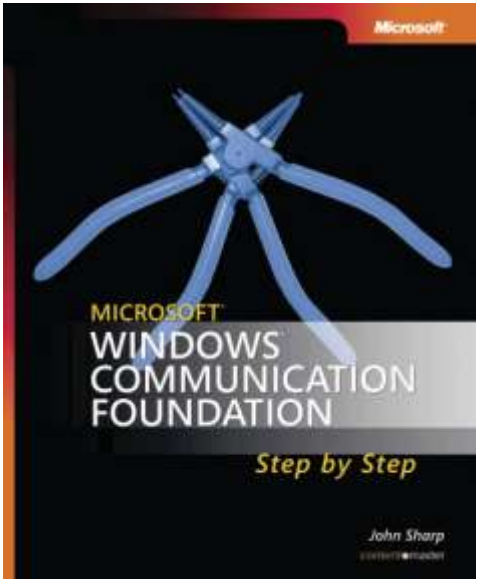


Microsoft® Windows® Communication Foundation Step by Step

John Sharp (Content Master)

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/10022.aspx

9780735623361
Publication Date: January 2007

Microsoft Press

Table of Contents

Chapter 4

Protecting an Enterprise WCF Service

After completing this chapter, you will be able to:

- Describe the different aspects of security that you should consider when implementing a WCF service.
- Explain how to provide privacy and integrity of messages at the message level and at the transport level when communicating between a client application and a WCF service.
- Explain how to configure a WCF service to authenticate users when running in a Windows environment and how a client application can provide a user's credentials to a WCF service for authentication.
- Describe how to define and use roles to authorize access to operations in a WCF service.
- Summarize how a WCF service can use impersonation to provide fine-grained access control over resources to authorized users.

Security is a fundamentally important aspect of any system, especially when a system comprises distributed applications and services. Security is also a very broad topic. For this reason, you are going to consider how to implement security in several different scenarios, spread across three chapters. This chapter concentrates on managing security within a single organization. In this environment, there is usually an inherent degree of trust between the computers running client applications and those hosting services. Users running applications are frequently members of the same, well-defined security domain. Services have access to the information in this security domain and can use it to authenticate users directly. In Chapter 5, "Protecting a WCF Service over the Internet," you will look at how to enforce security when client applications and services run in different security domains separated by an insecure network, where it is not possible, or even desirable, to directly authenticate users. In Chapter 15, "Managing Identity with Windows CardSpace," you will see how to implement an identity meta-system to help authenticate users in a federated environment.

What Is Security?

Security is concerned with protecting users running client applications, services, and the messages that pass between them. Security encompasses a range of issues. The most common aspects of security that most developers are familiar with include user authentication, where

a user attempts to prove their identity, and authorization, where a service decides which resources a user can access based on their identity. However, in a distributed environment, security has many other facets. These include:

- Maintaining confidentiality of communications between a client application and a service. It is possible for applications to eavesdrop on the data being transmitted across the network. For example, take a look at the number of software and hardware network analyzers available—many administrators use them for tracking connectivity and bandwidth problems in a network, but an unscrupulous user could also track the packets passing over the network for malicious purposes. The information in these packets could include private financial data or confidential personal information that should not be common knowledge even to other members of the same organization. Typically, you achieve confidentiality by encrypting messages.
- Preventing tampering or corruption of messages. In an environment where message confidentiality is assured, it is still possible for a malicious user to intercept messages and corrupt them before sending them to their final destination. You can use techniques such as message hashing to generate a digital signature for the file, which a service can use to help detect corrupt or modified messages.
- Ensuring verifiable delivery of messages. Even if a malicious user cannot decipher intercepted messages, the possibility of interception means that messages could either be diverted and not delivered at all or delivered repeatedly (known as a "replay attack"). Several schemes are available that can help detect replay attacks, including using a timestamp within a message (if the timestamp is outside reasonable limits when the service receives the message, it can discard it) and assigning unique identifiers to messages (if the service receives two messages with the same identifier, then it knows that there is a problem!). Similarly, using a reliable message protocol can help to ensure that messages are either delivered to the destination within a reasonable time or that the sender will be alerted if they are not. You will learn more about reliable messaging in Chapter 9, "Implementing Reliable Sessions."
- Preventing impersonation of services. Although not so common inside an enterprise as it is when using the Internet, it is possible for one service to impersonate another to obtain confidential data from a user. This phenomenon is sometimes known as "spoofing." The user running the client application thinks they are communicating with the real service but are actually sending their details and other information to an entirely different service that happens to respond in a similar manner. This means that it can be as important for a client application to authenticate the service and verify that it is genuine as it is for a service to authenticate the user running the client application. You will look at how you can implement this form of two-way authentication by using certificates in Chapter 5.

It is worth remembering that there is no such thing as absolute security. Hackers and fraudsters can invariably devise new and interesting ways to intercept, compromise, or otherwise disrupt the message flow. The important point is to be aware of the threats and have a plan for introduc-

ing countermeasures that can reduce their effects. Fortunately, WCF provides a highly extensible model that can adapt and evolve to meet many current security issues and (hopefully) counter new threats as they appear. The WCF implementation of security is also relatively unobtrusive. By careful design and configuration, you can separate many of the security-related aspects of a client application and service from the business logic, enabling you to modify or extend the security of your system without requiring that you rewrite large chunks of code.

Authentication and Authorization in a Windows Environment

To authenticate a user, a service must provide a means of enabling a user to identify herself and then prove that identity. Inside a single organization, it is common to maintain a single database of users and their means of identification. In a Windows environment, this typically means using *Active Directory*. In a single organization, it is not unreasonable to expect that all services and client applications have access to the same *Active Directory* database, and this database defines the security domain for the system. A service can be configured to use information held in the *Active Directory* database to authenticate users. When the user runs an application that accesses the service, the application can prompt the user for their username and password and transmit this information to the service. The service can query *Active Directory* to verify that the username is valid and the password is correct.

Note Many of the discussions on this chapter that refer to *Active Directory* also apply to Windows computers that are not actually part of a domain but that maintain their own local users and groups database. The exercises in this chapter have been tested on a stand-alone computer running Microsoft Windows XP and Windows Vista.

In a Windows domain, a service can also identify users by using the Kerberos protocol, and a WCF client application can verify the identity of a service by using the same protocol. However, Kerberos is only available if you have access to a Windows Server domain controller. This chapter does not describe how to configure a WCF service and client application to perform Kerberos authentication. For a brief summary of how Kerberos authentication works, see the Kerberos V5 Authentication page on the Microsoft Web site at *http://technet2.microsoft.com/WindowsServer/f/?en/library/d55683e8-1258-4555-93cb-77138d33beab1033.mspx*.

This approach works regardless of where the user is actually running the client application; it could be executing on a computer in the user's bedroom connecting to the service across an Intranet link, for example. However, a user located in the office might already be logged on to an organization's security domain, so prompting them for the username and password again becomes cumbersome (why should they need to keep on logging in?). Fortunately, the Windows operating system provides support for this very common scenario. When a user successfully logs in to a security domain, the details of the user's credentials are cached in the user's login process. When the user runs an application that requires authentication with a service, Windows can provide these details to the application, which can then forward them to the service. This mechanism is known as Windows Integrated Security.

Note In a very large organization, the security domain might span several *Active Directory* databases managed independently by administrators in different parts of the organization. It is possible to configure trust relationships between these separate domains, effectively presenting them as a single security domain.

After a service has verified the identity of the user running the client application, it must then determine whether the user has the appropriate authority to invoke the specified operations. Typically, administrators assign users to roles, and the service developer can indicate which roles are allowed to access which operations. WCF can utilize .NET Framework declarative security to associate roles with operations. WCF can use a *role provider* to determine to which roles a user belongs. The .NET Framework provides three role providers that you can use for storing role information. These role providers are:

- Windows Token Role Provider, which uses roles based on Active Directory groups.
- SQL Role Provider, which uses roles stored in a SQL Server database.
- Authorization Store Role Provider, which uses roles defined by using the Microsoft Authorization Manager tool. This tool enables you to store role information in Active Directory or in XML files.

More Info For detailed information on using Microsoft Authorization Manager to define and implement roles, see the Authorization Manager page on the Microsoft Web site at *http://technet2.microsoft.com/WindowsServer/en/library/1b4de9c6-4df9-4b5a-83e9-fb8d497723781033.mspx?mfr=true*.

In this chapter, you will use the Windows Token Role Provider. This provider is ideal for use inside an enterprise that uses Windows Integrated Security for authentication. In Chapter 5, you will see how to use the SQL Role Provider as this is more suited to Internet-based services.

