Implementing Message Layer Security with Kerberos in WSE 3.0

Context

You are implementing brokered authentication in an application deployed on computers running Windows with security implemented at the message layer. A Web service using Web Services Enhancements (WSE) 3.0 is processing messages from clients. The clients and services must use a standards-based security token that uses the organization's existing Active Directory infrastructure. The solution must be able to provide a complete set of security features, including data origin authentication and data confidentiality.

Objectives

The objectives of this pattern are to:

- Use an existing infrastructure that employs the Kerberos version 5 protocol at the message layer with a **KerberosToken** binary security token.
- Secure the communication channel to provide data confidentiality and data integrity by encrypting and signing messages with the **KerberosToken**.
- Impersonate authenticated clients that the **KerberosToken** represents to access resource on their behalf. A client can be a user, application, or server that needs to be authenticated before it can access a service.

Content

This pattern consists of the following sections:

- **Implementation strategy**. This section provides a high-level description of the strategy used to implement the solution that includes a description of the participants and the process.
- **Implementation approach**. This section describes the steps necessary to implement this pattern:
  - General setup
  - Client setup
  - Service setup
- **Resulting context**. This section outlines the benefits, liabilities, and security considerations related to this pattern.

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

Implementation Strategy

Use an existing Kerberos infrastructure, such as the one in Active Directory to provide authentication and access control on client workstations and servers that host Web applications. Use the **kerberosSecurity** policy assertion in WSE 3.0 to provide authentication, data confidentiality, and integrity at the message layer. For more information about WSE 3.0 policy, see Securing a Web Service on MSDN. This implementation also demonstrates how to use a **KerberosToken** to establish a Windows security context. The service then calls a second service that is configured for Windows Integrated Security.

Participants

Message layer security with the Kerberos protocol in WSE 3.0 involves the following participants:

- **Client**. The client accesses the Web service. The client provides the credentials for authentication during the request to the Web service.
- **Service**. The service is the Web service that requires authentication of a client prior to authorizing the client.
- **Key Distribution Center (KDC)**. The KDC is the authentication broker that authenticates clients and issues service tickets. On the Windows platform, the KDC is implemented in Active Directory.

Process

The "Process" section of Brokered Authentication: Kerberos in Chapter 1, "Authentication Patterns" describes how you can use a **KerberosToken** security token for message layer authentication with a Web service. The session keys created during Kerberos authentication also can sign and encrypt messages. These capabilities allow you to implement data origin authentication and data confidentiality as part of the authentication process. As a result, this pattern includes additional steps to represent a complete message layer security solution that implements authentication, data origin authentication, and data confidentiality.

Note: Windows 2000 does not support **KerberosToken** for signing and encryption. For more information about this and other information related to the Kerberos protocol, see Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

This pattern provides a more detailed description of the implementation process that the design pattern describes. The steps are divided into the following two parts, based on what happens on the client and then on the service:

- The client initializes a Kerberos security token and sends it in a message to a service.
- The service authenticates the client using information found in the security token.

Client: Initialize the Security Token and Send the Message

The client performs the following five steps to complete this task:

1. Request a service ticket.
2. Retrieve the service ticket.
3. Sign the message.
4. Encrypt the message.
5. Send the message to the service.

The steps are summarized in Figure 3.6.

