**ASP.NET Web Application Security**

**.NET Framework 4**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/330a99hc(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/330a99hc(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/330a99hc(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/330a99hc(d=printer,v=vs.71).ASPX)

ASP.NET, in conjunction with Microsoft Internet Information Services (IIS), can authenticate user credentials such as names and passwords using any of the following authentication methods:

* Windows: Basic, digest, or Integrated Windows Authentication (NTLM or Kerberos).
* Forms authentication, in which you create a login page and manage authentication in your application.
* Client Certificate authentication

ASP.NET controls access to site information by comparing authenticated credentials, or representations of them, to NTFS file system permissions or to an XML file that lists authorized users, authorized roles (groups), or authorized HTTP verbs.

This section contains topic that describe the specifics of ASP.NET security.

**How ASP.NET Security Works**

**.NET Framework 4**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ks310b8y(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ks310b8y(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ks310b8y(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/ks310b8y(d=printer,v=vs.71).ASPX)

Securing Web sites is a critical, complex issue for Web developers. Protecting a site requires careful planning, and Web site administrators and programmers must have a clear understanding of the options for securing their site.

ASP.NET works in concert with the Microsoft .NET Framework and Microsoft Internet Information Services (IIS) to help provide Web application security. To help protect your ASP.NET application, you should perform the two fundamental functions described in the following table.

|  |  |
| --- | --- |
| **Security function** | **Description** |
| [Authentication](http://msdn.microsoft.com/en-us/library/eeyk640h.ASPX) | Helps to verify that the user is, in fact, who the user claims to be. The application obtains credentials (various forms of identification, such as name and password) from a user and validates those credentials against some authority. If the credentials are valid, the entity that submitted the credentials is considered an authenticated identity. |
| [Authorization](http://msdn.microsoft.com/en-us/library/wce3kxhd.ASPX) | Limits access rights by granting or denying specific permissions to an authenticated identity. |

IIS can also grant or deny access based on a user's host name or IP address. Any further access authorization is performed by NTFS file access permission's URL authorization.

It is helpful to understand how all the various security subsystems interact. Since ASP.NET is built on the Microsoft .NET Framework, the ASP.NET application developer also has access to all the built-in security features of the .NET Framework, such as code access security and role-based user-access security. For details about the security capabilities of ASP.NET, see [ASP.NET Code Access Security](http://msdn.microsoft.com/en-us/library/87x8e4d1.ASPX).

**ASP.NET Security Architecture**

**.NET Framework 4**

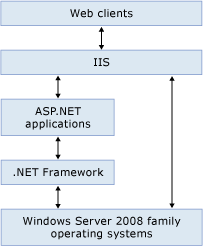
[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/yedba920(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/yedba920(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/yedba920(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/yedba920(d=printer,v=vs.71).ASPX)

This section provides an overview of the ASP.NET security infrastructure. The following illustration shows the relationships among the security systems in ASP.NET.

ASP.NET architecture



As the illustration shows, all Web clients communicate with ASP.NET applications through Microsoft Internet Information Services (IIS). IIS authenticates the request if required and then locates the requested resource (such as an ASP.NET application). If the client is authorized, the resource is made available.

When an ASP.NET application is running, it can use built-in ASP.NET security features. In addition, an ASP.NET application can use the security features of the .NET Framework. For more information, see [Key Security Concepts](http://msdn.microsoft.com/en-us/library/z164t8hs.ASPX).

[Integrating ASP.NET Authentication with IIS](javascript:void(0))

In addition to relying on the authentication capabilities of IIS, you can perform authentication in ASP.NET. When considering ASP.NET authentication, you should understand the interaction with IIS authentication services.

IIS assumes that a set of credentials maps to a Microsoft Windows NT account and that it should use those credentials to authenticate a user. The authentication methods used in IIS 7 are the following: anonymous, ASP.NET impersonation, basic, client certificate mapping, digest, forms, and Windows Integrated Security (NTLM or Kerberos). You can select the type of authentication by using IIS administrative services. For information, see [Configuring Authentication in IIS 7](http://go.microsoft.com/fwlink/?LinkId=198416).

If users request a URL that maps to an ASP.NET application, the request and authentication information are handed off to the application. ASP.NET provides forms authentication. Forms authentication is a system by which unauthenticated requests are redirected to an ASP.NET Web page that you create. The user provides credentials and submits the page. If your application authenticates the request, the system issues an authentication ticket in a cookie that contains the credentials or a key for reacquiring the identity. Subsequent requests include an authentication ticket with the request.

|  |
| --- |
| **Description: NoteNote** |
| ASP.NET membership and ASP.NET login controls implicitly work with forms authentication. |

[ASP.NET Configuration File Security Settings](javascript:void(0))

ASP.NET security settings are configured in the Machine.config and Web.config files. As with other configuration information, base settings and default settings are established in the Machine.config file in the Config subdirectory of the current .NET Framework installation. You can establish site-specific and application-specific settings (including overriding settings from the Machine.config file) in Web.config files in the Web site root and application root directories. Subdirectories inherit a directory's settings unless overridden by a Web.config file in the subdirectory. To see an example of the way in which the hierarchical configuration system works for security, see [configSections Element (General Settings Schema)](http://msdn.microsoft.com/en-us/library/ms228256.ASPX).

There are three major subsections to a Web.config file: the [authentication](http://msdn.microsoft.com/en-us/library/532aee0e.ASPX), [authorization](http://msdn.microsoft.com/en-us/library/8d82143t.ASPX), and [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) sections. The values for each security element are usually set in the Machine.config file and overridden as required in the application-level Web.config file. All subdirectories automatically inherit those settings. However, subdirectories can have their own configuration files that override inherited settings.

|  |
| --- |
| **Description: NoteNote** |
| ASP.NET configuration applies only to ASP.NET resources, namely those registered to be handled in IIS by the Aspnet\_isapi.dll extension. ASP.NET configuration cannot provide authorization for resources not processed by ASP.NET. Therefore, .txt, .htm, .html, .gif, .jpg, .jpeg, .asp, and other types of files are accessible by all users (subject to IIS permissions). For example, even though the ASP.NET resources in a directory might be restricted by a Web.config file, all users can still view the files located in that directory if directory browsing is turned on and no other restrictions are in place. You can put these types of files under ASP.NET security by explicitly mapping such file name extensions to the Aspnet\_isapi.dll extension using the IIS administration tool. However, processing these types of files through ASP.NET can affect the performance of the Web site. For more information about how to secure files in a folder, see [How to: Configure Specific Directories Using Location Settings](http://msdn.microsoft.com/en-us/library/ms178692.ASPX). |

You can use the [location](http://msdn.microsoft.com/en-us/library/b6x6shw7.ASPX) configuration element to specify a particular file or directory to which settings should apply. For more information, see [configSections Element (General Settings Schema)](http://msdn.microsoft.com/en-us/library/ms228256.ASPX) and [Configuring Specific Files and Subdirectories](http://msdn.microsoft.com/en-us/library/6hbkh9s7.ASPX). For more details about ASP.NET configuration in general, see [ASP.NET Configuration Overview](http://msdn.microsoft.com/en-us/library/ms178683.ASPX).