Transport and Message Level Security

User identity information has to be transported from a client application to a service. This information is critical, and so it should be transmitted in as secure a manner as possible. This normally means encrypting these details. Additionally, after the user has been authenticated, the contents of messages passing between the client application and service might also require some form of encryption, depending on the sensitivity of the information in these messages. There are many ways that client applications and services can achieve this aim, but the important point is that the client applications and the service must agree on the mechanism that they use, and they must be able to decrypt messages sent by the other. Various standardization efforts have led to the use of public/private key cryptography being used to this effect.

More Info For a good introduction to public key cryptography, visit the Understanding Public Key Cryptography page on the Microsoft Web site at *http://www.microsoft.com/technet/prodtechnol/exchange/guides/E2k3MsgSecGuide/6e75927b-bec3-475b-bf09-764c8ffc7027.mspx?mfr=true*.

When building Web services, you can perform authentication and encryption at two points when sending and receiving messages: at the transport level and at the message level.

Transport Level Security

Transport level authentication is typically implemented at the operating system level before the application or service receiving the message even knows that there is a message to receive! A service can specify the type of credentials it requires, but it is the operating system's responsibility to ensure that the correct credentials are provided and to validate them.

Many communications protocols can encrypt and decrypt data as it is sent and received. The most common example of such a protocol is HTTPS, which uses a technology called the Secure Sockets Layer (SSL) to encrypt and decrypt data by using keys provided in certificates. When a client application connects to a service by using the HTTPS protocol, the underlying transport infrastructure for the client application and service can negotiate over the degree of encryption to perform and exchange a certificate containing keys that they can use to encrypt and decrypt messages. Because all of this happens at the transport level, it is transparent to the client application and service; all they have to do is specify that they will communicate using the HTTPS protocol. However, an administrator has to install and configure the appropriate certificates for the service host application. Unsurprisingly, you can also use transport level security with the TCP protocol (SSL is itself based on TCP). Named pipes also support transport level security.

More Info In this chapter, you will configure HTTPS for use with a self-hosted WCF service. If you are hosting a WCF service in IIS, the configuration process is a little different. You will learn more about configuring HTTPS with IIS in Chapter 5.

Message Level Security

Authentication at the message level is the responsibility of the service. The credentials of the user are included in messages sent to the service, and the service has to verify that they are valid. Additionally, message level privacy and integrity is also the responsibility of the client application and service—they encrypt and decrypt messages themselves using an agreed encryption algorithm and a negotiated set of encryption keys. Standards such as the WS-Security specification from OASIS describe the message level security schemes that many Web services implementations have adopted, and by following the recommendations of WS-Security you can help to ensure the interoperability of your client applications and services with those developed by using technologies other than WCF.

Transport level security has the advantage over message level security that it can often rely on hardware support and can be very efficient—encrypting and decrypting data can be a resource-intensive process, so anything that improves performance is very welcome. Additionally, transport level authentication checks are enforced before the client application actually starts sending application level messages, so performing authentication at this level detects authentication failures more quickly and with less network overhead. The primary disadvantage of transport level security is that it operates on a point-to-point basis; by the time the service receives a message, it has already been decrypted by the underlying transport mechanism. In a situation where a service should simply forward a message on to another service rather than process it, it has full access to the message contents. The service could modify the message or extract confidential information before forwarding it. Using message level encryption can help to mitigate this problem. Message level security provides end-to-end encryption. A client application and the service acting as the final destination can agree on an encryption key and an encryption algorithm to use for messages. When a message arrives at the intermediate service, it is still encrypted. If the intermediate service does not have access to the encryption key or has no knowledge of the selected encryption algorithm, it cannot easily decrypt the message.

Implementing message level security sounds like it could add quite a lot of work to the development effort required for building a service. However, WCF greatly simplifies matters and reduces the development effort required by incorporating much of the code required as part of the standard bindings you can specify when configuring an endpoint for a service. All you need to do is set the properties of your selected binding appropriately (you will see several examples throughout this chapter).

Implementing Security in a Windows Domain

In the following exercises, you will see how to use transport and message level security in some common scenarios that can arise within a single organization. Because it is easier to demonstrate and explain things this way around, you will start by learning how to implement message confidentiality by encrypting messages. You will then see how to authenticate users running in a Windows environment, and finally, how to use the Windows Token Role provider to authorize access to operations.

Protecting a TCP Service at the Message Level

Message encryption is a very common requirement of most distributed systems; so much so that the majority of the standard bindings available in the WCF library encrypt messages by default. For example, the NetTcpBinding binding automatically encrypts data at the transport level if you have configured SSL over TCP. The NetTcpBinding binding also supports encryption at the message level, giving you a greater degree of control over the encryption algorithm used and without requiring you to configure SSL. You will use message level security to implement message encryption in the first exercise.

Enable message level encryption for the NetTcpBinding binding for the WCF service

1. Using Visual Studio 2005, open the solution file ProductsService.sln located in the Microsoft Press\WCF Step By Step\Chapter 4\ProductsService folder under your \My Documents folder.

   This solution contains three projects: the ProductsService service, the ProductsServiceHost application, and the ProductsClient. These projects are configured to catch and handle SOAP faults, as described in Chapter 3, "Making Applications and Services Robust."

2. Expand the ProductsServiceHost project in Solution Explorer, right-click the App.config file, and then click Edit WCF Configuration.

3. In the WCF Service Configuration Editor, right-click the Bindings folder and then click New Binding Configuration.

4. In the Create a New Binding dialog box, select the netTcpBinding binding type and then click OK.

   The WCF Service Configuration Editor generates a binding configuration with the default settings for the NetTcpBinding binding.

5. In the right pane of the WCF Service Configuration Editor, change the *Name* property of the binding to *ProductsServiceTcpBindingConfig*.

6. Click the Security tab.

7. Change the *Mode* property to *Message*. Change the *AlgorithmSuite* property to *Basic128*. Leave the *MessageClientCredentialType* property set to Windows.

   These settings cause the binding to use message level security. Users will be expected to provide a valid Windows username and password, and all messages will be encrypted by using the Advanced Encryption Standard (AES) 128-bit algorithm. This is a widely used algorithm that is relatively quick to perform but should provide sufficient privacy for messages inside an organization (if you are sending messages across a public wide area network such as the Internet, you might prefer to use Basic256, which is the default value).

Note If you set the Mode to None, then the binding will not encrypt data and any settings you specify for transport or message level security will be ignored. The Transport mode selects transport level security (SSL) rather than message level security, and the TransportWithMessageCredential mode uses message level security to provide the identity of the user for authorization purposes, while performing encryption at the transport level. Transport level encryption is usually more efficient than message level encryption, although it requires more configuration on the part of the administrator.

8. In the left pane of the WCF Service Configuration Editor, expand the Products.ProductsServiceImpl service in the Services folder, expand the Endpoints folder, and then click the ProductsServiceTcpBinding endpoint.

9. In the right pane, set the *BindingConfiguration* property to *ProductsServiceTcpBindingConfig*.

   This action associates the binding configuration with the binding. All messages sent by using the ProductsServiceTcpBinding will use message level security and will be encrypted.

10. Save the configuration, and then exit the WCF Service Configuration Editor.

11. In Visual Studio 2005, open the file App.config in the ProductsServiceHost project. In the <system.serviceModel> section, you should see the new binding configuration, and the reference to this configuration in the ProductsServiceTcpBinding endpoint, as follows:

```
…
<system.serviceModel>
  <bindings>
    <netTcpBinding>
      <binding name="ProductsServiceTcpBindingConfig">
        <security mode="Message">
          <message algorithmSuite="Basic128" />
        </security>
      </binding>
    </netTcpBinding>
  </bindings>
  <services>
    <service behaviorConfiguration="ProductsBehavior"
             name="Products.ProductsServiceImpl">
      …
      <endpoint binding="netTcpBinding"
         bindingConfiguration="ProductsServiceTcpBindingConfig"
        name="ProductsServiceTcpBinding" contract="Products.IProductsService" />
    </service>
  </services>
  …
</system.serviceModel>
```

   Be careful not to change anything in this file. Close the App.config file when you have finished examining it.

The service will expect clients that connect to the endpoint for this binding to use the same message level security settings. You will configure the client next.

Enable message level encryption for the NetTcpBinding binding for the WCF client

1. In the ProductsClient project, edit the app.config file by using the WCF Service Configuration Editor.

2. In the WCF Service Configuration Editor, right-click the Bindings folder and then click New Binding Configuration.

> **Note** The client configuration file already contains a binding configuration for the basicHttpBinding that was generated in Chapter 1, "Introducing Windows Communication Foundation." Be careful not to modify this binding configuration by mistake!

