**Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication**

[](http://www.msdn.microsoft.com/practices)

**Web Services Security**

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy  
Microsoft Corporation

Published: November 2002

Last Revised: January 2006

**Applies to:**

* Web Services (.NET Framework 1.1)

See the "[patterns & practices Security Guidance for Applications Index](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/SecurityGuidanceIndex.asp)" for links to additional security resources.

See the [Landing Page](http://msdn.microsoft.com/en-us/library/cc394781.aspx) for the starting point and complete overview of *Building Secure ASP.NET Applications*.

**Summary:** This chapter focuses on platform-level security for Web services using the underlying features of IIS and ASP.NET. (29 printed pages)

**Contents**

[Web Service Security Model](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_webservicesecmodel)  
[Platform/Transport Security Architecture](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_platformtransportsecarch)  
[Authentication and Authorization Strategies](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_authstrategies)  
[Configuring Security](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_configuringsecurity)  
[Passing Credentials for Authentication to Web Services](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_passingcredentialsforauth)  
[Flowing the Original Caller](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_flowingorigcaller)  
[Trusted Subsystem](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_trustedsubsystem)  
[Accessing System Resources](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_accesssysresources)  
[Accessing Network Resources](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_accessnetworkresources)  
[Accessing COM Objects](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_accesscomobjects)  
[Using Client Certificates with Web Services](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_usingclientcertificates)  
[Secure Communication](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_securecommunication)  
[Summary](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_summary)

This chapter describes how to develop and apply authentication, authorization, and secure communication techniques to secure ASP.NET Web services and Web service messages. It describes security from the Web service perspective and shows you how to authenticate and authorize callers and how to flow security context through a Web service. It also explains, from a client-side perspective, how to call Web services with credentials and certificates to support server-side authentication.

**Web Service Security Model**

Web service security can be applied at three levels:

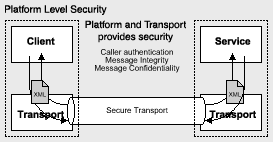
* Platform/transport-level (point-to-point) security
* Application-level (custom) security
* Message-level (end-to-end) security

Each approach has different strengths and weaknesses, and these are elaborated upon below. The choice of approach is largely dependent upon the characteristics of the architecture and platforms involved in the message exchange.

**Note**   This chapter focuses on platform- and application-level security. Message-level security is addressed by the Global XML Web Services Architecture (GXA) initiative and specifically the WS-Security specification. The Web Services Development Kit allows you to develop message-level security solutions that conform to the WS-Security specification.

**Platform/Transport Level (Point-to-Point) Security**

The transport channel between two endpoints (Web service client and Web service) can be used to provide point-to-point security. This is illustrated in Figure 10.1.



**Figure 10.1. Platform/transport-level security**

When you use platform security, which assumes a tightly-coupled Microsoft® Windows® operating system environment, for example, on corporate intranets:

* The Web server (IIS) provides Basic, Digest, Integrated, and Certificate authentication.
* The ASP.NET Web service inherits some of the ASP.NET authentication and authorization features.
* SSL and/or IPSec may be used to provide message integrity and confidentiality.

**When to use**

The transport-level security model is simple, well understood, and adequate for many (primarily intranet-based) scenarios, in which the transport mechanisms and endpoint configuration can be tightly controlled.

The main issues with transport-level security are:

* Security becomes tightly coupled to, and dependant upon, the underlying platform, transport mechanism, and security service provider (NTLM, Kerberos, and so on).
* Security is applied on a point to point basis, with no provision for multiple hops and routing through intermediate application nodes.
* Because the entire message needs to be encrypted, this approach can be costly.

**Application Level Security**

With this approach, the application takes over security and uses custom security features. For example:

* An application can use a custom SOAP header to pass user credentials to authenticate the user with each Web service request. A common approach is to pass a ticket (or user name or license) in the SOAP header.
* The application has the flexibility to generate its own **IPrincipal** object that contains roles. This might be a custom class or the **GenericPrincipal** class provided by the .NET Framework.
* The application can selectively encrypt what it needs to, although this requires secure key storage and developers must have knowledge of the relevant cryptography APIs.

An alternative technique is to use SSL to provide confidentiality and integrity and combine it with custom SOAP headers to perform authentication.

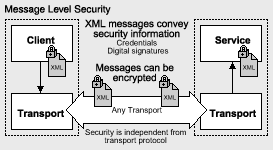
**When to use**

Use this approach when:

* You want to take advantage of an existing database schema of users and roles that is used within an existing application.
* You want to encrypt parts of a message, rather than the entire data stream.

**Message Level (End-to-End) Security**

This represents the most flexible and powerful approach and is the one used in Web Services Enhancements for Microsoft .NET (WSE). Message-level security is illustrated in Figure 10.2.



**Figure 10.2. Message-level security**

WS-Security specifications describe enhancements to SOAP messaging that provide message integrity, message confidentiality, and single message authentication.

* Authentication is provided by security tokens, which flow in SOAP headers. No specific type of token is required by WS-Security. The security tokens may include Kerberos tickets, X.509 certificates, or a custom Extensible Markup Language (XML) or binary token.
* Secure communication is provided by XML digital signatures to ensure message integrity and XML encryption for message confidentiality.

**When to use**

WS-Security can be used to construct a framework for exchanging secure messages in a heterogeneous Web services environment. It is ideally suited for most scenarios.

Message-level security:

* Can be independent from the underlying transport
* Enables a heterogeneous security architecture
* Provides end-to-end security and accommodates message routing through intermediate application nodes
* Supports multiple encryption technologies
* Supports non-repudiation

**The Web Services Enhancements for Microsoft .NET**

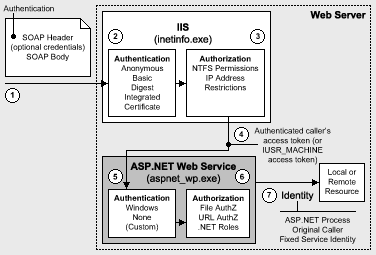
The Web Services Enhancements for Microsoft .NET (WSE) provides the necessary APIs to manage security in addition to other services such as routing and message-level referrals. The WSE conforms to the latest advanced Web services capabilities to keep pace with evolving Web services protocol specifications.