The following example shows the syntax of the security sections of a configuration file:

<authentication mode="[Windows|Forms|None]">

<forms name="name"

loginUrl="url"

protection="[All|None|Encryption|Validation]"

timeout="minutes"

path="path"

requireSSL="[true|false]"

slidingExpiration="[true|false]">

defaultUrl="string"

cookieless="[UseCookies|UseUri|AutoDetect|UseDeviceProfile]"

domain="string"

<credentials passwordFormat="[Clear|MD5|SHA1]">

<user name="\*\*\*\*\*\*\*\*"

password="\*\*\*\*\*\*\*\*"/>

</credentials>

</forms>

</authentication>

<authorization>

<allow users="comma-separated list of users"

roles="comma-separated list of roles"

verbs="comma-separated list of verbs" />

<deny users="comma-separated list of users"

roles="comma-separated list of roles"

verbs="comma-separated list of verbs" />

</authorization>

<identity impersonate ="[true|false]"

userName="domain\username"

password="password" />

<trust level="[Full|High|Medium|Low|Minimal]"

originUrl=""/>

<securityPolicy>

<trustLevel name="Full" policyFile="internal"/>

<trustLevel name="High" policyFile="web\_hightrust.config"/>

<trustLevel name="Medium" policyFile="web\_mediumtrust.config"/>

<trustLevel name="Low" policyFile="web\_lowtrust.config"/>

<trustLevel name="Minimal" policyFile="web\_minimaltrust.config"/>

</securityPolicy>

The default settings for these elements are shown in the following table.

|  |  |
| --- | --- |
| **Default value** | **Description** |
| <allow roles="" /> | An empty string indicating that all roles are allowed by default. |
| <allow users="\*" /> | An empty string indicating that all users are allowed access (no authentication is required). |
| <allow verbs="" /> | An empty string that indicates that no verbs are assigned by default. |
| <authentication mode="Windows" /> | The authentication type that determines the source of the current [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user.ASPX) value. The default is Windows. |
| <credentials passwordFormat="SHA1" /> | The hashing algorithm used on passwords. The default is [SHA1](http://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1.ASPX). |
| <deny roles="" /> | An empty string indicating that no roles are denied by default. |
| <deny users="" /> | An empty string indicating that no users are denied by default. |
| <deny verbs="" /> | An empty string that indicates that no verbs are assigned by default. |
| <forms cookieless="UseDeviceProfile" /> | The method to use to store the forms authentication ticket on the client. Valid values are UseCookies, UseUri, AutoDetect, UseDeviceProfile (default). |
| <forms defaultUrl="default.aspx" /> | A string that indicates the URL of the page to redirect to after login. |
| <forms domain="" /> | An empty string that indicates that no domain has been specified for the cookie. |
| <forms loginUrl="logon.aspx" /> | The URL to which the request is directed if you set the authentication mode to Forms and if the request does not have a valid authentication ticket. |
| <forms name=".ASPXAUTH" /> | The name under which the forms authentication cookie is stored on the user's computer. |
| <forms path="/" /> | The path to which forms authentication applies. The default is all paths from the application root down. |
| <forms protection="All" /> | The security applied to the forms authentication ticket. Values include: All, None, Encryption, and Validation. |
| <forms timeout="30" /> | The timeout in minutes before the forms authentication ticket expires and users must re-authenticate. |
| <forms requireSSL="false" /> | A Boolean value indicating whether an SSL connection is required to transmit the authentication cookie. |
| <forms slidingExpiration="true" /> | A Boolean value indicating whether sliding expiration is enabled. For more information, see the [SlidingExpiration](http://msdn.microsoft.com/en-us/library/system.web.security.formsauthentication.slidingexpiration.ASPX) property. |
| <identity impersonate="false" /> | A Boolean value indicating whether impersonation is disabled. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX). |
| <identity userName="" /> | An empty string indicating that no user identity is specified by default. |
| <identity password="" /> | An empty string indicating that no password for the user identity is specified by default. |
| <trust level="Full" originUrl="" /> | The security policy that will be applied to the application. |
| <trustLevel name="Full" policyFile="internal"/> | The default policy file for Full trust level. |
| <trustLevel name="High" policyFile="web\_hightrust.config"/> | The default policy file for High trust level. |
| <trustLevel name="Medium" policyFile="web\_mediumtrust.config"/> | The default policy file for Medium trust level. |
| <trustLevel name="Low" policyFile="web\_lowtrust.config"/> | The default policy file for Low trust level. |
| <trustLevel name="Minimal" policyFile="web\_minimaltrust.config"/> | The default policy file for Minimal trust level. |

**ASP.NET Security Data Flow**

**.NET Framework 4**

[Other Versions](javascript:;)



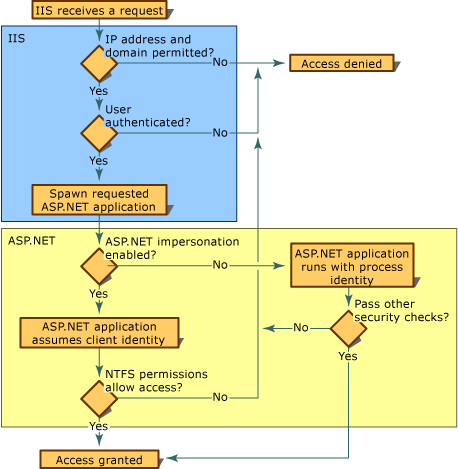
* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/xa68twcb(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/xa68twcb(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/xa68twcb(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/xa68twcb(d=printer,v=vs.71).ASPX)

You can design security into ASP.NET applications in a number of ways. This topic describes the security data flow for two common scenarios: impersonation and forms authentication using cookies.

[Scenario 1: Impersonation](javascript:void(0))

The impersonation scenario relies on Microsoft Internet Information Services (IIS) authentication and Microsoft Windows file access security to minimize security programming in the ASP.NET application itself. The data flow is shown in the following illustration.

Impersonation



The illustration shows the following sequence of events:

1. A request from a network client comes to IIS.
2. IIS authenticates the client using basic, digest, or Windows integrated security (NTLM or Kerberos).
3. If the client is authenticated, IIS passes the authenticated request to ASP.NET.
4. The ASP.NET application impersonates the requesting client using the access token passed from IIS, and it relies on NTFS file permissions for granting access to resources. The ASP.NET application needs only to verify that impersonation is set to true in the ASP.NET configuration file; no ASP.NET security code is required.

If impersonation is not enabled, the application runs with the ASP.NET process identity. For Microsoft Windows 2000 Server and Windows XP Professional, the default identity is a local account named ASPNET that is created automatically when ASP.NET is installed. For Microsoft Windows Server 2003, the default identity is the identity of the application pool for the IIS application (by default, the NETWORK SERVICE account.)

|  |
| --- |
| **NoteNote** |
| If impersonation is not enabled and you want to restrict access for a particular user or set of users, such as users authenticated using forms authentication, then you must use some other means of authorization, such as URL authorization. For more information on URL authorization, see [ASP.NET Authorization](http://msdn.microsoft.com/en-us/library/wce3kxhd.ASPX). |

For more details about using impersonation in ASP.NET applications, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX) and [Using IIS Authentication with ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/134ec8tc.ASPX).

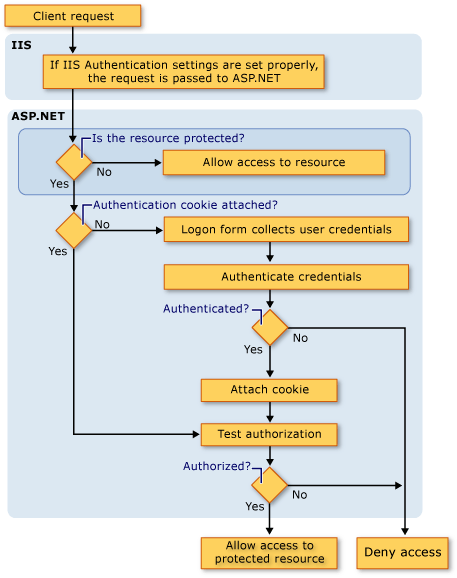
1. If access is granted, the ASP.NET application returns the requested resource through IIS.