3. In the Create a New Binding dialog box, select the netTcpBinding binding type and then click OK.
4. In the right pane of the WCF Service Configuration Editor, change the *Name* property of the binding to *ProductsClientTcpBindingConfig*.
5. Click the Security tab.
6. Change the Mode property to Message. Change the *AlgorithmSuite* property to *Basic128*. Leave the MessageClientCredentialType property set to *Windows*.

> **Note** If you select a different algorithm suite for the client and server, they will not be able to decipher each other's communications. This will result in a runtime exception in the channel stack. If you are curious about this, try setting the *AlgorithmSuite* to *TripleDes* (for example) and examine the exception that occurs when you run the solution later.

7. In the left pane of the WCF Service Configuration Editor, click the NetTcpBinding_IProductsService node in the Endpoints folder, under the Client folder.
8. In the right pane, set the BindingConfiguration property to *ProductsClientTcpBinding Config*.
9. Save the configuration, and then exit the WCF Service Configuration Editor.
10. Start the solution without debugging.
11. In the ProductsServiceHost form, click Start. If a Windows Security Alert dialog box appears, click Unblock to allow the service to access the TCP port.
12. In the client console window, press Enter. Verify that the client application runs exactly as before.
13. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

This exercise has shown you how easy it is to configure a WCF service and client application to secure messages by performing encryption, but how do you actually *know* that the messages have been encrypted? To answer this question, you can enable message tracing and then examine the messages as they flow in and out of the service.

Configure message tracing for the WCF service

1. In Visual Studio 2005, edit the App.config file for the ProductsServiceHost project by using the WCF Service Configuration Editor.
2. In the WCF Service Configuration Editor, expand the Diagnostics folder and then click Message Logging.
3. In the right pane displaying the message logging settings, set the following properties to *True*:
   - LogEntireMessage
   - *LogMessagesAtServiceLevel*
   - *LogMessagesAtTransportLevel*

   The *LogEntireMessage* property specifies whether the trace output should include the body of messages sent and received. Setting this property to *True* includes the body of the message. The default value, *False*, only traces the message header. Setting the *LogMessagesAtServiceLevel* property to *True* traces messages as they are presented to the service and as they are output from the service. If you are using message level security, this trace will show the unencrypted messages after they have been received and decrypted at the message level (for incoming messages) or before they are encrypted (for outgoing messages). Setting the *LogMessagesAtTransportLevel* property to *True* traces messages as they are sent to or received from the transport level. If you are using message level security, the messages traced at this point will be encrypted, although if you are using transport level security messages will already have been decrypted (for incoming messages) or not yet encrypted (for outgoing messages) at this point.

> **Important** Tracing at the message level records messages in their unencrypted form. You should ensure that you protect the trace files that are generated and only let authorized users examine this data.

4. In the left pane, right-click the Sources folder and then click New Source.

   All tracing information for WCF is received from one or more trace sources. In this case, you will use the MessageLogging source, which traces messages. You can also use other sources. For example, the ServiceModel source traces events that occur in a service, such as tracking when a service starts listening, receives requests, and sends responses.
5. In the right pane, set the *Name* property to System.*ServiceModel.MessageLogging*. Set the *Trace level* property to *Verbose*.
6. In the left pane, right-click the Listeners folder, and then click New Listener.

   A listener object is responsible for receiving data from the trace sources, formatting and filtering them, and then sending them to a destination.
7. In the right pane, set the *Name* property to *MessageLog*.

8. In the *InitData* property, click the ellipses button. In the Save Log As dialog box, move to the Microsoft Press\WCF Step By Step\Chapter 4 folder under your \My Documents folder. Set the file name to Products.svclog, and then click Save.

   The *InitData* property specifies the name of the file that the listener will use for saving trace data. When tracing starts, if this file does not exist, the listener will create it; otherwise, it will append trace information to the end of any existing data in the file.

9. In the *TraceOutputOptions* property, click the dropdown arrow. Clear all items in the list. The trace output options are useful if you are tracing messages for multiple client applications and you need to be able to correlate the different request and response messages. In this example, you will be running a single client application, so this additional information is not really necessary.

10. Verify that the *TypeName* property is set to *System.Diagnostics.XmlWriter.TraceListener*. The listener can output data in several formats. However, you will be using another tool called the Service Trace Viewer to examine the trace output, and this tool expects the data to be in XML format.

11. Click Add at the bottom of the right pane. In the Add Tracing Source dialog box, select the System.ServiceModel.MessageLogging source, and then click OK.

12. Save the configuration, and then exit the WCF Service Configuration Editor.

Run the WCF client and service and examine the trace output

1. Start the solution without debugging.
2. In the ProductsServiceHost form, click Start.
3. In the client console window, press Enter. Verify that the client application still runs correctly.
4. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.
5. On the Windows Start menu, point to All Programs, point to Microsoft Windows SDK, point to Tools, and then click Service Trace Viewer.
6. In the Service Trace Viewer, on the File menu, click Add.
7. In the Open dialog box, move to the Microsoft Press\WCF Step By Step\Chapter 4\ProductsService folder under your \My Documents folder, select the file Products.svclog, and then click Open.
8. In the Service Trace Viewer, in the left pane, click the Message tab. You will see a list of messages sent and received by the service, identified by their *Action* values.

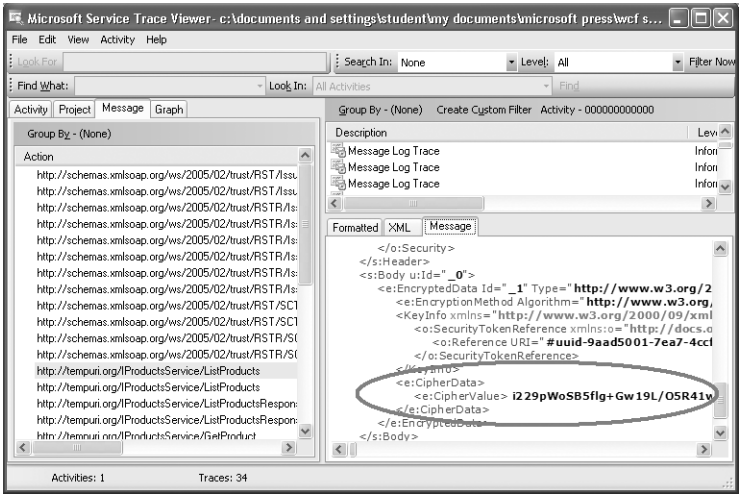
> **Tip** Expand the *Action* column in this pane to see more of the name for each action.

At the top of this list are a number of messages in the *http://schemas.xmlsoap.org/ws/2005/02/trust* namespace. These messages are concerned with sending and verifying

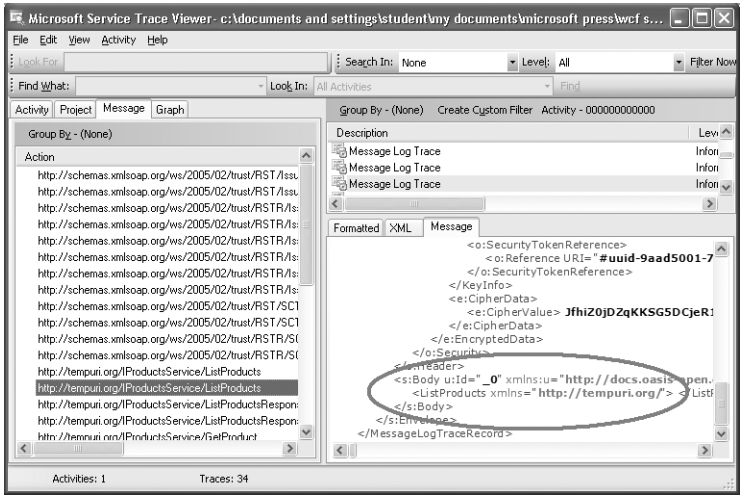
the user's identity, and negotiating the encryption mechanism and encryption keys that the client application and WCF service will use for sending and receiving messages. These messages are followed by the application messages received and sent by the WCF service, identified by the *http://tempuri.org* namespace.

9. Click the first message with the action *http://tempuri.org/IProductsService/ListProducts*. Note that each action occurs twice. This is because you traced each message twice: once at the message level and once at the transport level.

10. In the lower right pane, click the Message tab. The window will display the entire SOAP message. This is the version of the message passed from the transport level to the message level. The message has a rather lengthy SOAP header, which you can examine at your leisure. The interesting part is the SOAP body, at the end of the message. This is the encrypted ListProducts request received from the client application. The *<e:CipherValue>* element contains the data for the request, as highlighted in the following image:



11. In the left pane, click the second message with the action *http://tempuri.org/IProductsService/ListProducts*. In the right pane, scroll to the end of the Message window. This is the unencrypted version of the message passed from the message level to the service:



12. In the left pane, click the first message with the action *http://tempuri.org/IProductsService/ListProductsResponse*. In the right pane, examine the message body in the Message window. You can see that this is an unencrypted message containing the list of products returned in response to the ListProducts request. This message is the output from the service to the message level and so has not yet been encrypted.