**More information**

* For the latest news about WSE, see the Web Services and Other Distributed Technologies Developer Center page on MSDN® at <http://msdn.microsoft.com/webservices/>.
* For more information about the WS-Specification, see the [WS-Security Specification Index](http://msdn.microsoft.com/webservices/webservices/building/security/default.aspx?pull=/library/en-us/dnglobspec/html/ws-security.asp) page.

**Platform/Transport Security Architecture**

* The ASP.NET Web services platform security architecture is shown in Figure 10.3.



**Figure 10.3. Web services security architecture**

Figure 10.3 illustrates the authentication and authorization mechanisms provided by ASP.NET Web services. When a client calls a Web service, the following sequence of authentication and authorization events occurs:

1. The SOAP request is received from the network. This may or may not contain authentication credentials depending upon the type of authentication being used.
2. IIS optionally authenticates the caller by using Basic, Digest, Integrated (NTLM or Kerberos), or Certificate authentication. In heterogeneous environments where IIS (Windows) authentication is not possible, IIS is configured for anonymous authentication. In this scenario, the client may be authenticated by using message-level attributes such as tickets passed in the SOAP header.
3. IIS can also be configured to accept requests only from client computers with specific IP addresses.
4. IIS passes the authenticated caller's Windows access token to ASP.NET (this may be the anonymous Internet user's access token, if the Web service is configured for anonymous authentication).
5. ASP.NET authenticates the caller. If ASP.NET is configured for Windows authentication, no additional authentication occurs at this point; IIS authenticates the caller.

If a non-Windows authentication method is being used, the ASP.NET authentication mode is set to None to allow custom authentication.

1. ASP.NET authorizes access to the requested Web service (.asmx file) by using URL authorization and File authorization, which uses NTFS permissions associated with the .asmx file to determine whether or not access should be granted to the authenticated caller.

**Note**   File authorization is only supported for Windows authentication.

For fine-grained authorization, .NET roles can also be used (either declaratively or programmatically) to ensure that the caller is authorized to access the requested Web method.

1. Code within the Web service may access local and/or remote resources by using a particular identity. By default, ASP.NET Web services perform no impersonation and, as a result, the configured ASP.NET process account provides the identity. Alternate options include the original caller's identity, or a configured service identity.

**Gatekeepers**

The gatekeepers within an ASP.NET Web service are:

* IIS
  + If IIS anonymous authentication is disabled IIS only allows requests from authenticated users.
  + IP Address Restrictions

IIS can be configured to only allow requests from computers with specific IP addresses.

* ASP.NET
  + The File authorization HTTP Module (for Windows authentication only)
  + The URL authorization HTTP Module
* Principal Permission Demands and Explicit Role Checks

**More information**

* For more information about the gatekeepers, see [Gatekeepers](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."
* For more information about configuring security, see "Configuring Security" later in this chapter.

**Authentication and Authorization Strategies**

This section explains which authorization options (configurable and programmatic) are available for a set of commonly used authentication schemes.

The following authentication schemes are summarized here:

* Windows authentication with impersonation
* Windows authentication without impersonation
* Windows authentication using a fixed identity

**Windows Authentication with Impersonation**

The following configuration elements show you how to enable Windows (IIS) authentication and impersonation declaratively in Web.config or Machine.config.

**Note**   You should configure authentication on a per-Web service basis in each Web service's Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl07');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl07');)

<authentication mode="Windows" />

<identity impersonate="true" />

With this configuration, your Web service code impersonates the IIS-authenticated caller. To impersonate the original caller, you must turn off anonymous access in IIS. With anonymous access, the Web service code impersonates the anonymous Internet user account (which by default is IUSR\_MACHINE).

**Configurable security**

When you use Windows authentication together with impersonation, the following authorization options are available to you:

* **Windows Access Control Lists** (**ACLs)**
  + **Web service (.asmx) file**. File authorization performs access checks for requested ASP.NET resources (which include the .asmx Web service file) using the original caller's security context. The original caller must be granted at least read access to the .asmx file.
  + **Resources accessed by your Web service**. Windows ACLs on resources accessed by your Web service (files, folders, registry keys, Active Directory® directory service objects and so on) must include an Access Control Entry (ACE) that grants read access to the original caller (because the Web service thread used for resource access is impersonating the caller).
* **URL Authorization.** This is configured in Machine.config and/or Web.config. With Windows authentication, user names take the form DomainName\UserName and roles map one-to-one with Windows groups.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl08');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl08');)

<authorization>

<deny user="DomainName\UserName" />

<allow roles="DomainName\WindowsGroup" />

</authorization>

**Programmatic security**

Programmatic security refers to security checks located within your Web service code. The following programmatic security options are available when you use Windows authentication and impersonation:

* **Principal Permission Demands**
  + Imperative (in-line within a method's code)

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl09');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl09');)

PrincipalPermission permCheck = new PrincipalPermission(

null, @"DomainName\WindowsGroup");

permCheck.Demand();

* + Declarative (these attributes can precede Web methods or Web classes)

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl10');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl10');)

// Demand that the caller is a member of a specific role (for

// Windows

// authentication this is the same as a Windows group)

[PrincipalPermission(SecurityAction.Demand,

Role=@"DomainName\WindowsGroup")]

// Demand that the caller is a specific user

[PrincipalPermission(SecurityAction.Demand,

Name=@"DomainName\UserName")]

* **Explicit Role Checks**. You can perform role checking using the **IPrincipal** interface.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl11');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl11');)

IPrincipal.IsInRole(@"DomainName\WindowsGroup");

**Note**   When using the Role Manager feature In ASP.Net 2.0, you can use the Roles API for role checks. For more information, see "[How To: Use Role Manager in ASP.NET 2.0](http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000013.asp)."