[Scenario 2 - Forms Authentication](javascript:void(0))

In the forms authentication scenario, an application collects credentials such as name and password directly from the user and makes its own determination about their authenticity. IIS authentication is not used by the application, but IIS authentication settings can affect forms authentication. As a rule, when you use forms authentication, you enable anonymous access in IIS. Otherwise, if users do not pass IIS authentication, they do not reach your application in order to provide a user name and password to forms authentication.

The data flow in this scenario is shown in the following illustration.

Forms authentication



This illustration shows the following sequence of events:

1. A user generates a request for a protected resource.
2. IIS receives the request, and because IIS anonymous access is enabled, IIS does not perform any user authentication and the request is passed to the ASP.NET application.
3. Because the ASP.NET authentication mode is set to forms, the ASP.NET application examines the request for a forms authentication ticket (a specific cookie). If there is no authentication ticket attached to the request, ASP.NET redirects the request to the logon page specified in the application's configuration file.
4. On the logon page, the user enters the required credentials, usually a name and password. The application code checks the credentials to confirm their authenticity. If the credentials are authenticated, the application code attaches an authentication ticket to the response that represents the user credentials. (The password is not included). If authentication fails, the response is returned with an access denied message or the logon form is presented again.

The authentication ticket that is issued is included with subsequent requests to the ASP.NET application. ASP.NET checks the ticket for validity using a message authentication check (MAC).

1. If the user is authenticated, ASP.NET checks authorization and can either allow access to the originally requested resource, redirect the request to some other page, or redirect the request to a custom authorization module where the credentials are tested for authorization to access the protected resource. If authorization fails, ASP.NET redirects the user to the logon page.

If the user is authorized, access is granted to the protected resource; or the application might require an additional test of the credentials before authorizing access to the protected resource, depending on the design of the application.

|  |
| --- |
| **NoteNote** |
| Forms authentication and authorization checking apply only to resources protected by the [authentication](http://msdn.microsoft.com/en-us/library/532aee0e.ASPX) and [authorization](http://msdn.microsoft.com/en-us/library/8d82143t.ASPX) configuration elements. Access to Windows resources protected using Access Control Lists (ACLs) is checked against the current Windows identity of the ASP.NET application. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX). |

**ASP.NET Authentication**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/eeyk640h(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/eeyk640h(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/eeyk640h(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/eeyk640h(d=printer,v=vs.71).ASPX)

Authentication is the process of obtaining identification credentials such as name and password from a user and validating those credentials against some authority. If the credentials are valid, the entity that submitted the credentials is considered an authenticated identity. Once an identity has been authenticated, the authorization process determines whether that identity has access to a given resource.

ASP.NET implements authentication through authentication providers, the code modules that contain the code necessary to authenticate the requestor's credentials. The topics in this section describe the authentication providers built into ASP.NET.

[In This Section](javascript:void(0))

|  |  |
| --- | --- |
| **Term** | **Definition** |
| [Windows Authentication Provider](http://msdn.microsoft.com/en-us/library/907hb5w9.ASPX) | Provides information on how to use Windows authentication in conjunction with Microsoft Internet Information Services (IIS) authentication to secure ASP.NET applications. |
| [Forms Authentication Provider](http://msdn.microsoft.com/en-us/library/9wff0kyh.ASPX) | Provides information on how to create an application-specific login form and perform authentication using your own code. A convenient way to work with forms authentication is to use ASP.NET membership and ASP.NET login controls, which together provide a way to collect user credentials, authenticate them, and manage them, using little or no code. For more information, see [Managing Users by Using Membership](http://msdn.microsoft.com/en-us/library/tw292whz.ASPX) and [ASP.NET Login Controls Overview](http://msdn.microsoft.com/en-us/library/ms178329.ASPX). |

You might also consider using Windows Live ID for user authentication. For information about how to use Windows Live ID to authenticate users for you website, see [Windows Live ID SDK](http://go.microsoft.com/fwlink/?LinkId=106346).

**ASP.NET Authorization**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/wce3kxhd(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/wce3kxhd(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/wce3kxhd(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/wce3kxhd(d=printer,v=vs.71).ASPX)

Authorization determines whether an identity should be granted access to a specific resource. In ASP.NET, there are two ways to authorize access to a given resource:

* File authorization   File authorization is performed by the [FileAuthorizationModule](http://msdn.microsoft.com/en-us/library/system.web.security.fileauthorizationmodule.ASPX). It checks the access control list (ACL) of the .aspx or .asmx handler file to determine whether a user should have access to the file. ACL permissions are verified for the user's Windows identity (if Windows authentication is enabled) or for the Windows identity of the ASP.NET process. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX).
* URL authorization   URL authorization is performed by the [UrlAuthorizationModule](http://msdn.microsoft.com/en-us/library/system.web.security.urlauthorizationmodule.ASPX), which maps users and roles to URLs in ASP.NET applications. This module can be used to selectively allow or deny access to arbitrary parts of an application (typically directories) for specific users or roles.

[Using URL Authorization](javascript:void(0))

With URL authorization, you explicitly allow or deny access to a particular directory by user name or role. To do so, you create an authorization section in the configuration file for that directory. To enable URL authorization, you specify a list of users or roles in the [allow](http://msdn.microsoft.com/en-us/library/acsd09b0.ASPX) or [deny](http://msdn.microsoft.com/en-us/library/8aeskccd.ASPX) elements of the [authorization](http://msdn.microsoft.com/en-us/library/8d82143t.ASPX) section of a configuration file. The permissions established for a directory also apply to its subdirectories, unless configuration files in a subdirectory override them.

The following shows the syntax for the authorization section:

<authorization>

<[allow|deny] users roles verbs />

</authorization>

The allow or deny element is required. You must specify either the users or the roles attribute. Both can be included, but both are not required. The verbs attribute is optional.

The allow and deny elements grant and revoke access, respectively. Each element supports the attributes shown in the following table:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| users | Identifies the targeted identities (user accounts) for this element.  Anonymous users are identified using a question mark (?). You can specify all authenticated users using an asterisk (\*). |
| roles | Identifies a role (a [RolePrincipal](http://msdn.microsoft.com/en-us/library/system.web.security.roleprincipal.ASPX) object) for the current request that is allowed or denied access to the resource. For more information, see [Managing Authorization Using Roles](http://msdn.microsoft.com/en-us/library/9ab2fxh0.ASPX). |
| verbs | Defines the HTTP verbs to which the action applies, such as GET, HEAD, and POST. The default is "\*", which specifies all verbs. |

The following example grants access to the Kim identity and members of the Admins role, and denies access to the John identity (unless the John identity is included in the Admins role) and to all anonymous users:

<authorization>

<allow users="Kim"/>

<allow roles="Admins"/>

<deny users="John"/>

<deny users="?"/>

</authorization>

The following authorization section shows how to allow access to the John identity and deny access to all other users:

<authorization>

<allow users="John"/>

<deny users="\*"/>

</authorization>

You can specify multiple entities for both the users and roles attributes by using a comma-separated list, as shown in the following example:

<allow users="John, Kim, contoso\Jane"/>

Note that if you specify a domain account name, the name must include both the domain and user name (contoso\Jane).

The following example allows all users to perform an HTTP GET for a resource, but allows only the Kim identity to perform a POST operation:

<authorization>

<allow verbs="GET" users="\*"/>

<allow verbs="POST" users="Kim"/>

<deny verbs="POST" users="\*"/>

</authorization>

Rules are applied as follows:

* Rules contained in application-level configuration files take precedence over inherited rules. The system determines which rule takes precedence by constructing a merged list of all rules for a URL, with the most recent rules (those nearest in the hierarchy) at the head of the list.
* Given a set of merged rules for an application, ASP.NET starts at the head of the list and checks rules until the first match is found. The default configuration for ASP.NET contains an <allow users="\*"> element, which authorizes all users. (By default, this rule is applied last.) If no other authorization rules match, the request is allowed. If a match is found and the match is a deny element, the request is returned with the 401 HTTP status code. If an allow element matches, the module allows the request to be processed further.

