mutualCertificate10> Element in the WSE 3.0 documentation.

When you add a Web reference to the service from the client application to create a Web service proxy, two proxies are generated for the Web service: one is a nonWSE 3.0 proxy and the other is WSE 3.0–enabled. This pattern uses the WSE 3.0-enabled proxy class, which is name + "Wse." For example, if your Web service is named "MyService," your WSE 3.0–enabled Web service proxy class name would be "MyServiceWse."

The following code example demonstrates how to bind the policy assertion described previously for calling the Web service proxy. This example uses a WSE 3.0–enabled Web service proxy.

```
...
using System.Globalization;
...
try
{
   Service.ServiceWse service = new Service.ServiceWse();

   service.SetPolicy("x509");

   Service.Product product = service.GetProductInformation(txtProductName.Text);

   txtResults.Text = String.Format(CultureInfo.InvariantCulture,
                     "Product Name: {0}, Unit Prize: {1}, Quantity: {2}",
                     product.Name, product.UnitPrice, product.Quantity);

}
catch (Exception ex)
{
   txtResults.Text = ex.ToString();
}
...
```

Configure the Service

You must configure the service to enable SOAP extensions by performing the following steps.

► To enable a Visual Studio 2005 project to support SOAP extensions

1. In Visual Studio 2005, right-click the application project, and then click **WSE Settings 3.0**.
2. On the **General** tab, select the **Enable Microsoft Web Services Enhancement SOAP Protocol Factory** check box, and then click **OK**.

After you enable the service to support WSE 3.0, you also must enable policy support. If your application does not currently have a policy cache file, you can add one, and enable policy support by performing the following steps.

► **To add policy support to a WSE 3.0-enabled Visual Studio 2005 project**

1. In Visual Studio 2005, right-click the application project, and then click **WSE Settings 3.0.**
2. On the **Policy** tab, select the **Enable Policy** check box. Selecting this setting adds a policy cache file with the default name wse3policyCache.config.
3. Under **Edit Application Policy**, click **Add**, and then type a policy friendly name for the new application policy, such as "x509."
4. Click **OK** to start the WSE 3.0 Security Settings Wizard, and then click **Next**.
5. On the **Authentication Settings** page, the wizard provides a choice to secure a service or a client. Select **secure a service application** to configure the service. The wizard also provides a choice of authentication methods. Select **Certificate**, and then click **Next**.
6. On the **Authorized Clients** page, the wizard presents the option to perform authorization. If you want to add authorization to your service policy, select the **Perform Authorization** checkbox, click **Add** to add the X.509 certificates for the clients that you want to authorize to call the service, and then click **Next**.
7. On the **Message Protection** page, the wizard displays configuration options for message protection. By default, the **Enable WS-Security 1.1 Extensions** check box is selected. Clear this check box to use the **mutualCertificate10Security** assertion. Leave it selected if you want to use the **mutualCertificate11Security** assertion. For **Protection Order**, select the option for **Sign, Encrypt, Encrypt Signature**, and then click **Next**.

   Note: These settings must be the same on the client and the service.

8. On the **Server Certificate** page, click **Select Certificate** to select the appropriate X.509 certificate to use for the service, click **OK**, and then click **Next**.
9. On the **Create Security Settings** page, review your settings, and then click **Finish**.

After you complete these tasks, your service security policy should look similar to the following code example.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="x509">
    <authorization>
      <allow user="CN=WSE2QuickStartClient" />
      <deny user="*" />
    </authorization>
    <mutualCertificate10Security establishSecurityContext="true"
renewExpiredSecurityContext="true" requireSignatureConfirmation="false"
messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"
requireDerivedKeys="false" ttlInSeconds="300">
```

(continued)

(continued)