**When to use**

Use Windows authentication and impersonation when:

* The clients of the Web service can be identified by using Windows accounts, which can be authenticated by the server.
* You need to flow the original caller's security context through the Web service and onto the next tier. For example, a set of serviced components that use Enterprise Services (COM+) roles, or onto a data tier that requires fine-grained (per-user) authorization.
* You need to flow the original caller's security context to the downstream tiers to support operating system-level auditing.
* You need to use the original user's security context to access local resources.
* You need to delegate the original user's security context to access downstream resources.

**Important**   Using impersonation/delegation can reduce scalability, because it impacts database connection pooling. As an alternative approach, consider using the trusted subsystem model where the Web service authorizes callers and then uses a fixed identity for database access. You can flow the caller's identity at the application level; for example, by using stored procedure parameters.

**More information**

* For more information about Windows authentication and impersonation, see Chapter 8, [ASP.NET Security](http://msdn.microsoft.com/en-us/library/cc414309.aspx).
* For more information about URL authorization, see [URL Authorization Notes](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**Windows Authentication without Impersonation**

The following configuration elements show how you enable Windows (IIS) authentication with no impersonation declaratively in Web.config.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl15');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl15');)

<authentication mode="Windows" />

<!-- The following setting is equivalent to having no identity element -->

<identity impersonate="false" />

**Configurable security**

When you use Windows authentication without impersonation, the following authorization options are available to you:

* **Windows** **ACLs**
  + **Web Service (.asmx) file**. File authorization performs access checks for requested ASP.NET resources (which include the .asmx Web service file) using the original caller. Impersonation is not required.
  + **Resources accessed by your application**. Windows ACLs on resources accessed by your application (files, folders, registry keys, Active Directory objects) must include an ACE that grants read access to the ASP.NET process identity (the default identity used by the Web service thread when accessing resources).
* **URL Authorization**

This is configured in Machine.config and Web.config. With Windows authentication, user names take the form DomainName\UserName and roles map one-to-one with Windows groups.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl16');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl16');)

<authorization>

<deny user="DomainName\UserName" />

<allow roles="DomainName\WindowsGroup" />

</authorization>

**Programmatic security**

Programmatic security refers to security checks located within your Web service code. The following programmatic security options are available when you use Windows authentication without impersonation:

* **Principal Permission Demands**
  + Imperative

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl17');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl17');)

PrincipalPermission permCheck = new PrincipalPermission(

null, @"DomainName\WindowsGroup");

permCheck.Demand();

* + Declarative

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18');)

// Demand that the caller is a member of a specific role (for

// Windows

// authentication this is the same as a Windows group)

[PrincipalPermission(SecurityAction.Demand,

Role=@"DomainName\WindowsGroup")]

// Demand that the caller is a specific user

[PrincipalPermission(SecurityAction.Demand,

Name=@"DomainName\UserName")]

* **Explicit Role Checks**. You can perform role checking using the **IPrincipal** interface.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19');)

IPrincipal.IsInRole(@"DomainName\WindowsGroup");

**Note**   When using the Role Manager feature in ASP.Net 2.0, you can use the Roles API for role checks. For more information, see "[How To: Use Role Manager in ASP.NET 2.0](http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000013.asp)."

**When to use**

Use Windows authentication without impersonation when:

* The clients of the Web service can be identified by using Windows accounts, which can be authenticated by the server.
* You want to use the trusted subsystem model and authorize clients within the Web service and then use a fixed identity to access downstream resources (for example, databases) in order to support connection pooling.

**More information**

* For more information about Windows authentication and impersonation, see Chapter 8, [ASP.NET Security](http://msdn.microsoft.com/en-us/library/cc414309.aspx).
* For more information about URL authorization, see [URL Authorization Notes](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**Windows Authentication Using a Fixed Identity**

The <**identity**> element within Web.config supports optional user name and password attributes which allows you to configure a specific fixed identity for your Web service to impersonate. This is shown in the following configuration file fragment.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23');)

<identity impersonate="true" userName="DomainName\UserName"

password="ClearTextPassword" />

**When to use**

This approach is typically used when you have multiple Web services on the same Web server that need to run under different identities; for example, in application hosting scenarios where data/resource isolation is a concern.

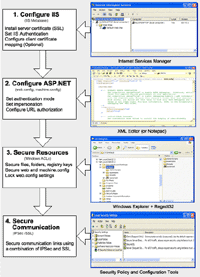
**Note**However, if you are running Windows Server 2003 with IIS 6.0 configured to run in worker isolation mode (the default), you can avoid impersonation by configuring your ASP.NET application to run in a custom application pool that runs under a specific domain identity. For more information, see "[How To: Create a Service Account for an ASP.NET 2.0 Application](http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000009.asp)."

**More information**

* For more information about Windows authentication and impersonation, see Chapter 8, [ASP.NET Security](http://msdn.microsoft.com/en-us/library/cc414309.aspx).
* For more information about URL authorization, see [URL Authorization Notes](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**Configuring Security**

This section shows you the practical steps required to configure security for an ASP.NET Web service. These are summarized in Figure 10.4.

[](http://msdn.microsoft.com/en-us/library/Aa302390.f08sn03(en-us,MSDN.10).gif)

**Figure 10.4. Configuring ASP.NET Web service security**

**Configure IIS Settings**

For detailed information about how to configure IIS security settings, see [Configuring Security](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security," because the information is also applicable to ASP.NET Web services.

**Configure ASP.NET Settings**

Application-level configuration settings are maintained in Web.config files, which are located in your Web service's virtual root directory. Configure the following settings:

1. **Configure Authentication**. This should be set on a per-Web service basis (not in Machine.config) in the Web.config file located in the Web service's virtual root directory.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl29');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl29');)

<authentication mode="Windows|None" />

**Note**   Web services do not currently support Passport or Forms authentication. For custom and message-level authentication, set the mode to None.

1. **Configure Impersonation and Authorization**. For detailed information, see [Configuring Security](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**More information**

For more information about URL authorization, see [URL Authorization Notes](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**Secure Resources**

You should use the same techniques to secure Web resources as presented in Chapter 8, [ASP.NET Security](http://msdn.microsoft.com/en-us/library/cc414309.aspx). In addition, however, for Web services consider removing the HTTP-GET and HTTP-POST protocol from Machine.config on production servers.