In a configuration file, you can also create a [location](http://msdn.microsoft.com/en-us/library/b6x6shw7.ASPX) element to specify a particular file or directory to which settings in that the location element should apply.

**ASP.NET Impersonation**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.71).ASPX)

When using impersonation, ASP.NET applications can execute with the Windows identity (user account) of the user making the request. Impersonation is commonly used in applications that rely on Microsoft Internet Information Services (IIS) to authenticate the user.

ASP.NET impersonation is disabled by default. If impersonation is enabled for an ASP.NET application, that application runs in the context of the identity whose access token IIS passes to ASP.NET. That token can be either an authenticated user token, such as a token for a logged-in Windows user, or the token that IIS provides for anonymous users (typically, the IUSR\_MACHINENAME identity).

When impersonation is enabled, only your application code runs under the context of the impersonated user. Applications are compiled and configuration information is loaded using the identity of the ASP.NET process. For more information, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.ASPX). The compiled application is put in the Temporary ASP.NET files directory. The application identity that is being impersonated needs to have read/write access to this directory. The impersonated application identity also requires at least read access to the files in your application directory and subdirectories. For more information, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX).

|  |
| --- |
| **NoteNote** |
| Because ASP.NET uses the Windows identity of the ASP.NET process when compiling applications and loading configuration information, you must keep application code and configuration information private between applications on a server that hosts multiple applications. On Windows Server 2003 you can create multiple application pools and specify a unique identity for each application pool. You can then restrict access to application files using access control lists (ACLs) (if file system is formatted using NTFS) and these identities. For example, consider two applications, App1 and App2, where the information in each application must be kept private. You can put App1 in the ApplicationPool1 application pool which has an identity of ID\_ApplicationPool1. You can put App2 in the ApplicationPool2 application pool which has an identity of ID\_ApplicationPool2. The ID\_ApplicationPool1 account is given access to the files in App1, but denied access to the files in App2. ID\_ApplicationPool2 is given access to the files in App2, but denied access to the files in App1. Note that you cannot make this separation on Windows 2000 or Windows XP Professional, because on those operating systems, the process identity for all ASP.NET applications is a single identity. |

You control impersonation using the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) configuration element. As with other configuration directives, this directive applies hierarchically. A minimal configuration file to enable impersonation for an application might look like the following example:

<configuration>

<system.web>

<identity impersonate="true"/>

</system.web>

</configuration>

You can also add support for specific names to run an application as a configurable identity, as shown in the following example:

<identity impersonate="true"

userName="contoso\Jane"

password="\*\*\*\*\*\*\*\*" />

Substitute the correct password for the value listed in the previous example.

|  |
| --- |
| **NoteNote** |
| In the preceding example, the user name and password are stored in clear text in the configuration file. To improve the security of your application, it is recommended that you restrict the access to your Web.config file using an Access Control List (ACL) and that you encrypt the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX)configuration element in your Web.config file using protected configuration. For more information, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.ASPX). |

The configuration illustrated in the example enables the entire application to run using the contoso\Jane identity, regardless of the identity of the request. This type of impersonation can be delegated to another computer. That is, if you specify the user name and password for the impersonated user, you can connect to another computer on the network and request resources, such as files or access to SQL Server, using integrated security. If you enable impersonation and do not specify a domain account as the identity, you will not be able to connect to another computer on the network unless your IIS application is configured to use Basic authentication.

|  |
| --- |
| **NoteNote** |
| On Windows 2000, you cannot impersonate using specific user credentials for the identity of the ASP.NET worker process. But you can enable impersonation without specific user credentials so that your application impersonates the identity determined by IIS. For more information, see article 810204, "PRB: Per Request Impersonation Does Not Work on Windows 2000 with ASP.NET," in the Microsoft Knowledge Base at http://support.microsoft.com. |

[Reading the Impersonated Identity](javascript:void(0))

The following code example shows how to programmatically read the identity of the impersonated user:

C#

[VB](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer).ASPX?cs-save-lang=1&cs-lang=vb#code-snippet-3)

String username =

System.Security.Principal.WindowsIdentity.GetCurrent().Name;

# Overview of Web Application Security Threats

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/f13d73y6(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/f13d73y6(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/f13d73y6(d=printer,v=vs.80).ASPX)

If unknown users can access your Web application, the odds are almost certain that malicious users will try to gain unauthorized access to your application. Servers that are accessible to the public on the Internet are constantly probed for vulnerabilities. Therefore, it is recommended that you take precautions and build security into all of your Web applications.

More detailed information about best practices for writing secure code and securing applications can be found in the book "Writing Secure Code" by Michael Howard and David LeBlanc, or through the guidance provided by [Microsoft Patterns and Practices](http://go.microsoft.com/fwlink/?linkid=37129).

[Security Technology Is Only Part of the Solution](javascript:void(0))

Implementing security is only part of the solution. Another important part is vigilance. Even if your system has many security safeguards, you need to watch it closely in these ways:

* Monitor your system's event logs. Watch for repeated attempts to log into your system or for excessive requests being made against your Web server.
* Continually keep your application server up to date with the latest security updates for Microsoft Windows and Internet Information Services (IIS), as well as any updates for Microsoft SQL Server or other data sources that your application might use.

[Threat Modeling](javascript:void(0))

An important part of developing a more secure application is to understand the threats to it. Microsoft has developed a way to categorize threats: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege (STRIDE). The sections below briefly describe these threats and how they apply to Web applications.

### Spoofing

To spoof is to impersonate a user or process in an unauthorized way. At its simplest, spoofing can mean typing in a different user's credentials. A malicious uses might also change the contents of a cookie to pretend that he or she is a different user or that the cookie comes from a different server.

In general, you can help prevent spoofing by using stringent authentication. Any time someone requests access to non-public information, be sure they are who they say they are. You can also help defend against spoofing by keeping credential information safe. For example, do not keep a password or other sensitive information in a cookie, where a malicious user can easily find or modify it.

### Tampering

Tampering means changing or deleting a resource without authorization. One example is defacing a Web page, where the malicious user gets into your site and changes files. An indirect way to tamper is by using a script exploit. A malicious user manages to get code (script) to execute by masking it as user input from a page or as a link.

A primary defense against tampering is to use Windows security to lock down files, directories, and other Windows resources. The application should also run with minimum privileges. You help guard against script exploits by not trusting any information that comes from a user or even from a database. Whenever you get information from an untrusted source, take steps to be sure it does not contain any executable code.

### Repudiation

A repudiation threat involves carrying out a transaction in such a way that there is no proof after the fact of the principals involved in the transaction. In a Web application, this can mean impersonating an innocent user's credentials. You can help guard against repudiation by using stringent authentication. In addition, use the logging features of Windows to keep an audit trail of any activity on the server.

### Information Disclosure

Information disclosure simply means stealing or revealing information that is supposed to be private. A typical example is stealing passwords, but information disclosure can involve access to any file or resource on the server.

The best defense against information disclosure is to have no information to disclose. For example, if you avoid storing passwords, malicious users cannot steal them. An alternative to storing passwords is to store only a hash of the password. When a user presents credentials, you can hash the user's password and compare only the hashes of the two. If you do store sensitive information, use Windows security to help secure it. As always, you should use authentication to help ensure that only authorized users can access restricted information. If you must expose sensitive information, it is recommended that you encrypt the information when stored and use Secure Sockets Layer (SSL) to encrypt the information when sent to and from the browser.