```
      <serviceToken>
        <!-- WSE2 QuickStart Server Certificate -->
        <x509 storeLocation="LocalMachine" storeName="My"
findValue="CN=WSE2QuickStartServer" findType="FindBySubjectDistinguishedName" />
      </serviceToken>
      <protection>
        <request signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <response signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="false" />
      </protection>
    </mutualCertificate10Security>
    <requireActionHeader />
  </policy>
</policies>
```

The service's policy file is similar to the client's policy file with the following exceptions:

- An **<authorization>** assertion limits which clients are allowed to access the service by specifying a comma-separated list of client names as they appear in the X.509 certificate's subject name attribute. The **<allow>** element should appear first with a list of authorized clients. If only specific clients are authorized to access the service, the **<deny>** element should always be specified immediately after the **<allow>** element with an asterisk (*) in its **user** attribute. This prevents access by all clients except those using X.509 certificates that are explicitly authorized in the **<allow>** element.
- The **<clientToken>** element is not specified because the service uses the X.509 security token attached to the request message to verify the signature on the request message, and to encrypt the response for transmission to the client.

The following code example demonstrates how to apply the policy provided previously when the service processes a request.

```
using System;
using System.Web.Services;

using Microsoft.Web.Services3;

using Microsoft.Practices.WSSP.WSE3.QuickStart.Common;

namespace Microsoft.Practices.WSSP.WSE3.QuickStart.MessageLayerX509.Service
{
    ///<summary>
    ///This class represents a Web service used to query products catalog.
    ///</summary>
```

(continued)

(continued)

```
    [WebService(Namespace =
"http://schemas.microsoft.com/WSSP/WSE3/QuickStart/BrokeredAuthentication/2005-
10/MessageLayerX509.wsdl")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [Policy("x509")]
    public class Service : System.Web.Services.WebService
    {
        ///<summary>
        ///Constructor
        ///</summary>
        public Service()
        {
        }