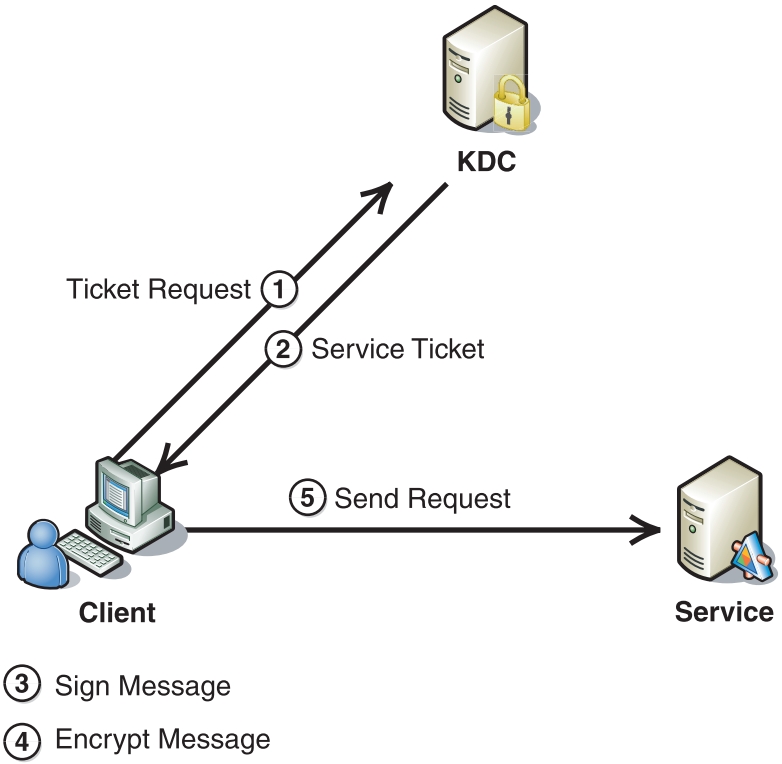

Figure 3.6
Initializing and sending a message using the Kerberos protocol

The following sections describe these steps.

Step One: Request a Service Ticket

The client interacts with the KDC to retrieve a service ticket, which it then uses to access the Web service. This action is actually performed as a request to the Security Support Provider Interface (SSPI) implementation in the Local Security Authority (LSA) to initialize a security context. The LSA accesses the client's ticket-granting ticket (TGT) and uses that to request a service ticket from the KDC. For more information about the Kerberos protocol implementation in Windows-based software, see Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

Step Two: Retrieve the Service Ticket

The ticket-granting service (TGS) creates a service ticket and returns it to the Local Security Authority (LSA). The LSA uses the service ticket to complete the initialization of a security context. WSE 3.0 uses the new security context to initialize a **KerberosToken** that is used to access the service. When a **KerberosToken** is initialized in WSE 3.0, two keys are derived from the session key in the service ticket for both the client and the service to use. One key is used to sign messages, and the other is used to encrypt them as described later in the process.

Step Three: Sign the Message

The message is signed using the security token retrieved in the previous step. You can choose to sign one or more portions of the message, such as the address header or the message body. An XML signature is created using a symmetric signature algorithm that computes a hash from the data to be signed using a signing key derived from the session key in the security token. When an XML signature is validated, the data used to create the signature is also validated to provide data origin authentication. At a minimum, you should include the addressing headers, timestamp, and message body in the message signature.

Step Four: Encrypt the Message

You can encrypt the message body using a security token that is derived from the session key in the security token. In addition, you should encrypt the message signature to reduce the risk of an offline cryptographic attack on the signature.

Step Five: Send the Message to the Service

After the client computer signs and encrypts the message, it sends the message to the service. When the message is sent, WSE 3.0 automatically adds the Kerberos security token to the message as a **BinarySecurityToken**.

Service: Authenticate the Client

There are five steps for the service to perform to complete this task:

1. Validate the token.
2. Decrypt the message.
3. Verify the XML signature.
4. Authorize and/or impersonate the client (optional).
5. Initialize and send a response to the client (optional).

The steps are summarized in Figure 3.7.

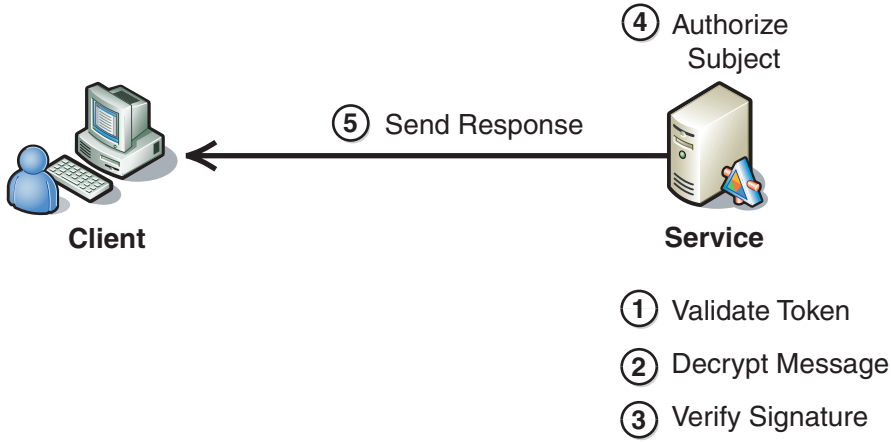

Figure 3.7
Authenticating a client using the Kerberos protocol

The following describes each of these steps in this section.