### Denial of Service

A denial of service attack is to deliberately cause an application to be less available than it should be. A typical example is to overload a Web application so that it cannot serve ordinary users. Alternatively, malicious users might try to simply crash your server.

IIS enables you to throttle applications, which means that it limits the number of requests it will serve. You might be able to deny access to users or IP addresses known to be malicious. Keeping your applications online is a matter of running robust code. You should test your application thoroughly and respond appropriately to error conditions wherever possible.

### Elevation of Privilege

An elevation of privilege attack is to use malicious means to get more permissions than normally assigned. For example, in a successful elevation-of-privilege attack, a malicious user manages to get administrative privileges to your Web server, giving himself or herself access to any data on the server as well as control over server capabilities.

To help protect against elevation of privilege, run the application in a least-privilege context if practical. For example, it is recommended that you do not run ASP.NET applications as the SYSTEM (administrative) user.

**Basic Security Practices for Web Applications**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer,v=vs.80).ASPX)

The topic of creating a secure Web application is extensive. It requires study to understand security vulnerabilities. You also need to familiarize yourself with the security facilities of Windows, the .NET Framework, and ASP.NET. Finally, it is essential to understand how to use these security features to counter threats.

Even if you are not experienced with security, there are basic measures that you should take to protect your Web application. The following list provides minimum-security guidelines that apply to all Web applications and that you should follow:

* [General Web Application Security Recommendations](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor1)
* [Run Applications with Least Privileges](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor2)
* [Know Your Users](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor3)
* [Guard Against Malicious User Input](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor4)
* [Access Databases Securely](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor5)
* [Create Safe Error Messages](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor6)
* [Keep Sensitive Information Safely](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor7)
* [Use Cookies Securely](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor8)
* [Guard Against Denial-of-Service Threats](http://msdn.microsoft.com/en-us/library/zdh19h94(d=printer).ASPX#cpconbestsecuritypracticesforwebapplicationsanchor9)

|  |
| --- |
| **NoteNote** |
| For comprehensive and detailed security guidance which will help you to design, develop, configure, and deploy more secure ASP.NET Web applications, see the security modules provided on the [Microsoft Patterns and Practices](http://go.microsoft.com/fwlink/?LinkId=37129) Web site. |

[General Web Application Security Recommendations](javascript:void(0))

Even the most elaborate application security can fail if a malicious user can use simple ways to get to your computers. Follow these guidelines:

* Back up often and keep your backups physically secure.
* Keep your Web server computer physically secure so that unauthorized users cannot get to it, turn it off, or take it.
* Use the Windows NTFS file system, not FAT32. NTFS offers substantially more security than FAT32. For details, see the Windows documentation.
* Secure the Web server computer and all computers on the same network with strong passwords.
* Secure IIS. For details, see the [Microsoft TechNet Security Center](http://go.microsoft.com/fwlink/?LinkId=50530) Web site.
* Close unused ports and turn off unused services.
* Run a virus checker that monitors inbound and outbound traffic.
* Establish and enforce a policy that forbids users from keeping their passwords written down in an easy-to-find location.
* Use a firewall. For recommendations, see [Microsoft Firewall Guidelines](http://go.microsoft.com/fwlink/?LinkId=50217) on the Microsoft security site.
* Install the latest security patches from Microsoft and other vendors. For example, the see the [Microsoft TechNet Security Center](http://go.microsoft.com/fwlink/?LinkId=50530) Web site has a list of the latest security bulletins for all Microsoft products. Other vendors have similar sites.
* Use Windows event logging and examine the logs frequently for suspicious activity. This includes repeated attempts to log on to your system and an extremely high number of requests against your Web server.

[Run Applications with Least Privileges](javascript:void(0))

When your application runs, it runs within a context that has specific privileges on the local computer and potentially on remote computers. For information on configuring the application identity, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.ASPX). To run with least privileges, follow these guidelines:

* Do not run your application with the identity of a system user (administrator).
* Run the application in the context of a user with the minimum practical privileges.
* Set permissions (Access Control Lists or ACLs) on all the resources required for your application. Use the least permissive setting. For example, if practical in your application, set files to be read-only. For a list of the minimum required ACL permissions required for the identity of your ASP.NET application, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX).
* Keep files for your Web application in a folder below the application root. Do not allow users the option of specifying a path for any file access in your application. This helps prevent users from getting access to the root of your server.

[Know Your Users](javascript:void(0))

In many applications, users access the site anonymously (without having to provide credentials). If so, your application accesses resources by running in the context of a predefined user. By default, this context is the local ASPNET user (on Windows 2000 or Windows XP) or NETWORK SERVICE user (on Windows Server 2003) on the Web server computer. To restrict access to users who are authenticated, follow these guidelines:

* If your application is an intranet application, configure it to use Windows integrated security. That way, the user's logon credentials can be used to access resources. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX).
* If you need to gather credentials from the user, use one of the ASP.NET authentication strategies. For an example, see [Managing Users by Using Membership](http://msdn.microsoft.com/en-us/library/tw292whz.ASPX).

[Guard Against Malicious User Input](javascript:void(0))

As a general rule, never assume that input you get from users is safe. It is easy for malicious users to send potentially dangerous information from the client to your application. To guard against malicious input, follow these guidelines:

* In ASP.NET Web pages, filter user input to check for HTML tags, which might contain script. For details, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.ASPX).
* Never echo (display) unfiltered user input. Before displaying untrusted information, encode HTML to turn potentially harmful script into display strings.
* Never store unfiltered user input in a database.
* If you want to accept some HTML from a user, filter it manually. In your filter, explicitly define what you will accept. Do not create a filter that tries to filter out malicious input; it is very difficult to anticipate all possible malicious input.
* Do not assume that information you get from the HTTP request header (in the [HttpRequest](http://msdn.microsoft.com/en-us/library/system.web.httprequest.ASPX) object) is safe. Use safeguards for query strings, cookies, and so on. Be aware that information the browser reports to the server (user agent information) can be spoofed, in case that is important in your application.
* If possible, do not store sensitive information in a place accessible from the browser, such as hidden fields or cookies. For example, do not store a password in a cookie.

|  |
| --- |
| **NoteNote** |
| View state is stored in a hidden field in an encoded format. By default, it includes a message authentication code (MAC) so that the page can determine whether view state was tampered with. If sensitive information is stored in view state, encrypt by setting the page's [ViewStateEncryptionMode](http://msdn.microsoft.com/en-us/library/system.web.ui.page.viewstateencryptionmode.ASPX) property to true. |

[Access Databases Securely](javascript:void(0))

Databases typically have their own security. An important aspect of a secure Web application is designing a way for the application to access the database securely. Follow these guidelines:

* Use the inherent security of your database to limit who can access database resources. The exact strategy depends on your database and your application:
  + If practical in your application, use integrated security so that only Windows-authenticated users can access the database. Integrated security is more secure than passing explicit credentials to the database.
  + If your application involves anonymous access, create a single user with very limited permissions, and perform queries by connecting as this user.
* Do not create SQL statements by concatenating strings that involve user input. Instead, create a parameterized query and use user input to set parameter values.
* If you must store a user name and password somewhere to use as the database login credentials, store them in the Web.config file and secure the file with protected configuration. For details, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.ASPX).

For more information on accessing data securely, see [Securing Data Access](http://msdn.microsoft.com/en-us/library/ms178375.ASPX) and [Securing ADO.NET Applications](http://msdn.microsoft.com/en-us/library/ecb3hak0.ASPX).