13. In the left pane, click the second message with the action *http://tempuri.org/IProductsService/ListProductsResponse*. In the right pane, scroll to the bottom of the Message window and examine the message body. This time you can see that this is the encrypted response sent by the message level to the transport level for transmission back to the client.

14. Examine the other messages. When you have finished, close the Service Trace Viewer.

Protecting an HTTP Service at the Transport Level

If you recall, the ProductsServiceHost application exposes two endpoints for clients to connect to: one based on the TCP protocol and the other using HTTP. The HTTP endpoint is configured to use the BasicHttpBinding binding. The BasicHttpBinding binding conforms to the WS-BasicProfile 1.1 specification and is intended for use with existing legacy Web services and clients. It is fully interoperable with ASP.NET Web services. By default, this binding provides minimal security; it does not support message level encryption or authentication, for example. To implement message confidentiality and remain interoperable with ASP.NET Web services, you should use transport level security. This requires you to configure HTTPS.

Note The BasicHttpBinding binding also supports message level security. Ordinary ASP.NET Web services and client applications do not implement the WS-Security specification, and so will not be able to communicate with a service that implements message level security. However, Microsoft Web Services Enhancements (WSE) does support WS-Security,

so Web services that you create by using WSE can communicate with a WCF service through an endpoint based on the BasicHttpBinding binding by using message level security.

Specify transport level security for the BasicHttpBinding binding for the WCF service

1. In Visual Studio 2005, in the ProductsServiceHost project in Solution Explorer, edit the App.config file by using the WCF Service Configuration Editor.
2. In the WCF Service Configuration Editor, right-click the Bindings folder and then click New Binding Configuration.
3. In the Create a New Binding dialog box, select the basicHttpBinding binding type and then click OK.
4. In the right pane of the WCF Service Configuration Editor, change the *Name* property of the binding to *ProductsServiceBasicHttpBindingConfig*.
5. Click the Security tab. Set the Mode to Transport.

   In this mode, message security is provided by using HTTPS. You must configure SSL for the service by using a certificate. The client authenticates the service by using the service's SSL certificate. The service authenticates the client by using the mechanism specified by the *TransportClientCredentialType* property. The default value of *None* does not provide any authentication—you will examine some of the other values you can specify for this property later in this chapter.
6. In the left pane of the WCF Service Configuration Editor, expand the ProductsServicesImpl service in the Services folder, expand the Endpoints folder, and then click the ProductsServiceHttpEndpoint endpoint.
7. In the right pane, set the *BindingConfiguration* property to *ProductsServiceBasicHttpBindingConfig*.
8. HTTP Web services that implement transport level security *must* specify the https scheme, so change the *Address* property as follows:

   ```
   https://localhost:8000/ProductsService/ProductsService.svc
   ```
9. Save the configuration, and exit the WCF Service Configuration Editor.
10. Rebuild the ProductsServiceHost project.

The next step is to reconfigure and modify the client to connect to the service by using the endpoint corresponding to the BasicHttpBinding binding.

Specify transport level security for the BasicHttpBinding binding for the WCF client

1. In the ProductsClient project, edit the app.config file by using the WCF Service Configuration Editor.

2. In the WCF Service Configuration Editor, expand the Bindings folder and then click the BasicHttpBinding_IProductsService binding.
3. In the right pane of the WCF Service Configuration Editor, change the *Name* property of the binding to *ProductsClientBasicHttpBindingConfig*. (This is to make the name of the binding consistent with the other bindings you have created. The original binding name was generated by the svcutil utility back in Chapter 1.)
4. Click the Security tab. Change the Mode to Transport.
5. In the left pane of the WCF Service Configuration Editor, click the BasicHttpBinding_IProductsService endpoint in the Endpoints folder, under the Client folder.
6. In the right pane, change the address to use the https scheme as shown below, and verify that the *BindingConfiguration* property has changed to *ProductsClientBasicHttpBinding-Config*:

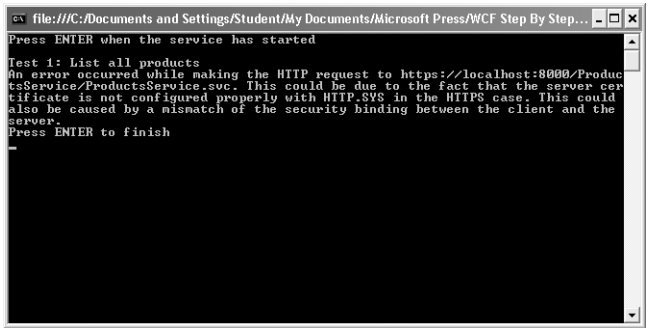
   ```
   https://localhost:8000/ProductsService/ProductsService.svc
   ```

7. Save the configuration, and then exit the WCF Service Configuration Editor.
8. In Visual Studio 2005, in Solution Explorer, open the Program.cs file for the ProductsClient project.
9. In the Main method, update the statement that creates the proxy object to connect to the WCF service by using the endpoint named BasicHttpBinding_IProductsService:

   ```
   ProductsServiceClient proxy = new ProductsServiceClient("BasicHttpBinding_
   IProductsService");
   ```

10. Rebuild the ProductsClient project.

    If you try and run the client and service at this point, the client will fail with a CommunicationException, like this:



This error occurs because you have not yet configured transport security for the HTTPS protocol. In the next exercise, you will create a certificate for the WCF service, and configure SSL for the service by using the httpcfg utility.

Configure the WCF HTTP endpoint with an SSL certificate

1. On the Windows Start menu, point to All Programs, point to Microsoft Windows SDK, and then click CMD Shell.

   A command prompt window opens, with an environment configured for running the Windows SDK tools.

2. In the command prompt window, type the following command:

   ```
   makecert -sr LocalMachine -ss My -n CN=HTTPS-Server -sky exchange -sk HTTPS-Key
   ```

   The makecert utility is a useful tool for creating test certificates that you can use for development purposes. The command shown here creates a certificate that is stored in the Personal certificates store for the LocalMachine account. For detailed information about the options for the makecert utility, see the Windows SDK Documentation installed with the Windows SDK.

> **Important** Certificates that you create by using the makecert utility should not be used in a production environment as they are not certified by a verifiable certification authority. Remember that the service sends this certificate to the client to prove its identity. The client must be able to trust that this certificate was created by a reliable source that can verify the veracity of the service. When deploying a production service, you should obtain your certificates from recognized certification authority, such as VeriSign or Thawte. Alternatively, you can use Windows Certificate Services, which enables an enterprise to generate its own certificates.

   To use the httpcfg utility to configure SSL for the service, you need to find the thumbprint of the certificate. The thumbprint is a hexadecimal string that uniquely identifies the certificate. You can obtain this information by using the Certificates Microsoft Management Console snap-in.

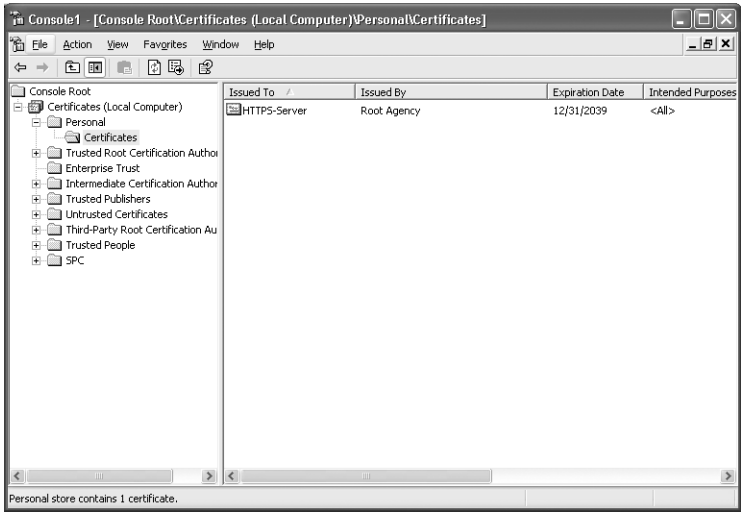
3. In the command prompt window, type the following command:

   ```
   mmc
   ```

   This command starts the Microsoft Management Console, displaying the default Console Root window.