**Disable HTTP-GET, HTTP-POST**

By default, clients can communicate with ASP.NET Web services, using three protocols: HTTP-GET, HTTP-POST, and SOAP over HTTP. You should disable support for both the HTTP-GET and HTTP-POST protocols at the machine level on production machines that do not require them. This is to avoid a potential security breach that could allow a malicious Web page to access an internal Web service running behind a firewall.

**Note**   Disabling these protocols means that a new client will not be able to test an XML Web service using the **Invoke** button on the Web service test page. Instead, you must create a test client program by adding a reference to the Web service using Microsoft Visual Studio® .NET development system. You may want to leave these protocols enabled on development computers to allow developers to use the test page.

**To disable the HTTP-GET and HTTP-POST protocols for an entire computer**

1. If you are running .NET Framework 1.1, edit the Machine.config file. If you are running .NET Framework 2.0, edit the machine-level web.config file.
2. If applicable, comment out the lines within the <**webServices**> element that add support for HTTP-GET and HTTP-POST. After doing so, configuration setting should appear as follows.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl33');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl33');)

<webServices>

<protocols>

<add name="HttpSoap"/>

<!-- <add name="HttpPost"/> -->

<!-- <add name="HttpGet"/> -->

<add name="Documentation"/>

</protocols>

</webServices>

1. Save the configuration file.

**Note**   For special cases where you have Web service clients that communicate with a Web service using either HTTP-GET or HTTP-POST, you can add support for those protocols in the application's Web.config file, by creating a <**webServices**> and adding support for these protocols with the <**protocol**> and <**add**> elements, as shown earlier.

**More information**

For detailed Information about securing resources, see [Secure Resources](http://msdn.microsoft.com/en-us/library/aa302388.aspx) within Chapter 8, "ASP.NET Security."

**Secure Communication**

Use a combination of SSL and IPSec to secure communication links.

**More information**

* For information about calling a Web service using SSL, see [How To: Call a Web Service Using SSL from ASP.NET 1.1](http://msdn.microsoft.com/en-us/library/cc394734.aspx) in the Reference section of this guide.
* For information about using IPSec between two computers, see [How To: Use IPSec to Provide Secure Communication Between Two Servers](http://msdn.microsoft.com/en-us/library/aa302413.aspx) in the Reference section of this guide.

**Passing Credentials for Authentication to Web Services**

When you call a Web service, you do so by using a Web service proxy; a local object that exposes the same set of methods as the target Web service.

You can generate a Web service proxy by using the Wsdl.exe command line utility. Alternatively, if you are using Visual Studio .NET you can generate the proxy by adding a Web reference to the project.

**Note**   If the Web service for which you want to generate a proxy is configured to require client certificates, you must temporarily switch off that requirement while you add the reference, or an error occurs. After you add the reference, you must remember to reconfigure the service to require certificates.

An alternate approach would be to keep an offline Web Services Description Language (WSDL) file available to consumer applications. You must remember to update this if your Web service interface changes.

**Specifying Client Credentials for Windows Authentication**

If you are using Windows authentication, you must specify the credentials to be used for authentication using the **Credentials** property of the Web service proxy. If you do not explicitly set this property, the Web service is called without any credentials. If Windows authentication is required, this will result in an HTTP status 401, access denied response.

**Using DefaultCredentials**

Client credentials do not flow implicitly. The Web service consumer must set the credentials and authentication details on the proxy. To flow the security context of the client's Windows security context (either from an impersonating thread token or process token) to a Web service you can set the **Credentials** property of the Web service proxy to **CredentialCache**. **DefaultCredentials** as shown below.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl37');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl37');)

proxy.Credentials = System.Net.CredentialCache.DefaultCredentials;

Consider the following points before you use this approach:

* This flows the client credentials only when you use NTLM, Kerberos, or Negotiate authentication.
* If a client-side application (for example, a Windows Forms application) calls the Web service, the credentials are obtained from the user's interactive logon session.
* Server-side applications—such as ASP.NET Web applications—use the process identity unless impersonation is configured. In that case the impersonated caller's identity is used.

**Using specific credentials**

To use a specific set of credentials for authentication when you call a Web service, use the following code.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl38');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl38');)

CredentialCache cache = new CredentialCache();

cache.Add( new Uri(proxy.Url), // Web service URL

"Negotiate", // Kerberos or NTLM

new NetworkCredential("username", "password", "domainname") );

proxy.Credentials = cache;

In the above example, the requested Negotiate authentication type results in either Kerberos or NTLM authentication.

**Always request a specific authentication type**

You should always request a specific authentication type as illustrated above. Avoid direct use of the **NetworkCredential** class as shown in the following code.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39');)

proxy.Credentials = new

NetworkCredential("username", "password", "domainname");

This should be avoided in production code because you have no control over the authentication mechanism used by the Web service and as a result you have no control over how the credentials are used.

For example, you may expect a Kerberos or NTLM authentication challenge from the server but instead you may receive a Basic challenge. In this case, the supplied user name and password will be sent to the server in clear text form.

**Set the PreAuthenticate property**

The proxy's **PreAuthenticate** property can be set to true or false. Set it to true to supply specific authentication credentials to cause a **WWW-authenticate** HTTP header to be passed with the Web request. This saves the Web server denying access on the request, and performing authentication on the subsequent retry request.

**Note**   Pre-authentication only applies after the Web service successfully authenticates the first time. Pre-authentication has no impact on the first Web request.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl40');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl40');)

private void ConfigureProxy( WebClientProtocol proxy,

string domain, string username,

string password )

{

// To improve performance, force pre-authentication

proxy.PreAuthenticate = true;

// Set the credentials

CredentialCache cache = new CredentialCache();

cache.Add( new Uri(proxy.Url),

"Negotiate",

new NetworkCredential(username, password, domain) );

proxy.Credentials = cache;

proxy.ConnectionGroupName = username;

}

**Using the ConnectionGroupName property**

Notice that the above code sets the **ConnectionGroupName** property of the Web service proxy. This is only required if the security context used to connect to the Web service varies from one request to the next as described below.