        ///<summary>
        ///Returns some information about the specified product.
        ///</summary>
        ///<param name="productName"></param>
        ///<returns></returns>
        [WebMethod]
        public Product GetProductInformation( string productName )
        {
            Product product = new Product();
            product.Name = productName;
            product.Quantity = 10;
            product.UnitPrice = 2.5M;
            return product;
        }
    }
}
```

In the previous code example, a reference to **Microsoft.Practices.WSSP.WSE3.QuickStart.Common** found in the WSSP QuickStarts code provides a reference to the **Product** class. Replace these as necessary for your application.

Resulting Context

This section describes some of the more significant benefits, liabilities, and security considerations of using this implementation pattern.

Note: The information in this section is not intended to be comprehensive. However, it does discuss many of the issues that are most commonly encountered for this pattern.

Benefits

The benefits of using the Implementing Message Layer Security with X.509 Certificates in WSE 3.0 pattern include the following:

- Authentication, data confidentiality, and data origin authentication using a single security mechanism.
- Authentication can occur over well-known Internet firewall friendly ports using well-known protocols (for example, HTTP/HTTPS over port 80/443).
- Authentication and message protection can occur across organizational boundaries and security domains, because you do not need to propagate the private key.

Liabilities

The liabilities associated with using the Implementing Message Layer Security with X.509 Certificates in WSE 3.0 pattern include the following:

- You need to store the private keys somewhere securely, such as on a smart card or a computer, which makes them less portable than passwords.
- The use of asymmetric cryptography is computationally intensive and may cause performance issues, even though WSE 3.0 optimizes asymmetric cryptography for performance. Most of the time, you can mitigate this issue by deploying servers with more processors or by adding more servers to a load balancing cluster.
- Signature verification using test certificates generated using the MakeCert utility can cause serious performance issues. You can use certificates issued by a CA to mitigate this issue. For more information about obtaining certificates, see the X.509 Technical Supplement in Chapter 7, "Technical Supplements."
- Implementing message layer security is likely to reduce the throughput and increase the latency of Web services, due to the overhead of the cryptographic operations that support canonicalization, XML signatures, and encryption. As part of your development process, you should identify performance objectives for your application and test the application against those objectives. For more information, see *Improving .NET Performance and Scalability*.

Security Considerations

Security considerations associated with using the Implementing Message Layer Security with X.509 Certificates in WSE 3.0 pattern include the following:

- Web services are susceptible to man-in-the-middle attacks, where an attacker could replace the signature and certificate information. To mitigate such an attack, the service must be able to limit the population of potential clients to a trusted group, either individually based on the client's X.509 certificate or as a group through a limited population of clients that are defined by a certificate trust chain. You can specify an **<authorization>** assertion in the service policy to restrict the clients that are allowed to access the service based on their subject distinguished name.

- When using X.509 certificates for authorization, WSE 3.0 only allows authorization to occur based on a certificate's subject distinguished name, not by certificate identifier. This creates the potential for confusion if there are different certificates issued by different CAs with the same subject distinguished name. If both CAs are trusted by the service, WSE 3.0 cannot distinguish between the certificates for authorization purposes. For this reason, it is especially important to verify certificate trust chains. A service that does not require trust chain verification could be exploited by an attacker creating a bogus certificate with the same subject distinguished name as an authorized certificate, and then using it to access the service. A potential solution to this problem is to extend the **X509SecurityTokenManger** released with WSE 3.0 to return a certificate identifier, such as the certificate's SHA1 thumbprint, instead of the subject distinguished name for authorization checks. A certificate identifier provides a more distinct way to identify the certificate than a subject distinguished name. For more information about this subject, see the next section.
- WSE 3.0 does not encrypt the client certificate that is attached to a request message. If you need to protect the identity of clients from disclosure to eavesdroppers, this introduces a potential information disclosure vulnerability. This is because certificates often contain information that eavesdroppers can use to identify the client.

Extensions

This section provides examples of how to extend the base pattern to provide additional security features.

Role-based Authorization

The lifetime of an X.509 certificate is typically greater than that of other security token types. As a result, it is difficult to provide security roles directly in an X.509 certificate. This would require you to use extensibility mechanisms to provide custom role information in the certificate. Because the role memberships of the certificate owner are likely to change before the certificate expires, the CA would need to issue a new certificate every time the certificate owner's roles change.

It is possible to establish a security context for a client that has successfully authenticated using a X.509 certificate by associating roles with its X.509 certificate. You can accomplish this by implementing a custom **X509SecurityTokenManager** on the service to construct a security principal, and then attach it to the security token. You can retrieve the roles for the security principal from a database, Active Directory, or another service that can provide roles for an identity to eliminate the need to provide them directly within the certificate.

The following code example provides an example of a custom **X509SecurityTokenManager**. After the **CustomX509SecurityTokenManager** authenticates the client, it constructs a **GenericPrincipal** with security roles and attaches it to the security token. You can copy this code and paste it into a new class file. However, you must provide code where indicated by comments to retrieve user roles from a database or other service provider, and change the namespace to suit your project.

```
using System;
using System.Xml;
using System.Security.Cryptography.Xml;
using System.Security.Principal;

using Microsoft.Web.Services3.Security;
using Microsoft.Web.Services3.Security.Tokens;

namespace Microsoft.Practices.WSSP.WSE3.QuickStart.MessageLayerX509.Service
{
    /// <summary>
    /// By implementing X509SecurityTokenManager we can manipulate the token
    /// on messages received.
    /// </summary>
    public class CustomX509SecurityTokenManager : X509SecurityTokenManager
    {
        /// <summary>
        /// Constructs an instance of this security token manager.
        /// </summary>
        public CustomX509SecurityTokenManager()
        {
        }

        /// <summary>
        /// Constructs an instance of this security token manager.
        /// </summary>
        /// <param name="nodes">An XmlNodeList containing XML elements from a
configuration file.</param>
        public CustomX509SecurityTokenManager(XmlNodeList nodes)
            : base(nodes)
        {
        }