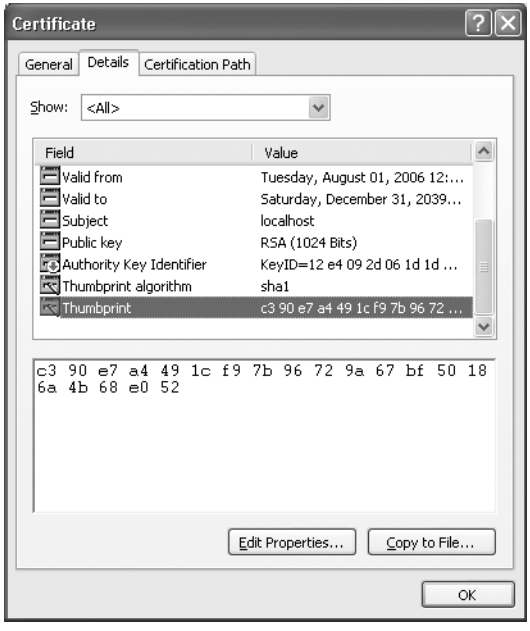
4. In the File menu, click Add/Remove Snap-In.
5. In the Add/Remove Snap-In dialog box, click Add.
6. In the Add Standalone Snap-In dialog box, select the Certificates snap-in and then click Add.
7. In the Certificates Snap-In dialog box, select Computer account and then click Next.
8. In the Select Computer dialog box, select Local computer and then click Finish.
9. In the Add Standalone Snap-In dialog box, click Close.
10. In the Add/Remove Snap-In dialog box, click OK.

11. In the Console Root window, expand the Certificates node, expand the Personal folder, and then click the Certificates folder. The HTTPS-Server certificate that you created by using the makecert utility should be displayed:



12. Double-click the HTTPS-Server certificate.

13. In the Certificate window, click the Details tab. Scroll to the bottom of the window displaying the details of the certificate. Click the *Thumbprint* property, and make a note of the hexadecimal string displayed in the lower window:



Tip You might find it useful to simply select the text in the lower window and copy it to the Windows clipboard.

14. Click OK, close the Microsoft Management Console window, and return to the command prompt window.

15. In the command prompt window, type the command shown below. Replace the hexadecimal string following the *–h* flag with the digits from the certificate thumbprint (remove all spaces from the thumbprint string first):

```
httpcfg set ssl –i 0.0.0.0:8000 –h c390e7a4491cf97b96729167bf50186a4b68e052
```

If this command is successful, it should report the message "HttpSetServiceConfiguration completed with 0."

Note Be very careful to specify the correct thumbprint. If you type an invalid thumbprint, the command still succeeds, but the client will not be able to communicate with the service as the thumbprint does not refer to a valid certificate.

This command binds the certificate with the thumbprint indicated with the *–h* flag to the port indicated by the *–i* flag. The port is specified as the IP address of the computer followed by the port. Specifying an IP address of 0.0.0.0 denotes the local computer.

Note Under Windows Vista, use the netsh command to configure SSL rather than httpcfg., like this: netsh http add sslcert ipport=0.0.0.0:8000 certhash=c390e7a4491cf97b96729167bf50186a4b68e052 appid={00112233-4455-6677-8899-AABBCCDDEEFF}. The certhash parameter specifies the thumbprint. The appid parameter is a GUID that identifies this binding of the certificate to the port; you can use any unique GUID.

Warning When a client application receives a certificate from a server, the WCF runtime attempts to ascertain that the certificate is valid and that the authority that issued it is trusted. The WCF runtime will fail this check when using the certificate that you have just installed. The following exercise shows how to force the WCF runtime to override this check and allow this certificate to be used. You should *never* do this in a production environment! The code is provided as-is, and without further explanation (it is not the author's work—it was written by developers at Microsoft and is included in one of the WCF technology samples provided with the Windows SDK). In the real world, you should go out and buy a valid certificate.

Add code to the WCF client to override certificate validation checking

1. In Visual Studio 2005, edit the Program.cs file for the ProductsClient project.

2. Add the following *using* statements to the list at the top of the file:

```
using System.Security.Crytography.X509Certificates;
using System.Net;
```

3. Add the following class to the *ProductsClient* namespace, underneath the *Program* class:

> **Note** The code for this class is available in the PermissiveCertificatePolicy.cs file in the Chapter 4 folder, if you don't want to type it in manually.

```
// WARNING: This code is only needed for test certificates such as those
// created by makecert. It is not recommended for production code.
class PermissiveCertificatePolicy
{
    string subjectName;
    static PermissiveCertificatePolicy currentPolicy;
    PermissiveCertificatePolicy(string subjectName)
    {
        this.subjectName = subjectName;
        ServicePointManager.ServerCertificateValidationCallback +=
            new System.Net.Security.RemoteCertificateValidationCallback
            (RemoteCertValidate);
    }

    public static void Enact(string subjectName)
    {
        currentPolicy = new PermissiveCertificatePolicy(subjectName);
    }

    bool RemoteCertValidate(object sender, X509Certificate cert,
            X509Chain chain, System.Net.Security.SslPolicyErrors error)
    {
        if (cert.Subject == subjectName)
        {
            return true;
        }

        return false;
    }
}
```

4. Add the following statement shown in bold to the Main method of the *Program* class, immediately before creating the proxy object:

```
…
PermissiveCertificatePolicy.Enact("CN=HTTPS-Server");
ProductServiceClient proxy = new ProductServiceClient(…);
…
```

Run the WCF client and service

1. Start the solution without debugging.
2. In the ProductsServiceHost form, click Start.

3. In the client console window, press Enter. Verify that the client application runs correctly.
4. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

Protecting an HTTP Service at the Message Level

You can configure the BasicHttpBinding binding to provide message level security by selecting the Message security mode for the binding. In this mode, the service uses SOAP message level security to encrypt the message. The service must have a certificate installed, and the client uses the public key from the service's certificate to perform the encryption. The service can send the certificate containing its public key at the start of the message exchange, or an administrator can install the service certificate on the client computer before the client application (in which case you must specify how to locate the service certificate in the client certificate store by adding a service behavior using the <*serviceCredentials*> element to the client configuration file). You will learn more about this in Chapter 5. Additionally, the only authentication mechanism supported by a WCF service that uses this mode requires that the client application identifies itself with a certificate—you cannot use authentication mechanisms such as Windows Integrated Security with this mode.

One other option is to use the TransportWithMessageCredential security mode. This is a hybrid combination of message level and transport level security. The service uses the HTTPS protocol and a certificate to provide message integrity and confidentiality. Client authentication is handled at the message level by using SOAP message security, and the client application can provide a username and password to identify the user. You will learn more about this security mode in Chapter 5.

If you really want to implement message level security for a WCF service with the minimum of fuss and configuration, you can opt to use the WSHttpBinding binding. The WSHttpBinding binding conforms to the current WS-* specifications and follows the WS-Security specification for encrypting messages and authenticating users by default. The following exercises demonstrate how to use the WSHttpBinding binding to implement message level security over HTTP.

Configure the WCF service to use the WSHttpBinding binding

1. In Visual Studio 2005, edit the App.config file for the ProductsServiceHost project by using the WCF Service Configuration Editor.
2. In the left pane, expand the Products.ProductsServiceImpl node under the Services folder, right-click Endpoints, and then click New Service Endpoint.
3. In the right pane, set the properties of the endpoint to the values in the following table. Leave all other properties with their default value:

Property	Value
Name	ProductsServiceWSHttpEndpoint
Address	http://localhost:8010/ProductsService/ProductsService.svc
Binding	wsHttpBinding
Contract	Products.IProductsService

Notice that the scheme used for the address of this endpoint is http, and not https.

4. Save the changes, and exit the WCF Service Configuration Editor.
5. Rebuild the ProductsServiceHost project.

Configure the WCF client to use the WSHttpBinding binding

1. Edit the app.config file for the ProductsClient project by using the WCF Service Configuration Editor.
2. In the left pane, right-click Endpoints in the Client folder, and then click New Client Endpoint.
3. In the right pane, set the properties of the endpoint to the values in the following table:

Property	Value
Name	WSHttpBinding_IProductsService
Address	http://localhost:8010/ProductsService/ProductsService.svc
Binding	wsHttpBinding
Contract	ProductsClient.ProductsService.IProductsService

4. Save the changes, and exit the WCF Service Configuration Editor.
5. In Visual Studio 2005, edit the Program.cs file in the ProductsClient project. In the Main method, change the code that creates the proxy object to use the new binding, as follows:

```
ProductsServiceClient proxy = new
    ProductsServiceClient("WSHttpBinding_IProductsService");
```

6. Rebuild the ProductsClient project.

Run the WCF client and service and examine the trace output

1. Using Windows Explorer, delete the existing trace file Products.svclog in the Microsoft Press\WCF Step By Step\Chapter 4\ProductsService folder under your \My Documents folder.
2. In Visual Studio 2005, start the solution without debugging.
3. In the ProductsServiceHost form, click Start. In the client console window, press Enter. Verify that the client application still runs correctly. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.
4. Start the Service Trace Viewer tool, and open the Products.svclog file.