If you have an ASP.NET Web application that connects to a Web service and flows the security context of the original caller (by using **DefaultCredentials** or by setting explicit credentials, as shown above), you should set the **ConnectionGroupName** property of the Web service proxy within the Web application. This is to prevent a new, unauthenticated client from reusing an old, authenticated TCP connection to the Web service that is associated with a previous client's authentication credentials. Connection reuse can occur as a result of HTTP KeepAlives and authentication persistence which is enabled for performance reasons within IIS.

Set the **ConnectionGroupName** property to an identifier (such as the caller's user name) that distinguishes one caller from the next as shown in the previous code fragment.

**Note**   If the original caller's security context does not flow through the Web application and onto the Web service, and instead the Web application connects to the Web service using a fixed identity (such as the Web application's ASP.NET process identity), you do not need to set the **ConnectionGroupName** property. In this scenario, the connection security context remains constant from one caller to the next.

**Calling Web Services from Non-Windows Clients**

There are a number of authentication approaches that work for cross-browser scenarios. These include:

* **Certificate Authentication**. This uses cross-platform X.509 certificates.
* **Basic Authentication**. For an example of how to use Basic authentication against a custom data store (without requiring Active Directory), see [Web Services Security - HTTP Basic Authentication without Active Directory](http://www.rassoc.com/gregr/weblog/stories/2002/06/26/webServicesSecurityHttpBasicAuthenticationWithoutActiveDirectory.html).
* **WSE Message Level Approaches**. Use the Web Services Enhancements for Microsoft .NET to implement WS-Security solutions.
* **Custom Approaches**. For example, flow credentials in SOAP headers.

**Proxy Server Authentication**

Proxy server authentication is not supported by the Visual Studio .NET **Add** **Web** **Reference** dialog box (although it will be supported with the next version of Visual Studio .NET). As a result you might receive an HTTP status 407: "Proxy Authentication Required" response when you attempt to add a Web reference.

**Note**   You may not see this error when you view the .asmx file from a browser, because the browser automatically sends credentials.

To work around this issue, you can use the Wsdl.exe command line utility (instead of the **Add** **Web** **Reference** dialog) as shown below.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl42');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl42');)

wsdl.exe /proxy:http://<YourProxy> /pu:<YourName> /pp:<YourPassword> /pd:<YourDomain> http://www.YouWebServer.com/YourWebService/YourService.asmx

If you need to programmatically set the proxy server authentication information, use the following code.

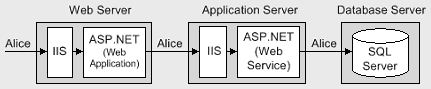
[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl43');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl43');)

YourWebServiceProxy.Proxy.Credentials = CredentialsCache.DefaultCredentials;

**Flowing the Original Caller**

This section describes how you can flow the original caller's security context through an ASP.NET Web application and onto a Web service located on a remote application server. You may need to do this in order to support per-user authorization within the Web service or within subsequent downstream subsystems (for example, databases, where you want to authorize original callers to individual database objects).

In Figure 10.5, the security context of the original caller (Alice) flows through the front-end Web server that hosts an ASP.NET Web application, onto the remote object, hosted by ASP.NET on a remote application server and finally through to a backend database server.



**Figure 10.5. Flowing the original caller's security context**

In order to flow credentials to a Web service, the Web service client (the ASP.NET Web application in this scenario) must configure the Web service proxy and explicitly set the proxy's **Credentials** property, as described in "Passing Credentials for Authentication to Web Services" earlier in this chapter.

There are two ways to flow the caller's context.

* **Pass default credentials and use Kerberos authentication (and delegation)**. This approach requires that you impersonate within the ASP.NET Web application and configure the remote object proxy with **DefaultCredentials** obtained from the impersonated caller's security context.
* **Pass explicit credentials and use Basic or Forms authentication**. This approach does not require impersonation within the ASP.NET Web application. Instead, you programmatically configure the Web service proxy with explicit credentials obtained from either server variables (with Basic authentication) or HTML form fields (with Forms authentication) that are available to the Web application. With Basic or Forms authentication, the user name and password are available to the server in clear text.

**Default Credentials with Kerberos Delegation**

To use Kerberos delegation, all computers (servers and clients) must be running Windows 2000 or later. Additionally, client accounts that are to be delegated must be stored in Active Directory and must not be marked as "Sensitive and cannot be delegated." Additionally, the machine account or custom ASP.NET process identity should be configured in Active Directory as trusted for delegation.

The following tables show the configuration steps required on the Web server, and application server.

**Configuring the Web server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| Disable Anonymous access for your Web application's virtual root directory  Enable Windows Integrated Authentication for the Web application's virtual root | Kerberos authentication will be negotiated assuming clients and server are running Windows 2000 or later. **Note**   If you are using Internet Explorer 6 on Windows 2000, it defaults to NTLM authentication instead of the required Kerberos authentication. To enable Kerberos delegation, see article Q299838, [Unable to Negotiate Kerberos Authentication after upgrading to Internet Explorer 6](http://support.microsoft.com/default.aspx?scid=kb;en-us;Q299838), in the Microsoft Knowledge Base. |
| **Configure ASP.NET** |  |
| **Step** | **More Information** |
| Configure your ASP.NET Web application to use Windows authentication | Edit Web.config in your Web application's virtual directory Set the <**authentication**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl45');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl45');)  <authentication mode="Windows" /> |
| Configure your ASP.NET Web application for impersonation | Edit Web.config in your Web application's virtual directory Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl46');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl46');)  <identity impersonate="true" /> |

|  |  |
| --- | --- |
| **Configure the Web Service Proxy** |  |
| **Step** | **More Information** |
| Set the credentials property of the Web service proxy to **DefaultCredentials**. | See [Using DefaultCredentials](http://msdn.microsoft.com/en-us/library/aa302390.aspx#secnetch10_usingdefaultcreds) earlier in this chapter for a code sample. |

**Configuring the remote application server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| Disable Anonymous access for your Web service's virtual root directory  Enable Windows Integrated Authentication for the Web application's virtual root |  |
| **Configure ASP.NET (Web Service Host)** |  |
| **Step** | **More Information** |
| Configure ASP.NET to use Windows authentication | Edit Web.config in the Web service's virtual directory. Set the <**authentication**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl47');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl47');)  <authentication mode="Windows" /> |
| Configure ASP.NET for impersonation | Edit Web.config in the Web service's virtual directory. Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl48');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl48');)  <identity impersonate="true" />  **Note**   This step is only required if you want to flow the original caller's security context through the Web service and onto the next downstream, subsystem (for example, a database). With impersonation enabled here, resource access (local and remote) uses the impersonated original caller's security context. If your requirement is simply to allow per-user authorization checks in the Web service, you do not need to impersonate here. |

**More information**

For more information about configuring Kerberos delegation, see [How To: Implement Kerberos Delegation for Windows 2000](http://msdn.microsoft.com/en-us/library/cc394676.aspx) in the Reference section of this guide.