[Create Safe Error Messages](javascript:void(0))

If you are not careful, a malicious user can deduce important information about your application from the error messages it displays. Follow these guidelines:

* Do not write error messages that echo information that might be useful to malicious users, such as a user name.
* Configure the application not to show detailed errors to users. If you want to display detailed error messages for debugging, determine first whether the user is local to the Web server. For details, see [How to: Display Safe Error Messages](http://msdn.microsoft.com/en-us/library/994a1482.ASPX).
* Use the [customErrors](http://msdn.microsoft.com/en-us/library/h0hfz6fc.ASPX)configuration element to control who can view exceptions from the server.
* Create custom error handling for situations that are prone to error, such as database access. For more information, see [Error Handling in ASP.NET Pages and Applications](http://msdn.microsoft.com/en-us/library/w16865z6.ASPX).

[Keep Sensitive Information Safely](javascript:void(0))

Sensitive information is any information that you need to keep private. A typical piece of sensitive information is a password or an encryption key. If a malicious user can get to the sensitive information, then the data protected by the secret is compromised. Follow these guidelines:

* If your application transmits sensitive information between the browser and the server, consider using the Secure Sockets Layer (SSL). For details on how to secure a site with SSL, see article Q307267, "HOW TO: Secure XML Web Services with Secure Socket Layer in Windows 2000" in the Microsoft Knowledge Base at http://support.microsoft.com.
* Use protected configuration to secure sensitive information in configuration files such as the Web.config or Machine.config files. For more information, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.ASPX).
* If you must store sensitive information, do not keep it in a Web page, even in a form that you think people will not be able to see it (such as in server code).
* Use the strong encryption algorithms supplied in the [System.Security.Cryptography](http://msdn.microsoft.com/en-us/library/system.security.cryptography.ASPX) namespace.

[Use Cookies Securely](javascript:void(0))

Cookies are a useful way to keep user-specific information available. However, because cookies are sent to the browser's computer, they are vulnerable to spoofing or other malicious use. Follow these guidelines:

* Do not store any critical information in cookies. For example, do not store a user's password in a cookie, even temporarily. As a rule, do not keep anything in a cookie that, if spoofed, can compromise your application. Instead, keep a reference in the cookie to a location on the server where the information is.
* Set expiration dates on cookies to the shortest practical time you can. Avoid permanent cookies if possible.
* Consider encrypting information in cookies.
* Consider setting the [Secure](http://msdn.microsoft.com/en-us/library/system.web.httpcookie.secure.ASPX) and [HttpOnly](http://msdn.microsoft.com/en-us/library/system.web.httpcookie.httponly.ASPX) properties on the cookie to true.

[Guard Against Denial-of-Service Threats](javascript:void(0))

An indirect way that a malicious user can compromise your application is by making it unavailable. The malicious user can keep the application too busy to service other users, or if can simply cause the application to crash. Follow these guidelines:

* Use error handling (for example, try-catch). Include a finally block in which you release resources in case of failure.
* Configure IIS to use process throttling, which prevents an application from using up a disproportionate amount of CPU time.
* Test size limits of user input before using or storing it.
* Put size safeguards on database queries. For example, before you display query results in an ASP.NET Web page, be sure that there are not an unreasonable number of records.
* Put a size limit on file uploads, if those are part of your application. You can set a limit in the Web.config file using syntax such as the following, where the maxRequestLength value is in kilobytes:

<configuration>

<system.web>

<httpRuntime maxRequestLength="4096" />

</system.web>

</configuration>

You can also use [RequestLengthDiskThreshold](http://msdn.microsoft.com/en-us/library/system.web.configuration.httpruntimesection.requestlengthdiskthreshold.ASPX) property in to reduce the memory overhead of large uploads and form posts.

**Storing Sensitive Information Using ASP.NET**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/kek09k9k(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/kek09k9k(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/kek09k9k(d=printer,v=vs.80).ASPX)

Often in an ASP.NET application you are required to make use of highly sensitive information. For example, you may need to use a user ID and password to connect to a database or you may be storing user IDs and passwords that customers use to access your application. While you can use secure sockets layer (SSL) to encrypt information as it is passed over the network, that information must also be protected when it is stored both on the server and on the client. This topic covers some general guidelines for storing sensitive information.

|  |
| --- |
| **NoteNote** |
| For more information about how to store sensitive information, see [Improving Web Application Security: Threats and Countermeasures](http://go.microsoft.com/fwlink/?LinkId=50218) on the MSDN Web site. |

[Avoid Storing Sensitive Information When Possible](javascript:void(0))

The best way to avoid exposing sensitive information in an application is not to store it. Minimize the places where sensitive information is stored. Avoid storing sensitive information for your application in a cookie or a control that is persisted in the browser, which would expose the sensitive information to clients of your application. Avoid storing sensitive information in your application logic. Instead, retrieve the sensitive information from a secure configuration location or from the client.

[Encrypt Sensitive Information](javascript:void(0))

When you do store sensitive information, avoid storing it in human-readable text or in an easily decoded format, such as Base64 encoding. Instead, encrypt the information so that, if it is exposed to an attacker somehow, the attacker cannot easily determine what the sensitive information contains.

If the sensitive information needs only to be verified and not decrypted to a human-readable format, encrypt the sensitive information using a one-way hash. Then, when comparing the sensitive information received from a source that is being validated, hash the value received and compare the hashes for verification. For example, if you are using ASP.NET Membership and Forms Authentication to provide user authentication for your application, set the password format to [Hashed](http://msdn.microsoft.com/en-us/library/system.web.security.membershippasswordformat.ASPX) so that passwords are encrypted using a one-way hash when they are stored in the data source or compared for validation.

When storing sensitive information such as connection strings, user credentials, or encryption keys in the Web.config file for an application, encrypt the sensitive sections of the Web.config file using a protected configuration provider. For more information about protected configuration, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.ASPX).

For more information about using encryption to protect sensitive information, see [.NET Framework Cryptography Model](http://msdn.microsoft.com/en-us/library/0ss79b2x.ASPX).

[Protect Sensitive Information using Permissions](javascript:void(0))

When storing sensitive information in files, databases, the registry, or other locations, use NTFS Access Control Lists and database permissions to restrict access to the information to only the required sources and only the require access. For more information, see your database documentation or [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX).

# Limiting Access to ASP.NET Web Sites

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/3yfs7yc7(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/3yfs7yc7(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/3yfs7yc7(d=printer,v=vs.80).ASPX)

Limiting access to an application is generally divided into two topics: authentication, which is how an application identifies who you are, and authorization, which is how an application identifies what you are permission to do. This topic provides an overview of authentication and authorization in ASP.NET Web applications. For more detailed information, see [ASP.NET Web Application Security](http://msdn.microsoft.com/en-us/library/330a99hc.ASPX).

[Authenticating Users](javascript:void(0))

ASP.NET applications offer several different options for authenticating users. For read-only applications that can be viewed by anyone, use anonymous authentication. For more restricted access to an application, you need to use some form of authentication to identify users. There are two identities that you should to consider when authenticating users for your ASP.NET application: the application identity that is used to access Windows resources and the ASP.NET user identity that is used to identify a user to ASP.NET.

Your application can run without an ASP.NET user identity, but you will always have a Windows application identity. To help secure your application, you should restrict the Windows identity for the application to the required resources, such as file and database access.

### ASP.NET Application Identity

When an ASP.NET page is executing, the server must have a security context, or identity, for the process that is executing the ASP.NET code. This identity is used when securing resources using Windows Integrated security, such as files protected using the NTFS file system or network resources.

For example, the files that contain the application code stored in the App\_Code subdirectory of an application only need to be read by the ASP.NET application identity. Therefore, the security settings for the files in the App\_Code directory can be restricted so that the ASP.NET application identity has Read access only. Another common use of the Windows identity of the ASP.NET application is as the identity of a connection to a SQL Server using Integrated Security. For more information, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX) and [How to: Access SQL Server Using Windows Integrated Security](http://msdn.microsoft.com/en-us/library/bsz5788z.ASPX).