Step One: Validate the Token

When a service receives a message with a Kerberos security token, it needs to acquire credentials for the service account from the service host. To access the credentials, a service must be running under a process identity that has access to the service credentials. For more information about configuring the service identity with different operating systems, see Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

The service credentials contain the service's master key, which decrypts the service ticket in the message that the client sent. The service ticket contains a session key, which decrypts the authenticator and validates the message. For more information about Kerberos authenticators, see Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

After the service validates the message, it accepts the security token and uses the client's information in the service ticket to initialize a security context.

tep Two: Decrypt the Message

When WSE 3.0 receives a message that has been encrypted, WSE 3.0 policy does the following to decrypt the message:

1. Retrieve the symmetric session key from the service ticket.
2. Generate the derived encryption key from the session key.
3. Use the derived key to decrypt the message data with a symmetric algorithm.

Note: The policy on the server does not stop someone from sending an unencrypted message. However, it does reject a message at the server if it is not encrypted. The client can also implement a policy assertion that requires outbound messages to be encrypted.

Step Three: Verify the XML Signature

After the service receives the message. WSE 3.0 policy validates the message signature using the derived signing key that was sent with the message. This step validates the origin of that data to provide data origin authentication. However, note that XML signatures created using a symmetric algorithm do not support nonrepudiation. For more information about data origin authentication, see Data Origin Authentication in Chapter 2, "Message Protection Patterns."

Step Four: Authorize and/or Impersonate the Client (Optional)

By default, when WSE 3.0 receives a message that contains a Kerberos security token, it accesses the client's authorization claims that are contained in the service ticket. The Kerberos protocol allows these claims to perform authorization tasks, and the service ticket also can impersonate the client.

Step Five: Initialize and Send a Response to the Client Computer (Optional)

If the service returns a secure response to the client, the response must use the same security token that the request used. To accomplish this, the request message must be signed and encrypted using keys derived from the same token that was received in the request message from the client.

Implementation Approach

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

- General setup
- Client setup
- Service setup

Note: Applications using **KerberosTokens** will not function properly if they are hosted in Cassini. You must use Internet Information Services (IIS) 6.0 to host them. Cassini is a local access only Web server distributed with Visual Studio 2005 to allow Web development without IIS. One way to tell if a Web application is hosted in Cassini or in IIS is to look at the project in the Visual Studio 2005 solution explorer. If the Web application project appears as a file path (for example, C:\directory), Cassini is hosting the application. If the Web application project appears as an URL, IIS is hosting it.

General Setup

You must install the WSE 3.0 SDK on the computers that you use to develop WSE-enabled applications. After you have installed WSE 3.0, 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 Service Enhancements** check box, and then click **OK**.

You may be required to perform additional configuration steps to allow the ASP.NET process to access service credentials. You can use either of the following two approaches for this purpose. The approach to use depends on the Windows operating system that you used to install WSE 3.0:

- **Use the existing ASP.NET worker process**. This is the preferred option to use on computers running Windows Server 2003. No configuration is required in this case. The ASP.NET worker process uses a different account that has all of the necessary rights required to access service credentials.
- **Create a new domain account and map that account to the service host using setspn.exe**. This is the preferred option to use on computers running Windows XP and Windows 2000. To use this option, modify the Machine.config file and set the **userName** and **password** attributes to the new domain account in the **processModel** element, and then reset IIS 6.0. For more information about this option, see "Kerberos Operations for Web Services" in the Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

Caution: It is theoretically possible to configure the **processModel** element to use the SYSTEM account in a production environment. While this would give the ASP.NET process access to service credentials, using this account represents a serious security risk that could be catastrophic to your environment.