**Explicit Credentials with Basic or Forms Authentication**

As an alternative to Kerberos delegation, you can use Basic or Forms authentication at the Web application to capture the client's credentials and then use Basic (or Integrated Windows) authentication to the Web service.

With this approach, the client's clear-text credentials are available to the Web application. These can be passed to the Web service through the Web service proxy. For this, you must write code in the Web application to retrieve the client's credentials and configure the proxy.

**Basic authentication**

With Basic authentication, the original caller's credentials are available to the Web application in server variables. The following code shows how to retrieve them and configure the Web service proxy.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl50');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl50');)

// Retrieve client's credentials (available with Basic

// authentication)

string pwd = Request.ServerVariables["AUTH\_PASSWORD"];

string uid = Request.ServerVariables["AUTH\_USER"];

// Associate the credentials with the Web service proxy

// To improve performance, force preauthentication

proxy.PreAuthenticate = true;

// Set the credentials

CredentialCache cache = new CredentialCache();

cache.Add( new Uri(proxy.Url),

"Basic",

new NetworkCredential(uid, pwd, domain) );

proxy.Credentials = cache;

**Forms authentication**

With Forms authentication, the original caller's credentials are available to the Web application in form fields (rather than server variables). In this case, use the following code.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl51');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl51');)

// Retrieve client's credentials from the logon form

string pwd = txtPassword.Text;

string uid = txtUid.Text;

// Associate the credentials with the Web service proxy

// To improve performance, force preauthentication

proxy.PreAuthenticate = true;

// Set the credentials

CredentialCache cache = new CredentialCache();

cache.Add( new Uri(proxy.Url),

"Basic",

new NetworkCredential(uid, pwd, domain) );

proxy.Credentials = cache;

The following tables show the configuration steps required on the Web server, and application server.

**Configuring the Web server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| To use Basic authentication, disable Anonymous access for your Web application's virtual root directory and select Basic authentication  - or -  To use Forms authentication, enable anonymous access | Both Basic and Forms authentication should be used in conjunction with SSL to protect the clear text credentials sent over the network. If you use Basic authentication, SSL should be used for all pages (not just the initial logon page), as Basic credentials are transmitted with every request.     If you use Forms authentication, SSL should be used for all secure pages to protect the clear text credentials on the initial logon and to protect the authentication ticket passed on subsequent requests. The authentication cookie should be configured in machine.config or web.config to require SSL. |
| **Configure ASP.NET** |  |
| **Step** | **More Information** |
| If you use Basic authentication, configure your ASP.NET Web application to use Windows authentication  - or -  If you use Forms authentication, configure your ASP.NET Web application to use Forms authentication | Edit Web.config in your Web application's virtual directory Set the <**authentication**> element to:  <authentication mode="Windows" />    - or -  Edit Web.config in your Web application's virtual directory Set the <**authentication**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl52');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl52');)  <authentication mode="Forms" /> |
| Disable impersonation within the ASP.NET Web application | Edit Web.config in your Web application's virtual directory. Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl53');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl53');)  <identity impersonate="false" />  **Note**   This is equivalent to having no <**identity**> element. Impersonation is not required, as the user's credentials will be passed explicitly to the Web service through the proxy. |

|  |  |
| --- | --- |
| **Configure the Web Service Proxy** |  |
| **Step** | **More Information** |
| Write code to capture and explicitly set the credentials on the Web Service proxy | Refer to the code fragments shown earlier in the Basic Authentication and Forms Authentication sections. |

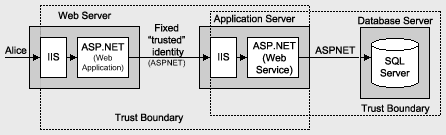
**Configuring the application server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| Disable Anonymous access for your application's virtual root directory  Enable Basic authentication | **Note**   Basic authentication at the (Web service) application server, allows the Web service to flow the original caller's security context to the database (as the caller's user name and password are available in clear text and can be used to respond to network authentication challenges from the database server). If you don't need to flow the original caller's security context beyond the Web service, consider configuring IIS at the application server to use Windows Integrated authentication, as this provides tighter security—credentials are not passed across the network and are not available to the Web service. |
| **Configure ASP.NET (Web Service)** |  |
| **Step** | **More Information** |
| Configure ASP.NET to use Windows authentication | Edit Web.config in the Web service's virtual directory. Set the <**authentication**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl54');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl54');)  <authentication mode="Windows" /> |
| Configure your ASP.NET Web service for impersonation | Edit Web.config in the Web service's virtual directory. Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55');)  <identity impersonate="true" />  **Note**   This step is only required if you want to flow the original caller's security context through the Web service and onto the next downstream, subsystem (for example, a database). With impersonation enabled here, resource access (local and remote) uses the impersonated original caller's security context. If your requirement is simply to allow per-user authorization checks in the Web service, you do not need to impersonate here. |

**Trusted Subsystem**

The trusted subsystem model provides an alternative (and simpler to implement) approach to flowing the original caller's security context. In this model a trust boundary exists between the Web service and Web application. The Web service trusts the Web application to properly authenticate and authorize callers, prior to letting requests proceed to the Web service. No authentication of the original callers occurs at the Web service. The Web service authenticates the fixed trusted identity used by the Web application to communicate with the Web service. In most cases, this is the process identity of the ASP.NET worker process.