The identity of an ASP.NET application is determined by several factors. By default, ASP.NET pages run with the Windows identity of the service that processes ASP.NET pages on the Web server. On a computer running Windows Server 2003, that identity is the identity of the application pool that the ASP.NET application is a part of (by default, the NETWORK SERVICE account). On computers running Windows 2000 and Windows XP Professional, the identity is the local ASPNET account is created when the .NET Framework is installed. This identity can be configured to a different identity if desired. For more information, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.ASPX).

You can modify the Windows identity that your ASP.NET page runs as by using the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) element of the system.web configuration section. The identity element can be used to instruct ASP.NET to impersonate a Windows user ID. Impersonating a Windows identity means that the ASP.NET pages for the application will run as that Windows identity. You can specify a user name and password to impersonate. Alternatively, you can enable impersonation and ASP.NET will run in one of two ways: an anonymous identity specified by IIS, or the authenticated browser identity as determined by IIS (for example, Anonymous authentication, Windows Integrated (NTLM) authentication, and so on). For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX).

If you are impersonating a Windows identity, you can execute code that reverts to the original identity of the process instead of the impersonated user ID. For this reason, in environments where you need to keep one application separate from another, you should isolate the applications in separate application pools on computers running Windows Server 2003. Each application pool should be configured with a unique Windows identity.

You can easily determine the Windows identity of the operating system thread that your ASP.NET page is running by using the [Name](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.name.ASPX) property of the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.ASPX) returned by the [GetCurrent](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.getcurrent.ASPX) method as shown in the following code example.

<%=System.Security.Principal.WindowsIdentity.GetCurrent().Name%>

### ASP.NET User

The ASP.NET user identity is used to access ASP.NET-specific resources. For example, you can identify a portion of your application that is only available to certain users, while other portions of your application are available to all users.

The ASP.NET user is determined by the authentication element of the system.web section of the Web.config file for your application. You have several options for authenticating the ASP.NET identity for your application. You can use the Windows user name that is determined by IIS, ASP.NET Forms authentication, Passport authentication, or a custom authentication scheme. The ASP.NET identity can be accessed using the [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user.ASPX) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.ASPX). For details, see [ASP.NET Authentication](http://msdn.microsoft.com/en-us/library/eeyk640h.ASPX).

If you are using ASP.NET Forms authentication or a custom authentication solution to provide the ASP.NET identity, you can use ASP.NET membership to provide a user data store and user management functionality. For more information, see [Managing Users by Using Membership](http://msdn.microsoft.com/en-us/library/tw292whz.ASPX).

[Authorizing Users](javascript:void(0))

Authorization involves restricting user access to only those resources that are required. This includes restricting access to only the required files, databases, and portions of your application. In addition, this includes using Code Access Security to restrict access to code.

You can restrict file access by using NTFS access control lists and the [FileAuthorizationModule](http://msdn.microsoft.com/en-us/library/system.web.security.fileauthorizationmodule.ASPX). For more information, see [ASP.NET Authorization](http://msdn.microsoft.com/en-us/library/wce3kxhd.ASPX) and [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX).

You can restrict access to portions of your application by using the [UrlAuthorizationModule](http://msdn.microsoft.com/en-us/library/system.web.security.urlauthorizationmodule.ASPX) and ASP.NET Role Management. For more information, see [ASP.NET Authorization](http://msdn.microsoft.com/en-us/library/wce3kxhd.ASPX) and [Managing Authorization Using Roles](http://msdn.microsoft.com/en-us/library/9ab2fxh0.ASPX).

**Script Exploits Overview**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/w1sw53ds(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/w1sw53ds(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/w1sw53ds(d=printer,v=vs.80).ASPX)
* [Visual Studio .NET 2003](http://msdn.microsoft.com/en-us/library/w1sw53ds(d=printer,v=vs.71).ASPX)

From the perspective of a browser, a Web page is simply a long string of characters. The browser processes the string sequentially, displaying some characters while interpreting other characters, such as <b> and <script> according to special rules. If a malicious user can insert some of those special characters into a page, the browser will not know that the characters are not supposed to be there, and it will process them as part of the page.

A simplistic script exploit might work as follows. If an application allows users to post comments about the latest movies for other users to read, the exploit steps might be:

1. The application displays a form where users enter comments. The malicious user writes a comment that includes a <script> block in it.
2. The form is posted and the malicious user's comment is stored in a database.
3. Another user visits the site. When the page is constructed, it reads comments out of the database and puts them into the page. The malicious user's <script> block is written into the page as if it were a text comment.
4. When the second user's browser displays the page, it gets to the <script> block and executes it.

There are other ways that malicious users can exploit script. Most script exploits require the application to accept the malicious input and inject it (or echo it) into a page where it will be executed by the browser. The potential damage from such an exploit depends on the script that is executed. It can be trivial, such as an annoying message that pops up in the browser. But it can also do serious damage by stealing cookies, stealing user input (such as a password), and, if Internet security is lax, running native code on the user's computer.

For information about preventing script exploits, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.ASPX).

|  |
| --- |
| **NoteNote** |
| ASP.NET helps protect against script exploits that are disguised as URLs by checking for potentially dangerous strings, such as "<!", "</", and "<?". For more information, see [HtmlEncode](http://msdn.microsoft.com/en-us/library/system.web.httpserverutility.htmlencode.ASPX) and [ValidateRequest](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.validaterequest.ASPX). |

[SQL Statement Exploits](javascript:void(0))

A variation on a script exploit is one that causes malicious SQL statements to be executed. This can occur if an application prompts users for information and then concatenates the user's input into a string representing the SQL statement. For example, an application might prompt for a customer name with the intention of executing a statement, such as the following:

"Select \* From Customers where CustomerName = " & txtCustomerName.Value

But a malicious user who knows something about the database could use the text box to enter an embedded SQL statement with the customer name, resulting in a statement like the following:

Select \* From Customers Where CustomerName = 'a' Delete From

Customers Where CustomerName > ''

When the query is executed, the database is compromised.

[Guarding Against Scripting Exploits](javascript:void(0))

The primary defense against scripting exploits is to never trust information coming from a user. Assume that any data posted to your application from a browser can contain malicious script.

Similarly, any time that you write a string into a page, you should assume that the string could contain malicious script (unless you programmatically created the string yourself). For example, when you read strings out of a database, you should assume that they can contain malicious script. The most security-conscious developers distrust even their own databases, on the theory that a malicious user might have found a way to tamper with the database.

ASP.NET provides you with several ways to help protect against scripting exploits:

* ASP.NET performs request validation against query-string and form variables as well as cookie values. By default, if the current [Request](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.request.ASPX) contains HTML-encoded elements or certain HTML characters (such as &#151; for an em dash), the ASP.NET page framework raises an error.
* If you want to display strings in your application but do not trust them, apply HTML encoding to them when the strings are written back in a response. For example, with encoding, the tag <b> becomes &lt;b&gt;. You might do this if the strings that you are displaying are from a database whose contents you are not sure that you can trust.
* If you want your application to accept some HTML (for example, some formatting instructions from users), you should encode the HTML at the client before it is submitted to the server. For more information, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.ASPX).
* To help protect against SQL statement exploits, never create SQL queries using string concatenation. Instead, use a parameterized query and assign user input to parameter objects.
* Always validate form input against a set of expected values and string formatting/type validation. For example, if a specific form variable is expected to be an integer, use the [TryParse](http://msdn.microsoft.com/en-us/library/system.int32.tryparse.ASPX) method to verify that the value really is an integer and use range checking to help ensure that the value is within an acceptable range.