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"
```

Client Setup

This task requires the following steps to configure the client to implement this pattern:

1. **Configure the policy**. Includes the WSE 3.0 policy configuration settings necessary to implement this pattern on the client.
2. **Add the client code**. Includes the coding necessary to successfully implement this pattern on the client.

Configure the Policy

After enabling the client application to support WSE 3.0, you must enable policy support. If your application does not currently have a policy cache file, you can add one, and then perform the following procedure to enable policy support.

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

1. In Visual Studio, right-click the application project, and then click **WSE Settings 3.0**.
2. On the **Policy** tab, select the **Enable Policy** checkbox. Selecting this checkbox 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 "KerberosClient."
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 the option for **secure a client application** to configure the client.
6. The wizard also provides a choice of authentication methods in the same step. Select **Windows**, and then click **Next**.
7. On the **Kerberos Token** page, the wizard provides you with the option to provide a service principal name (SPN) and to specify the impersonation level for the Kerberos Token. The example for this pattern specifies the SPN as "http/server1." Replace "server1" with the name of the target Web server for the service that you will use. Then select **Impersonation** for the impersonation level and click **Next**.

8. On the **Message Protection** page, the wizard provides you with configuration options for message protection. You should select the option for **Sign, Encrypt, Encrypt Signature**. By default, the **Enable WS-Security 1.1 Extensions** setting is selected. Ensure that the extensions are enabled if you want WSE 3.0 to use signature confirmation to correlate a response message from the service with the request message that prompted it.
9. If your primary concern is interoperability, clear the **Enable WS-Security 1.1 Extensions** check box.

   – or –

   If you want to use signature confirmation, leave the **Enable WS-Security 1.1 Extensions** check box selected, click **Advanced Settings**, ensure the **Enable signature confirmation** check box is selected, click **OK**, and then click **Next**.
10. On the **Create Security Settings** page, review your settings, and then click **Finish**.

After you complete the procedure, your client security policy cache should appear similar to the following code example.

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.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="KerberosClient">
    <kerberosSecurity establishSecurityContext="true"
renewExpiredSecurityContext="true" requireSignatureConfirmation="false"
messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"
requireDerivedKeys="true" ttlInSeconds="300">
      <token>
        <!-- By default this sample does not work until you have changed the
TargetMachineName value -->
        <!-- Change the TargetMachineName value to the machine name with the Web
Service e.g. targetPrincipal="host/server1" -->
        <kerberos targetPrincipal="http/server1"
impersonationLevel="Impersonation" />
      </token>
      <protection>
        <request signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <response signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="false" />
      </protection>
    </kerberosSecurity>
    <requireActionHeader />
  </policy>
</policies>
```

The **kerberosSecurity** policy assertion provides the ability to sign and encrypt messages using policy with a **KerberosToken** binary security token in WSE 3.0.

In the previous example, the policy is declared as **KerberosClient**. It contains an assertion called **<kerberosSecurity>**. The **requireSignatureConfirmation** attribute controls whether the policy uses signature confirmation to provide a correlation between a response and the request that prompted it.

The **messageProtectionOrder** attribute defines the order in which the policy signs and encrypts the message. As recommended in the previous "Process" section for this pattern, when the client computer signs a message, ensure that the message signature also is encrypted. Setting the value of the **messageProtectionOrder** attribute to **SignBeforeEncryptAndEncryptSignature** will provide the recommended behavior. For more information about Kerberos assertion policy settings, see <kerberosSecurity> Element on MSDN.

The **<token>** section of the assertion provides details about the Kerberos token. You must set the **targetPrincipal** attribute to "http/," and include the name of the server where the Web service is hosted. If you have created a custom SPN, ensure that its name appears here. The **impersonationLevel** attribute allows you to specify whether you want the token to identify the client or impersonate the client. In this pattern this attribute is set to **Impersonation** so that a resource can be accessed on the client's behalf.

Note: You can use the prefix "host/" instead of "http/" for the SPN. However, doing this eliminates the option to use Windows Integrated Security on the target to access additional resources on behalf of the client computer. For more information about SPNs, see Kerberos Technical Supplement for Windows in Chapter 7, "Technical Supplements."

The **<protection>** section of the assertion allows you to specify protection options on the request, response, and fault messages using the **<request>**, **<response>**, and **<fault>** elements, respectively. The options available for each of these elements are the same. The **signatureOptions** attribute allows you to specify which parts of the message are signed in a comma separated list. To achieve the behavior recommended previously in the Process section for when the client signs the message, specify the **IncludeAddressing**, **IncludeTimestamp**, and **IncludeSoapBody** attributes in the value for the **signatureOptions** attribute. Setting the value of the **encryptBody** attribute to **true** encrypts request messages and decrypts response messages.

Add the Client Code

The following example code displays how to implement the client as a Web service client. It provides an example of binding the previously defined policy to the proxy, and then calling a Web service. You can copy the code to insert it into a new code module.

```
...
Service.ServiceWse service = new Service.ServiceWse();
service.SetPolicy("KerberosClient");

Product[] products = service.GetProductInformation(txtProductName.Text);

txtResults.Text = string.Empty;
foreach (Product product in products)
{
    txtResults.Text += String.Format( CultureInfo.InvariantCulture,
"Product Name: {0}, Unit Prize: {1}, Quantity: {2}",
        product.Name, product.UnitPrice, product.Quantity);
}
...
```

If the client is an ASP.NET application, you must configure your client application to use Windows authentication and set the **impersonate** attribute of the **<identity>** attribute to **true** in the client's Web.config file or programmatically impersonate the user in code. The following code example provides an example of how to configure security in the client's Web.config file.

```
<configuration>
...
   <system.web>
   ...
    <authentication mode="Windows"/>
    <identity impersonate="true"/>
   ...
   </system.web>
...
</configuration>
```

Smart client applications automatically attach the Windows security context of the user currently logged on to the workstation. For this reason, this configuration step is not necessary if the client is a smart client application.