The trusted subsystem model is shown in Figure 10.6.



**Figure 10.6. The trusted subsystem model**

**Flowing the Caller's Identity**

If you use the trusted subsystem model, you may still need to flow the original caller's identity (name, not security context), for example, for auditing purposes at the database.

You can flow the identity at the application level by using method and stored procedure parameters and trusted query parameters (as shown in the following example) can be used to retrieve user-specific data from the database.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56');)

SELECT x,y,z FROM SomeTable WHERE UserName = "Alice"

**Configuration Steps**

The following tables show the configuration steps required on the Web server, and application server.

**Configuring the web server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| Configure IIS authentication | The Web application can use any form of authentication to authenticate the original callers. |
| **Configure ASP.NET** |  |
| **Step** | **More Information** |
| Configure authentication and make sure impersonation is disabled | Edit Web.config in your Web application's virtual directory. Set the <**authentication**> element to "Windows", "Forms" or "Passport."  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl57');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl57');)  <authentication mode="Windows|Forms|Passport" />  Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl58');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl58');)  <identity impersonate="false" />  (or remove the <**identity**> element) |
| Reset the password of the ASPNET account used to run ASP.NET OR create a least privileged domain account to run ASP.NET and specify account details on the <**processModel**> element within Web.config | For more information about how to access network resources (including Web services) from ASP.NET and about choosing and configuring a process account for ASP.NET, see [Accessing Network Resources](http://msdn.microsoft.com/en-us/library/aa302388.aspx) and [Process Identity for ASP.NET](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security." |

|  |  |
| --- | --- |
| **Configure Web Service Proxy** |  |
| **Step** | **More Information** |
| Configure the Web service proxy to use default credentials for all calls to the Web service | Use the following line of code:  proxy.Credentials = DefaultCredentials; |

**Configuring the application server**

|  |  |
| --- | --- |
| **Configure IIS** |  |
| **Step** | **More Information** |
| Disable Anonymous access for your Web service's virtual root directory  Enable Windows Integrated authentication |  |
| **Configure ASP.NET** |  |
| **Step** | **More Information** |
| Configure ASP.NET to use Windows authentication | Edit Web.config in the Web service's virtual directory. Set the <**authentication**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl61');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl61');)  <authentication mode="Windows" /> |
| Disable impersonation | Edit Web.config in the application's virtual directory. Set the <**identity**> element to:  [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62');)  <identity impersonate="false" /> |

**Accessing System Resources**

For details about accessing system resources (for example the event log and the registry) from ASP.NET Web services, see [Accessing System Resources](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security." The approaches and restrictions discussed in Chapter 8 also apply to ASP.NET Web services.

**Accessing Network Resources**

When you access network resources from a Web service, you need to consider the identity that is used to respond to network authentication challenges from the remote computer. You have three options:

* The process identity (determined by the account used to run the ASP.NET worker process)
* A service identity (for example, one created by calling **LogonUser**)
* The original caller identity (with the Web service configured for impersonation)

For details about the relative merits of each of these approaches, together with configuration details, see [Accessing Network Resources](http://msdn.microsoft.com/en-us/library/aa302388.aspx) in Chapter 8, "ASP.NET Security."

**Accessing COM Objects**

The **AspCompat** directive (used by Web applications when they call apartment threaded COM objects) is not available to Web services. This means that when you call apartment model objects from Web services, a thread switch occurs. This results in a slight performance hit, and if your Web service is impersonating, your impersonation token will be lost when calling the COM component. This typically results in an Access Denied exception.

**More Information**

* For more information about access denied exceptions when calling apartment threaded COM objects, see article Q325791, [PRB: Access Denied Error Message Occurs When Impersonating in ASP.NET and Calling STA COM Components](http://support.microsoft.com/default.aspx?scid=kb;en-us;Q325791), in the Microsoft Knowledge Base.
* For more information about accessing COM objects and using the AspCompat attribute, see [Accessing COM Objects](http://msdn.microsoft.com/en-us/library/aa302388.aspx) within Chapter 8, "ASP.NET Security."
* For more information about calling apartment threaded COM objects from Web services, see article Q303375, [INFO: XML Web Services and Apartment Objects](http://support.microsoft.com/default.aspx?scid=kb;en-us;Q303375), in the Microsoft Knowledge Base.

**Using Client Certificates with Web Services**

This section describes techniques for using X.509 client certificates for Web service authentication.

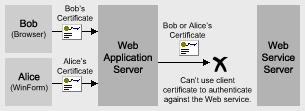
You can use client certificate authentication within a Web service to authenticate:

* Other Web services
* Applications that communicate directly with the Web service (for example, server-based or client-side desktop applications)

**Authenticating Web Browser Clients with Certificates**

A Web service cannot use client certificates to authenticate callers if they interact with an intermediate Web application, because it is not possible to forward the original caller's certificate through the Web application and onto the Web service. While the Web application can authenticate its clients with certificates, the same certificates cannot then be used by the Web service for authentication.

The reason that this server-to-server scenario fails is that the Web application does not have access to the client's certificate (specifically its private key) in its certificate store. This problem is illustrated in Figure 10.7.

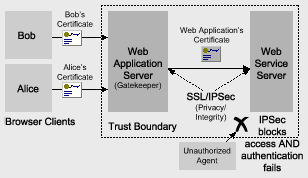


**Figure 10.7. Web service client certificate authentication**

**Using the Trusted Subsystem Model**

To address this restriction, and to support certificate authentication at the Web service, you must use a trusted-subsystem model. With this approach, the Web service authenticates the Web application using the Web application's certificate (and not the original caller's certificate). The Web service must trust the Web application to authenticate its users and to perform the necessary authorization to ensure that only authorized callers are able to access the data and functionality exposed by the Web service.

This approach is shown in Figure 10.8.



**Figure 10.8. The Web service authenticates the trusted Web application**

If the authorization logic within the Web service requires multiple roles, the Web application can send different certificates based upon the role membership of the caller. For example, one certificate may be used for members of an Administrators group (who are allowed to update data within a back-end database) and another certificate may be used for all other users (who are authorized only for read operations).