5. In the Service Trace Viewer, in the left pane, click the Message tab.
6. Click the first message with the action *http://tempuri.org/IProductsService/ListProducts*. In the lower right pane, click the Message tab. You can see that the message has been encrypted—the body element of the message contains encrypted data.
7. In the left pane, click the second message with the action *http://tempuri.org/IProductsService/ListProducts*. In the right pane, scroll to the end of the Message window. This is the unencrypted version of the message passed from the message level to the service.
8. Examine the two ListProductsResponse messages. As with the NetTcpBinding example earlier in this chapter, you can see the encrypted version of the message being output by the service to the message level and the encrypted version of the message passing from the message level to the transport level.
9. Close the Service Trace Viewer.

The WSHttpBinding binding uses the 256-bit version of the AES encryption algorithm to encrypt data by default. You can select a different algorithm by creating a binding behavior and specifying the algorithm to use in the *AlgorithmSuite* property of the behavior, as you did when configuring message level security for the NetTcpBinding binding earlier in this chapter.

Authenticating Windows Users

So far, you have seen how to configure the NetTcpBinding, BasicHttpBinding, and WSHttpBinding bindings to support confidentiality and privacy by encrypting messages. However, transporting messages securely is only useful if a service can verify the identity of the user running the client application. In the exercises that follow, you will look at how a service can authenticate a user when the client application and service are both running within the same Windows domain. In Chapter 5, you will see how to perform authentication when a client and service are located in different, possibly non-Windows, security domains.

You will start by adding code to the ProductsService service that displays the name of the user calling the ListProducts operation. You will then be able to see the effect that the authentication options available in WCF have on the identity passed from a client application to a service.

Note You can configure authentication to be largely transparent to the WCF service. You will see in the exercises in this section that most of the actual authentication process is performed by the WCF runtime executing the service. All the service needs to do is specify the type of authentication it requires.

Display the name of the user calling an operation in the WCF service

1. In Visual Studio 2005, add a reference to the *System.Windows.Forms* assembly to the ProductsService project.
2. Open the ProductsService.cs file.

This file contains the code that implements the operations for the ProductsService service.

3. Add the following *using* statements to the list at the top of the file:

```
using System.Threading;
using System.Windows.Forms;
```

4. Locate the ListProducts method in the *ProductsServiceImpl* class. Add the following statements as the first two lines of the method:

```
string userName = Thread.CurrentPrincipal.Identity.Name;
MessageBox.Show("Username is " + userName,
                "ProductsService Authentication");
```

The first statement retrieves the name of the Windows user that the current thread is running on behalf of. The second statement displays the username in a message box.

5. Edit the Program.cs file in the ProductsClient project. In the Main method, change the code that creates the proxy object to use the BasicHttpBinding binding, as follows:

```
ProductsServiceClient proxy = new
    ProductsServiceClient("BasicHttpBinding_IProductsService");
```

6. Start the solution without debugging.

7. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

A message box appears, displaying the user name sent by the client application. The user name will appear to be missing. This is not an error. By default, the BasicHttpBinding binding does not send authentication information about users. All messages are sent as the anonymous user.

8. Click OK, and verify that the client application still runs correctly.

9. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

In the next set of exercises, you will revisit the BasicHttpBinding binding and implement user authentication. Many of the authentication options available for this binding apply to other bindings as well.

Configure the BasicHttpBinding binding for the WCF service to use Basic authentication

1. Edit the App.config file in the ProductsServiceHost project by using the WCF Service Configuration Editor.

2. In the left pane, expand the Bindings folder and click the ProductsServiceBasicHttpBindingConfig node.
3. In the right pane, click the Security tab.

   Notice that the *TransportClientCredentialType* property is currently set to *None*, so the service is not expecting client applications to provide authentication information about users, and anyone who can connect to the service can send it messages and invoke operations.
4. Set the *TransportClientCredentialType* property to *Basic*.

   When using Basic authentication, the client application must provide a username and password, which is transmitted to the service. The WCF runtime executing the service can use this information to authenticate the user running the client application, and if the user is valid, it will provide the identity of the user to the service.
5. Save the configuration, and close the WCF Service Configuration Editor.
6. Start the solution.
7. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

   The client fails with a MessageSecurityException exception, "The HTTP request is unauthorized with client authentication scheme 'Anonymous'... ." The WCF runtime for the service was expecting the client application to provide a username and password, which it has not done.
8. Close the client console window, stop the service, and close the ProductsServiceHost form.

Modify the WCF client to supply the user credentials to the service

1. In Visual Studio 2005, edit the app.config file in the ProductsClient project by using the WCF Service Configuration Editor.
2. In the left pane, expand the Bindings folder and click the ProductsClientBasicHttpBindingConfig node.
3. In the right pane, click the Security tab.
4. Set the *TransportClientCredentialType* property to *Basic*.
5. Save the configuration, and close the WCF Service Configuration Editor.
6. Edit the Program.cs file in the ProductsClient project.
7. In the Main method, add the following statements shown in bold immediately after the code that creates the proxy object. Replace LON-DEV-01 with the name of your domain or computer (if you are not currently a member of a domain), replace Student with your username, and replace Pa$$w0rd with your password:

```
ProductsServiceClient proxy = new
    ProductsServiceClient("BasicHttpBinding_IProductsService");
```

```
proxy.ClientCredentials.UserName.UserName = "LON-DEV-01\\Student";
proxy.ClientCredentials.UserName.Password = "Pa$$w0rd";
```

The *ClientCredentials* property of a WCF proxy object provides a mechanism for a client application to provide the credentials to send to the service. The *UserName* property of *ClientCredentials* can hold a username and password. Other properties are available, such as *ClientCertificate*, which enable you to supply different types of credentials information as required by the service configuration.

Warning This code is for illustrative purposes in this exercise only. In a production application, you should prompt the user for their name and password. You should never hard-code these details into an application.

8. Start the solution without debugging.
9. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

   A message box appears, displaying the user name sent by the client application. This time, the user name appears as expected, verifying that the operation is executing with the credentials of the user.

10. Click OK, and verify that the client application still runs correctly.
11. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

Using Basic authentication, you can provide the username and password of the user, and the WCF runtime executing the service will check that these credentials are valid. If you provide an invalid username of password, the WCF runtime will reject the request and the client will receive another MessageSecurityException exception with the message "The HTTP request was forbidden... ."

Basic authentication is a good solution if the user running the client application is not currently logged into the security domain used by the service.

Note You can also configure the NetTCPBinding and WSHttpBinding bindings at the message level to require Username authentication. This is very similar to Basic authentication at the transport level as far as client application is concerned, although somewhat different as far as the service is concerned, as it takes responsibility for authenticating the user itself (typically using a custom database of usernames and passwords). However, usernames and passwords are not encrypted at the message level, so WCF insists that the underlying transport provide encryption to prevent the credential details being transmitted across an open network as clear text.

If the user is logged in to the domain, then you can make use of Windows Integrated Security to provide the user's credentials automatically, rather than prompting the user for them again (or worse still, hard-coding them in your application!).

Configure the BasicHttpBinding binding for the WCF service and client to use Windows authentication

1. Edit the App.config file in the ProductsServiceHost project by using the WCF Service Configuration Editor.
2. In the left pane, expand the Bindings folder, and click the ProductsServiceBasicHttpBindingConfig node.
3. In the right pane, click the Security tab.
4. Set the *TransportClientCredentialType* property to Windows.
5. Save the configuration and close the WCF Service Configuration Editor.
6. In Visual Studio 2005, edit the app.config file in the ProductsClient project by using the WCF Service Configuration Editor.
7. Repeat the process in steps 2 through 5, above and set the *TransportClientCredentialType* property of the ProductsClientBasicHttpBindingConfig binding configuration to Windows.
8. Save the configuration, and close the WCF Service Configuration Editor.
9. Edit the Program.cs file in the ProductsClient project.
10. In the Main method, comment out the two statements that add the username and password to the *ClientCredentials* property of the proxy object.
11. Start the solution without debugging.
12. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

    The message box appears displaying your Windows username, which was sent by the client application. However, rather than you having to supply the username and password, the WCF runtime executing the client application picked this information up from the user's process automatically.