Service Setup

This task consists of the following three steps to configure the client computer to implement this pattern:

- **Enable SOAP extensions**. Includes steps to enable the service application to support the WSE 3.0 SOAP Protocol Factory.
- **Configure the policy**. Includes the WSE 3.0 policy configuration settings necessary to implement this pattern on the service.
- **Use the service code**. Includes the coding necessary to successfully implement this pattern on the service.

Enable SOAP Extensions

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 select **WSE Settings 3.0**.
2. On the **General** tab, select the **Enable Microsoft Web Services Enhancement SOAP Protocol Factory** checkbox, and then click **OK**.

Configure the Policy

After enabling the service application to support WSE 3.0, you must enable policy support. If your application does not currently have a policy cache file, you can add one, and then perform the following procedure to enable policy support.

► **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 check box adds wse3policyCache.config as the default name for the policy cache file.
3. Under **Edit Application Policy**, click **Add**.
4. Type a policy friendly name for the new application policy, such as "KerberosService."

   Click **OK** to start the WSE Security Settings Wizard, and then click **Next**.
5. On the **Authentication Settings** page, the wizard provides you with the choice to secure a service or a client computer. Select the option to **secure a service application** to configure the client computer.
6. The wizard also provides you with a choice of authentication methods in the same step. Select the authentication method for **Windows**, and then click **Next**.

7. On the **Kerberos Token Claims** page, the wizard presents configuration authorization based on the user name or roles contained in the **KerberosToken**. By default, the **perform authorization** check box is cleared. If you want to perform authorization through the policy assertion, select the **perform authorization** check box, and then add users and roles as appropriate.
8. On the **Message Protection** page, the wizard presents configuration options for message protection. Microsoft recommends selecting the option for **Sign**, **Encrypt**, **Encrypt Signature**. By default, the **Enable WS-Security 1.1 Extensions** check box is selected. You must enable the extensions if you want WSE 3.0 to use signature confirmation to correlate a response message returned from the service with the request message that prompted it.
9. If your primary concern is interoperability, clear the **Enable WS-Security 1.1 Extensions** check box.

   – or –

   If you do want to use signature confirmation, leave the **Enable WS-Security 1.1 Extensions** check box selected, click **Advanced Settings**, ensure that the **Enable signature confirmation** check box also is selected, and then click **OK**. The message protection settings must be the same for both the client computer and the service.
10. On the **Create Security Settings** page, click **Next**, review your settings, and then click **Finish**.

After you complete the procedure, your service security policy should appear similar to the following code example.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="KerberosService">
    <authorization>
      <allow role="Users" />
      <deny role="*" />
    </authorization>
    <kerberosSecurity establishSecurityContext="true"
renewExpiredSecurityContext="true" requireSignatureConfirmation="false"
messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"
requireDerivedKeys="true" ttlInSeconds="300">
      <protection>
        <request signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <response signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true" />
        <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="false" />
      </protection>
    </kerberosSecurity>
    <requireActionHeader />
  </policy>
</policies>
```

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

- An **<authorization>** assertion is included to limit which clients are allowed or denied access to the service by specifying a comma-separated list of roles that the user belongs to that are defined in the Active Directory domain. The **<allow>** element should appear first, with a list of authorized roles. If only specific roles are authorized to access the service, the **<deny>** element should always be specified immediately following the **<allow>** element with an asterisk (*). In the previous policy file, the **<authorization>** assertion is configured to allow only users who belong to the "Users" role, and to deny all others.
- The **<token>** section is not included, as the service does not attach a **KerberosToken** to the request message when calling another service.

The service also requires an additional configuration update to support secure conversations. The service's policy file contains two attributes in the **<kerberosSecurity>** element named **establishSecurityContext** and **renewExpiredSecurityContext** that are used to enable secure conversation. By default, both of these attributes are set to **true**. However, to support secure conversation, the service must disable stateful security context tokens (SCTs). This is accomplished by adding the following configuration entry to the Web.config file of the service host.