        /// <summary>
        /// Adds a generic principal to the token
        /// </summary>
        /// <param name="token">The X509SecurityToken token</param>
        protected override void
AuthenticateToken(Microsoft.Web.Services3.Security.Tokens.X509SecurityToken token)
        {
            base.AuthenticateToken(token);

            // Assigns certificate's hexadecimally encoded SHA1 thumbprint to
GenericIdentity
            // Certificate's hexadecimally encoded SHA1 Thumbprint value can be
obtained using the Certificates MMC Snap-In
```

(continued)

(continued)

```
            string subjectKeyIdentifier = token.Certificate.Thumbprint;

            GenericIdentity identity = new GenericIdentity(subjectKeyIdentifier);
         //Replace the next line with your own code to retrieve roles from a role
store and populate the GenericPrincpal
            GenericPrincipal principal = new GenericPrincipal( identity, new
string[] {"role1, role2, role3"} );

            token.Principal = principal;
        }
    }
}
```

To use the previous example **CustomX509SecurityTokenManager**, you must create a security token manager entry in the service's Web.config file.

```
...
<microsoft.web.services3>
   ...
   <security>
   ...
      <binarySecurityTokenManager>
         <add type="
Microsoft.Practices.WSSP.WSE3.QuickStart.MessageLayerX509.Service.CustomX509Securi
tyTokenManager"
 valueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3"/>
      </binarySecurityTokenManager>
   </security
   ...
</microsoft.web.services3>
```

In the previous configuration example, you must modify the fully qualified class name for the custom security token manager to match the code for your **CustomX509SecurityTokenManager**.

With this extension to the base implementation, you can perform role-based authorization on the principal attached to the **X509SecurityToken** and avoid the limitation of authorization checks based solely on the identity represented in the certificate. You can perform authorization using policy or code. For an authorization example, see the service policy example under Configure the Service.

This extension also addresses the issue in the Security Considerations section of the base pattern about the ability to only perform identity-based authorization on the certificate's subject distinguished name, not based on a certificate identifier. While the default **X509SecurityTokenManager** adds the certificate's subject distinguished name to the **GenericIdentity**, the **CustomX509SecurityTokenManager** defined in this extension assigns the client certificate's hexadecimally encoded value of the SHA1 thumbprint to the security token identity instead of the certificate's subject distinguished name.

By assigning the client certificate's SHA1 thumbprint to the security token identity, you can use the hexadecimally encoded value of the client certificate's SHA1 thumbprint in the service's **<authorization>** assertion instead of the subject distinguished name. You can use the Certificates MMC Snap-In tool to obtain the hexadecimally encoded SHA1 thumbprint for a certificate. The following XML example provides an example of how to use the client certificate's SHA1 thumbprint in an **<authorization>** assertion in the service's policy cache.

```
...
<authorization>
      <allow user="ca7601381b4578502b62b8809825664f1e78dfa2" />
      <deny user="*" />
</authorization>
...
```

This code example mitigates the risk of confusing client identities by providing a way to identify client certificates that is more likely to be unique, as it performs authorization on the service when CAs issue different certificates with the same subject distinguished name.

Implementing Message Layer Security with a Security Token Service (STS) in WSE 3.0

Note: This pattern is currently under development. It is due for release in early 2006.

Context

You are implementing brokered authentication in an application deployed on computers running Windows operating system software with security implemented at the message layer. Web services need to authenticate clients in a heterogeneous environment so that you can implement additional controls, such as authorization and auditing. The authentication broker negotiates trust between client applications and Web services, which removes the need for a direct relationship. The authentication broker should issue signed security tokens for authentication.

Implementation Strategy

A QuickStart that demonstrates how to develop a Web Service Enhancements (WSE) 3.0 Security Token Service (STS) that issues XML tokens is currently under development. This pattern will be updated when the QuickStart is released.

If you are interested in obtaining a Community Technical Preview (CTP) release or would like to contribute requirements, join the Security Token Service Quickstart community workspace.