Note If you omitted to comment out the lines that populated the *ClientCredentials* object, the solution still works; the credentials provided are simply ignored. However, note the *ClientCredentials* property has a Windows property that you can use to provide a domain, username, and password to the service if you want the service to run as a different Windows user. Any values that you specify in the *Windows* property override those retrieved from the user's login process. The usual warnings about hard-coding usernames and password in your code still apply:

```
proxy.ClientCredentials.Windows.ClientCredential.Domain = "LON-DEV-01";
proxy.ClientCredentials.Windows.ClientCredential.UserName = "Administrator";
proxy.ClientCredentials.Windows.ClientCredential.Password = "P@ssw0rd";
```

13. Click OK in the message box, and verify that the client application still runs correctly.
14. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

When you use Windows Integrated Security, usernames and passwords are not transmitted as clear text. You can use Windows Integrated Security at the message level with the NetTCPBinding and WSHttpBinding bindings without needing to implement encryption at the transport level.

Examine the authentication mechanism used by the NetTcpBinding binding

1. Edit the App.config file in the ProductsServiceHost project by using the WCF Service Configuration Editor.
2. In the left pane, expand the Bindings folder, and click the ProductsServiceBasicTcpBindingConfig node.
3. In the right pane, click the Security tab.
4. Verify that the *MessageClientCredentialType* property is set to Windows.

   You have been using Windows Integrated Security without realizing it in earlier exercises!

Note The WSHttpBinding binding also defaults to using Windows Integrated Security.

5. Close the WCF Service Configuration Editor without saving changes.
6. Edit the Program.cs file for the ProductsClient project and modify the statement that creates the proxy object to use the NetTcpBinding binding, as follows:

```
ProductsServiceClient proxy = new
    ProductsServiceClient("NetTcpBinding_IProductsService");
```

7. Start the solution without debugging.
8. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

   The familiar message box appears, displaying your Windows user name, proving that the NetTcpBinding automatically picks up your identity from Windows.
9. Click OK, and allow the client application to finish. Press Enter to close the client console window. Stop the service and close the ProductsServiceHost form.

Authorizing Users

After a service has established the identity of the user, it can then determine whether the service should perform the requested operations for the user. Different operations in a service could be considered more privileged than others. For example, in the ProductsService service, you might wish to let any staff who work in the warehouse query the product information in the *AdventureWorks* database but limit access to operations such as ChangeStockLevel, which modify data, to staff members who are stock controllers. WCF can use the features of the .NET Framework to enable a developer to specify which users and roles have the authority to request operations. You can perform this task declaratively (by using attributes) or imperatively (by adding code to the operations).

The authorization mechanism used by WCF requires access to a database defining users and the roles that they can fulfill. If you are performing authentication by using Active Directory, it makes sense to use the *Active Directory* database to hold the roles for each user as well. Therefore, the first step is to ensure that the WCF service is configured to retrieve roles from Active Directory by using the Windows Token Role Provider.

Configure the WCF service to use the Windows Token Role Provider

1. Edit the App.config file in the ProductsServiceHost project by using the WCF Service Configuration Editor.
2. In the left pane, expand the Advanced folder, expand the Service Behaviors folder and then click the ProductsBehavior node.

   The ProductsBehavior behavior currently contains the *serviceDebug* element. You added this behavior to the service in Chapter 3.
3. In the right pane, click Add.
4. In the Adding Behavior Element Extension Sections dialog box, select serviceAuthorization and then click Add.

   The serviceAuthorization behavior is added to the list of behaviors.
5. In the left pane, click serviceAuthorization under the ProductsBehavior node.
6. In the right pane, verify that the *PrincipalPermissionMode* property is set to *UseWindowsGroups*.

   By default, WCF uses the Windows Token Role Provider to authenticate users, so you don't actually need to change anything. However, you can configure the *serviceBehavior* element to specify a different role provider, such as the SQL Role Provider or the Authorization Store Role Provider mentioned earlier in this chapter. (You will configure the service to use the SQL Role Provider in Chapter 5.)
7. Save the configuration and close the WCF Service Configuration Editor.

The next step is to define the roles that can request the operations in the WCF service. When using the Windows Token Role Provider, Active Directory groups correspond to roles, so you define groups in the *Active Directory* database and add users to these groups.

Note The following exercise assumes you do not have access to the *Active Directory* database for your organization, so it uses the Windows local users and groups database instead. The principles are the same, however.

Create groups for warehouse staff and stock controller staff

1. On the Windows Start menu, right-click My Computer, and then click Manage.

   The Computer Management console appears.

2. In the Computer Management console, under the System Tools node, expand the Local Users and Groups node, right-click the Groups folder, and then click New Group.

3. In the New Group dialog box, enter WarehouseStaff for the Group name, and then click Create.

4. Still in the New Group dialog box, enter StockControllers for the Group name, and then click Create.

5. Click Close to close the New Group dialog box.

   The two new groups should appear in the list of groups in right pane of the Computer Management console.

6. In the left pane of the Computer Management console, right-click the Users folder and then click New User.

7. In the New User dialog box, use the values in the following table to set the properties of the user and then click Create.

Property	Value
User name	Fred
Password	Pa$$w0rd
Confirm password	Pa$$w0rd
User must change password at next logon	Unchecked

8. Add another user by specifying the values in the following table, and then click Create again.

Property	Value
User name	Bert
Password	Pa$$w0rd

Property	Value
Confirm password	Pa$$w0rd
User must change password at next logon	Unchecked

9. Click Close the close the New User dialog box.
10. In the left pane of the Computer Management console, click the Users folder.

    The two new users should appear in the list in the right pane of the Computer Management console.
11. In the right pane of the Computer Management console, right-click Bert and then click Properties.
12. In the Bert Properties dialog box, click the Member Of tab and then click Add.
13. In the Select Groups dialog box, type WarehouseStaff in the text box and then click OK.

    Bert is added to the WarehouseStaff group.
14. In the Bert Properties dialog box, click OK.
15. In the right pane of the Computer Management console, right-click Fred and then click Properties.
16. In the Fred Properties dialog box, click the Member Of tab and then click Add.
17. In the Select Groups dialog box, type WarehouseStaff in the text box and then click OK.
18. Click Add again. In the Select Groups dialog box, type StockControllers in the text box and then click OK.

    Fred is added to the WarehouseStaff and StockControllers groups—he has two roles.
19. In the Fred Properties dialog box, click OK.
20. Close the Computer Management console.

You can now use the groups you have just defined to specify the roles that can request each of the operations in the ProductsService service. To show how to specify authorization declaratively and imperatively, you will use attributes to specify the role for the operations that simply query the *AdventureWorks* database, but you will write code to specify the role that can modify the database.

Specify the roles for the WCF service operations

1. In Visual Studio 2005, open the ProductsService.cs file in the ProductsService project.
2. Add the following *using* statements to the list at the top of the file:

```
using System.Security;
using System.Security.Permissions;
using System.Security.Principal;
```

3. Locate the ListProducts method in the *ProductsServiceImpl* class. Add the following attribute, shown in bold, to this method:

```
[PrincipalPermission(SecurityAction.Demand, Role="WarehouseStaff")]
public List<string> ListProducts()
{
    …
}
```

The *PrincipalPermission* attribute specifies the authorization requirements of the method. In this case, the SecurityAction.Demand parameter indicates that the method requires that the user meet the criteria specified by the following parameters. The Role parameter indicates that the user must be a member of the WarehouseStaff role.

You can identify specific users by using the optional Name parameter. However, if you specify Name and Role, then the user must match both criteria to be granted access (if the user is not a member of the specified role, they will not be allowed to execute the method). If you require users to be granted access to the method if they have a specific name *or* are a member of a specific group, you can use the *PrincipalPermission* attribute twice, like this:

```
[PrincipalPermission(SecurityAction.Demand, Role="WarehouseStaff")]
// LON-DEV-01\Student is not a member of the WarehouseStaff group
[PrincipalPermission(SecurityAction.Demand, Name="LON-DEV-01\\Student")]
public List<string> ListProducts()
{
    …
}
```

You can also specify SecurityAction.Deny as the first parameter to the *PrincipalPermission* attribute. If you do this, the specified users and roles will be explicitly denied access to the method.