```
<microsoft.web.services3>
   <tokenIssuer>
      <statefulSecurityContextToken enabled="false" />
   </tokenIssuer>
</microsoft.web.services3>
```

Use the Service Code

This step describes the code required to implement the service. The following code example displays how the service is implemented. You can copy the code to insert it into a new Web service class file.

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

using Microsoft.Web.Services3;

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

namespace
Microsoft.Practices.WSSP.WSE3.QuickStart.MessageLayerKerberos.SecondService
{
    /// <summary>
    /// This class represents a web service used to query products catalog
    /// </summary>
    [WebService(Namespace =
"http://schemas.microsoft.com/WSSP/WSE3/QuickStart/BrokeredAuthentication/2005-
10/MessageLayerKerberos/SecondService.wsdl")]
    public class Service : System.Web.Services.WebService
    {
        /// <summary>
        /// Constructor
        /// </summary>
        public Service()
        {
            InitializeComponent();
        }

        #region Component Designer generated code

        //Required by the Web Services Designer
        private System.ComponentModel.Container components = null;

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }
```

(continued)

(continued)

```
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion

        /// <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 = new Product();
            product.Name = productName;
            product.Quantity = 15;
            product.UnitPrice = 3.4M;
            return product;
        }
    }
}
```

In this code example, the service calls a second service that uses Windows Integrated Security. The service impersonates the client based on the received Kerberos token, delegated from the client to access the second Web service on behalf of the client. For more information about delegation, see Protocol Transition with Constrained Delegation Technical Supplement in Chapter 4, "Resource Access Patterns."

Resulting Context

This section describes some of the more significant benefits, liabilities, and security considerations of 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 Kerberos in WSE 3.0 pattern include the following:

- It provides single single-on capabilities that require a user to authenticate only once per session.
- It has broad acceptance as a brokered authentication protocol and the majority of large organizations that have centralized their authentication management infrastructure use the protocol.
- It is closely integrated with Windows (Windows 2000 or later). This enables the operating system to provide security capabilities such as impersonation, delegation, authorization, and auditing of the client.
- The Kerberos authentication process is more efficient than challenge/response. With the Kerberos protocol, authentication is performed by examining the security token sent in a request message. Challenge/response requires direct access to the authentication broker to authenticate a client.
- It supports mutual authentication when SPNs request a service ticket.
- It supports both signing and encryption of data in a Web service message.

Liabilities

The liabilities associated with the Implementing Message Layer Security with Kerberos in WSE 3.0 pattern include the following:

- The centralized nature of the Kerberos protocol requires a KDC to act as an authentication broker at all times. If the KDC fails, clients cannot establish new trust relationships with a service. Consider using redundant KDCs or providing an alternative mechanism, such as X.509 certificates for authentication. With Active Directory, you can improve KDC availability by establishing secondary domain controllers. This creates a redundant set of KDCs for the protocol to use.
- It is only useful for authentication and secure communication. In other words, the Kerberos protocol is not useful for securely persisting messages on a long-term basis because of the limited lifespan of tickets and session keys that it uses for encryption and signing.
- Proof that a client has authenticated cannot be established outside of the security domain where the client was authenticated, unless trust is explicitly established with the other security domain attempting to verify the client's security token.
- If you do not use signature confirmation, applications that use the **kerberosSecurity** policy are interoperable with applications implemented with the WS-Security 1.0 specification. If you do use signature confirmation, the application must support WS-Security 1.1.

- 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 the Implementing Message Layer Security with Kerberos in WSE 3.0 pattern include the following:

- When using Kerberos tokens at the message layer with Web services hosted on Windows 2000, the ASP.NET worker process requires higher privileges than it would normally possess.
- "Password guessing" attacks can occur against messages encrypted with a password equivalent (hash) that is derived from the user's password. The Kerberos protocol uses this derived key to encrypt data in the authentication request. An attacker could mount an offline dictionary attack by repeatedly attempting to decrypt the data in the authentication request sent to the KDC to discover the client's password.
- The Kerberos protocol does not implement authorization, although it is typically coupled with an authentication service that may store authorization information for a client. Resources can control access based on the client's authorization information, which is contained in the service ticket.

> **Note:** Active Directory provides authorization services that complement its Kerberos implementation.

- You cannot use the Kerberos protocol to facilitate nonrepudiation, because the client's identity secret is shared with the KDC.