**Note**   In scenarios such as these, a local certificate server (accessible only by the two servers) can be used to manage all the Web application certificates.

In this scenario:

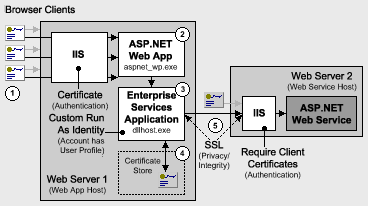
* The Web application authenticates its users by using client certificates.
* The Web application acts as a gatekeeper and authorizes its users and controls access to the Web service.
* The Web application calls the Web service and passes a different certificate that represents the application (or possibly a range of certificates based on the role membership of the caller).
* The Web service authenticates the Web application and it trusts the application to perform the necessary client authorization.
* IPSec is used between the Web application server and Web service server to provide additional access control. Unauthorized access attempts from other computers are prevented by IPSec. Certificate authentication at the Web service server also prevents unauthorized access.

**Solution implementation**

To use certificate authentication at the Web service in this scenario, use a separate process to call the Web service and pass the certificate. You cannot manipulate the certificates directly from the ASP.NET Web application because it does not have a loaded user profile and associated certificate store. The separate process must be configured to run using an account that has an associated user profile (and certificate store). You have two main options:

* You can use an Enterprise Services server application.
* You can use a Windows service.

Figure 10.9 illustrates this scenario with an Enterprise Services server application.



**Figure 10.9. Client certificate authentication with Web services**

The following summarizes the sequence of events illustrated by Figure 10.9:

1. The original caller is authenticated by the Web application using client certificates.
2. The Web application is the gatekeeper and is responsible for authorizing access to specific areas of functionality (including those that involve interaction with the Web service).
3. The Web application calls a serviced component running in an out-of-process Enterprise Services application.
4. The account used to run the Enterprise Services application has an associated user profile. The component accesses a client certificate from its certificate store, which is used by the Web service to authenticate the Web application.
5. The serviced component calls the Web service, passing the client certificate on each method request. The Web service authenticates the Web application using this certificate and trusts the Web application to correctly authorize original callers.

**Why use an additional process?**

An additional process is required (rather than using the ASP.NET worker process to contact the Web service) due to the fact that a user profile (containing a certificate store) is required.

A Web application that runs using the ASP.NET process identity account does not have access to any certificates on the Web server. This is because certificate stores are maintained within user profiles associated with interactive user accounts. User profiles are only created for interactive accounts when you physically log on using the account. The ASP.NET process identity account is not intended to be an interactive user account and is configured with the "Deny interactive logon" privilege for added security.

**Important**   Do not reconfigure the ASP.NET process identity account to remove this privilege and turn the account into an interactive logon account. Use a separate process with a configured service account to access certificates, as described earlier in this chapter.

**More information**

* For more information about how to implement this approach, see [How To: Call a Web Service Using Client Certificates from ASP.NET](http://msdn.microsoft.com/en-us/library/cc394727.aspx) in the Reference section of this guide.
* For more information about configuring IPSec, see [How To: Use IPSec to Provide Secure Communication Between Two Servers](http://msdn.microsoft.com/en-us/library/aa302413.aspx) in the Reference section of this guide.

**Secure Communication**

Secure communication is concerned with guaranteeing the integrity and confidentiality of Web service messages as they flow from application to application across the network. There are two approaches to this problem: transport-level options and message-level options.

**Transport Level Options**

Transport-level options include:

* SSL
* IPSec

These options may be appropriate if the following conditions are met:

* You are sending a message directly from your application to a Web service and the message will not be routed through intermediate systems.
* You can control the configuration of both endpoints involved in the message transfer.

**Message-Level Options**

Message-level approaches can be used to guarantee the confidentiality and integrity of messages as they pass through an arbitrary number of intermediate systems. Messages can be signed to provide integrity. For confidentiality, you can choose between encrypting the entire message or part of a message.

Use a message-level approach if the following conditions are met:

* You are sending a message to a Web service, and the message is likely to be forwarded to other Web services or may be routed through intermediate systems.
* You do not control the configuration of both endpoints. For example, if you are sending messages from one company to another.

**More information**

* The Web Service Development Toolkit will provide message encryption functionality in accordance with the WS-Security specification.
* For more information about SSL and IPSec, see Chapter 4, [Secure Communication](http://msdn.microsoft.com/en-us/library/cc414283.aspx).

**Summary**

This chapter has focused on platform/transport-level (point-to-point) Web service security provided by the underlying services of ASP.NET, IIS, and the operating system. While platform-level security provides secure solutions for tightly-coupled intranet scenarios, it is not suited to heterogeneous scenarios. For this, message-level security provided by the WS-Security specification is required. Use the Web Services Enhancements for Microsoft .NET (WSE) to build message-level Web service security solutions.

For tightly-coupled Windows domain environments:

* If you want to flow the original caller's identity from an ASP.NET Web application to a remote Web service, the ASP.NET Web application should use Kerberos authentication (with accounts configured for delegation), Basic authentication, or Forms authentication.
  + With Kerberos authentication, enable impersonation with the Web application and configure the **Credentials** property of the Web service proxy using **DefaultCredentials**.
  + With Basic or Forms authentication, capture the caller's credentials and set the **Credentials** property of the Web service proxy by adding a new **CredentialCache** object.

For Web-service to Web-service scenarios:

* Use Basic or Kerberos authentication and set credentials in the client proxy.
* Use an out of process Enterprise Services application or a Windows service to manipulate X.509 certificates from Web applications.
* As far as possible, use system-level authorization checks such as File and URL authorization.
* For granular authorization (for example, at the Web method level) use .NET roles (declaratively and imperatively).
* Authorize non-Windows users by using .NET roles (based on a **GenericPrincipal** object that contains roles).
* Disable HTTP-GET and HTTP-POST protocols on product servers.
* Use transport-level security if you are not worried about passing messages securely through intermediary systems.
* Use transport-level security if SSL performance is acceptable.
* Use WS-Security and WSE to develop message-level solutions.

[](http://www.msdn.microsoft.com/practices)