4. Apply the *PrincipalPermission* attribute with the WarehouseStaff group to the GetProduct and CurrentStockLevel methods, as shown in bold below:

```
[PrincipalPermission(SecurityAction.Demand, Role="WarehouseStaff")]
public Product GetProduct(string ProductNumber)
{
    …
}

[PrincipalPermission(SecurityAction.Demand, Role="WarehouseStaff")]
public int GetStockLevel(string ProductNumber)
{
    …
}
```

5. Locate the ChangeStockLevel method. Add the following code, shown in bold, to the start of this method:

```
public bool ChangeStockLevel(…)
{
```

```
        // Determine whether the user is a member of the StockControllers role
        WindowsPrincipal user = new WindowsPrincipal(
                (WindowsIdentity)Thread.CurrentPrincipal.Identity);
        if (!(user.IsInRole("StockControllers")))
        {
            // If the user is not in the StockControllers role,
            // throw a SecurityException
            throw new SecurityException("Access denied");
        }     …
}
```

The first statement retrieves the identity information for the user and uses it to create a *WindowsPrincipal* object. Note that the identity returned by the current thread must be cast to a *WindowsIdentity* object. A *WindowsPrincipal* object is a representation of the user. It exposes the IsInRole method that this code uses to determine whether the user is a member of the StockControllers role. The IsInRole method returns *true* if the user is a member of the role, *false* otherwise. If the user is not a member of the role, the code throws a SecurityException exception with the message "Access Denied."

> **Warning** It is tempting to provide more detail in the SecurityException exception. This practice is not recommended, as it could provide an attacker with useful information that they might be able to use to try and infiltrate your system. Keep the exception message bland!

Test the authorization for the WCF service

1. Start the solution without debugging.
2. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

   Assuming you are not currently logged in to Windows as Fred or Bert, the client application stops and reports the message "Access is denied" when attempting to invoke the ListProducts operation. This is because the authenticated Windows account for the client application must be a member of the WarehouseStaff role:

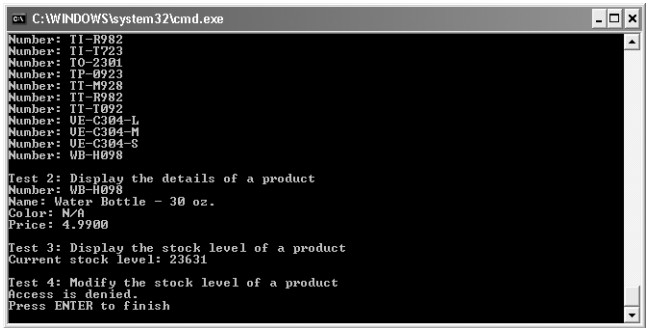
3. Press Enter to close the client console window, and then stop the service and close the ProductsServiceHost form.

4. In the ProductsClient project in Solution Explorer, open the Program.cs file.
5. In the Main method, add the following statements shown in bold immediately after the statement that creates the proxy object. Replace the value "*LON-DEV-01*" specified in the *Domain* property with the name of your computer:

```
ProductsServiceClient proxy = new
    ProductsServiceClient("NetTcpBinding_IProductsService");
proxy.ClientCredentials.Windows.ClientCredential.Domain = "LON-DEV-01";
proxy.ClientCredentials.Windows.ClientCredential.UserName = "Bert";
proxy.ClientCredentials.Windows.ClientCredential.Password = "Pa$$w0rd";
```

   These statements explicitly set the Windows credentials for the user to those of Bert. The WCF runtime on the client will send these credentials to the service, rather than using those in the user's login process.
6. Start the solution again, without debugging.
7. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

   This time, Bert is a member of the WarehouseStaff role and is granted access to the ListProducts, GetProduct, and CurrentStockLevel operations.
8. When the ListProducts method runs, it displays the message box confirming that the identity of the authenticated user is Bert. Click OK to continue execution. The first three tests run successfully, but when the client application attempts to perform test 4, which requires invoking the ChangeStockLevel operation, Bert has not been granted access to this method, and so the test fails with the "Access is denied" message:



9. Press Enter to close the client console window, and then stop the service and close the ProductsServiceHost form.
10. Return to the Program.cs file in the code view window.
11. In the Main method, change the Windows username of the user to Fred, like this:

```
proxy.ClientCredentials.Windows.ClientCredential.Domain = "LON-DEV-01";
proxy.ClientCredentials.Windows.ClientCredential.UserName = "Fred";
proxy.ClientCredentials.Windows.ClientCredential.Password = "Pa$$w0rd";
```

12. Build and start the solution again without debugging.
13. In the ProductsServiceHost form, click Start. In the client console window, press Enter.

    Fred is a member of the WarehouseStaff role and the StockControllers role, and so he is able to invoke all the operations in the ProductsService service.
14. When the ListProducts method displays the message box with the name of the authenticated user, verify that the username is Fred and then click OK.
15. The client application performs all four tests successfully. Press Enter to close the client console window, and then stop the service and close the ProductsServiceHost form.

Using Impersonation to Access Resources

Authenticating a user establishes the identity of the user to the WCF service, which can then perform authorization checks to verify that the user should be allowed to perform the requested operation. The method that implements the operation might require access to resources on the computer running the WCF service. By default, the service will attempt to gain access to these resources by using its own credentials. For example, when a method in the ProductsService service connects to the *AdventureWorks* database, it does so as the account running the service. When using Windows authentication, it is possible to specify that the WCF service should access resources by using the authenticated identity of the user instead. So, if Fred has been granted access to the *AdventureWorks* database, the WCF service can connect to SQL Server as Fred and will have access to all the database resources to which Fred has been granted access. If the user connects as Bert, the WCF service might be able to use a different set of resources in the database, depending on Bert's access rights. The same principle applies to other resources, such as files, folders, and network shares. Using impersonation gives an administrator fine-grained control over the ability of a WCF service to read or write possibly sensitive information and can provide an additional degree of security—just because the user can connect to the WCF service, they might not be able to perform operations that retrieve or modify confidential data unless the administrator has explicitly granted the user access to this data.

You can enable impersonation for an operation by setting the *Impersonation* property of the *OperationBehavior* attribute, like this (shown in bold):

```
[PrincipalPermission(SecurityAction.Demand, Role="WarehouseStaff")]
[OperationBehavior(Impersonation=ImpersonationLevel.Required)]
public List<string> ListProducts
{
  …
}
```

Specifying the value *ImpersonationLevel.Required* enforces impersonation. The client application must also agree to this requirement and specify the level of impersonation that the WCF service application can use (you will see how to do this shortly). You can also specify the *ImpersonationLevel.Allowed*, which enables the WCF service to impersonate the user if the cli-

ent application permits, but executes as the identity running the service application if not, and *ImpersonationLevel.NotAllowed*, which disables impersonation.

If you need to specify an impersonation level setting for all operations, you can set the *ImpersonateCallerForAllOperations* attribute of the *<serviceBehavior>* element of the service behavior to *true* in the service configuration file, as shown in bold below:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  …
  <system.serviceModel>
    …
    <services>
      <service behaviorConfiguration="ProductsBehavior" name="Products.ProductsServiceImpl">
        …
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ProductsBehavior">
          <serviceAuthorization principalPermissionMode="UseWindowsGroups"
            impersonateCallerForAllOperations="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

You configure the client application to indicate the level of impersonation that the service can use by defining a behavior for the endpoint and specifying the *AllowedImpersonationLevel* property. The following fragments of a client configuration file highlight the pertinent elements:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="ImpersonationBehavior">
          <clientCredentials>
            <windows allowedImpersonationLevel="Impersonation" />
            …
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    …
    <client>
      …
      <endpoint
         address="http://localhost:8010/ProductsService/ProductsService.svc"
        behaviorConfiguration="ImpersonationBehavior"
        binding="wsHttpBinding"
```

```
        contract="ProductsClient.ProductsService.IProductsService"
        name="WSHttpBinding_IProductsService" />
    </client>
  </system.serviceModel>
</configuration>
```

You can specify one of the following values for the *AllowedImpersonationLevel* property:

- *Impersonate*. The service can use the user's identity when accessing local resources on the computer hosting the service. However, the service cannot access resources on remote computers.
- *Delegation*. The service can use the user's identity when accessing local resources on the computer hosting the service and on remote computers. The service can pass the identity of the user on to remote services, which may authenticate the user and perform operations impersonating this user.
- *Identify*. The service can use the user's credentials to authenticate the user and authorize access to operations but cannot impersonate the user.
- *Anonymous*. The service does not use the user's identity to authenticate the user but can use the user's credentials to perform access checks against resources accessed by the service. This setting is only valid for transport mechanisms such as named pipes that connect a client application to a service executing on the same computer. If the service is running on a remote computer, the setting is handled in the same way as the "Identify" option.
- *None*. The service does not attempt to impersonate the user.

Summary

In this chapter, you have seen how to use the features of WCF bindings to control the degree of protection afforded to a WCF service. You have seen how to configure encryption for messages flowing between a client application and a service, at the message level and at the transport level. You have learned how to specify the authentication mode for a binding and how to pass Windows credentials from a client application to a WCF service. You have also learned how to authorize access to operations for authenticated users and how to provide access to resources based on a user's authenticated identity by using impersonation.