# How to: Display Safe Error Messages

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/994a1482(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/994a1482(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/994a1482(d=printer,v=vs.80).ASPX)

When your application displays error messages, it should not give away information that a malicious user might find helpful in attacking your system. For example, if your application unsuccessfully tries to log in to a database, it should not display an error message that includes the user name it is using.

There are a number of ways to control error messages, including the following:

* Configure the application not to show verbose error messages to remote users. (Remote users are those who request pages while not working on the Web server computer.) You can optionally redirect errors to an application page.
* Include error handling whenever practical and construct your own error messages. In your error handler, you can test to see whether the user is local and react accordingly.
* Create a global error handler at the page or application level that catches all unhandled exceptions and routes them to a generic error page. That way, even if you did not anticipate a problem, at least users will not see an exception page.

### To configure the application to turn off errors for remote users

* In the Web.config file for your application, make the following changes to the customErrors element:
  + Set the mode attribute to RemoteOnly (case-sensitive). This configures the application to show detailed errors only to local users (that is, to you, the developer).
  + Optionally include a defaultRedirect attribute that points to an application error page.
  + Optionally include <error> elements that redirect specific errors to specific pages. For example, you can redirect standard 404 errors (page not found) to your own application page.

The following code example shows a typical customErrors block in the Web.config file.

<customErrors mode="RemoteOnly" defaultRedirect="AppErrors.aspx">

<error statusCode="404" redirect="NoSuchPage.aspx"/>

<error statusCode="403" redirect="NoAccessAllowed.aspx"/>

</customErrors>

### To include error handling

1. Use a try-catch block around any statements that might generate errors.
2. Optionally, test for a local user with the [IsLocal](http://msdn.microsoft.com/en-us/library/system.web.httprequest.islocal.ASPX) property and modify error handling accordingly. The value 127.0.0.1 is equivalent to localhost and indicates that the browser is on the same computer as the Web server.

The following code example shows an error-handling block. If an error occurs, a session state variable is loaded with details about the message, and the application then displays a page that can read the [Session](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.session.ASPX) variable and display the error. (The error is deliberately written to provide no exploitable details to the user.) If the user is local, different error details are provided. In the finally block, an open resource is released.

VB

Try

SqlConnection1.Open()

SqlDataAdapter1.Fill(Me.DsPubs1)

Catch ex As Exception

If Request.IsLocal Then

Session("CurrentError") = ex.Message

Else

Session("CurrentError") = "Error processing page."

End If

Server.Transfer("ApplicationError.aspx")

Finally

SqlConnection1.Close()

End Try

C#

try

{

sqlConnection1.Open();

sqlDataAdapter1.Fill(dsCustomers1);

}

catch (Exception ex)

{

if(Request.IsLocal)

{ Session["CurrentError"] = ex.Message; }

else

{ Session["CurrentError"] = "Error processing page."; }

Server.Transfer("ApplicationError.aspx");

}

finally

{

this.sqlConnection1.Close();

}

[Creating a Global Error Handler](javascript:void(0))

You can also create an error handler that catches all unhandled exceptions at the page level or for the application as a whole.

### To create a global error handler

* To create a global handler in a page, create a handler for the [TemplateControl.Error](http://msdn.microsoft.com/en-us/library/system.web.ui.templatecontrol.error.ASPX) event. To create an application-wide error handler, in the Global.asax file, add code to the [HttpApplication.Error](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.error.ASPX) event. These methods are called if an unhandled exception occurs anywhere in your page or application, respectively. You can get information about the most recent error from the [GetLastError](http://msdn.microsoft.com/en-us/library/system.web.httpserverutility.getlasterror.ASPX) method.

|  |
| --- |
| Note**Note** |
| If you have a global error handler, it takes precedence over error handling specified in the defaultRedirect attribute of the [customErrors](http://msdn.microsoft.com/en-us/library/y123fsf7.ASPX) configuration element. |

* The following code example shows a handler that gets information about the current error, puts it into a [Session](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.session.ASPX) variable, and then calls a generic error-handling page that can extract and display the error information.

VB

Sub Application\_Error(ByVal sender As Object, ByVal e As EventArgs)

Session("CurrentError") = "Global: " & \_

Server.GetLastError.Message

Server.Transfer("lasterr.aspx")

End Sub

C#

protected void Application\_Error(Object sender, EventArgs e)

{

Session["CurrentError"] = "Global: " +

Server.GetLastError().Message;

Server.Transfer("lasterr.aspx");

}

**Accessing SQL Server from a Web Application**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ht43wsex(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ht43wsex(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ht43wsex(d=printer,v=vs.80).ASPX)

When a Web application involves database access, it must provide credentials to SQL Server (that is, it must log in to SQL Server) just as any other user or process would. In a Web application, this can introduce complications. For example, if the Web application runs anonymously, there might not be credentials to pass to SQL Server.

There are a number of ways to design SQL Server access for your Web application. The strategy you choose depends on how your computers are configured and whether you are on an intranet. The simplest options are:

* Use Windows integrated security. This option passes the user's credentials to SQL Server. Because of delegation issues, this frequently only works by default if SQL Server is on the same computer as IIS.
* Map the identity of your ASP.NET application to a Windows domain user and then log into the database as that user. This works well for anonymous access if SQL Server and the Web server are on separate computers.
* Access the SQL Server as the local identity of your ASP.NET application (for example, the local ASPNET account on a Windows 2000 server or the local NETWORK SERVICE account on a Windows Server 2003). This option works well for anonymous access.
* Pass an explicit user name and password in a connection string. This option can be less secure than other options so you should always use protected configuration to secure the connection strings. You can pass a predetermined user name and password.

[In This Section](javascript:void(0))

|  |  |
| --- | --- |
| **Term** | **Definition** |
| [How to: Access SQL Server Using Windows Integrated Security](http://msdn.microsoft.com/en-us/library/bsz5788z.ASPX) | Provides an example of how to use Windows integrated security for database access. |
| [How to: Access SQL Server Using a Mapped Windows Domain User](http://msdn.microsoft.com/en-us/library/2xzyzb0f.ASPX) | Provides an example of how to use a mapped Windows domain user for database access. |
| [How to: Access SQL Server as a Local User](http://msdn.microsoft.com/en-us/library/e2t54ss5.ASPX) | Provides an example of how to use a local user account for database access. |
| [How to: Access SQL Server Using Predetermined Credentials](http://msdn.microsoft.com/en-us/library/ds20z471.ASPX) | Provides an example of how to use predetermined login information for database access. |

# How to: Access SQL Server Using Windows Integrated Security

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/bsz5788z(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/bsz5788z(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/bsz5788z(d=printer,v=vs.80).ASPX)

If your application runs on a Windows-based intranet, you might be able to use Windows integrated authentication for database access. Integrated security uses the current Windows identity established on the operating system thread to access the SQL Server database. You can then map the Windows identity to a SQL Server database and permissions.

To connect to SQL Server using Windows integrated authentication, you must identify the Windows identity under which your ASP.NET application is running. You must also be sure that the identity has been granted access to the SQL Server database. This topic includes a code example that displays the current Windows identity of the ASP.NET application.

[Connecting to SQL Server](javascript:void(0))

If SQL Server is on a different computer than the Web server, the Windows identity must be able to flow across the network to the remote instance of SQL Server. (Windows networks that have been configured appropriately with Kerberos authentication are able to do this.) However, depending on the settings in the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) configuration element, the Windows identity established on the operating system thread for ASP.NET applications may not be able to flow properly to the remote SQL Server.

You can supply a specific user name and password for the Web site's worker process identity as shown in [How to: Access SQL Server Using a Mapped Windows Domain User](http://msdn.microsoft.com/en-us/library/2xzyzb0f.ASPX), or you can impersonate the authenticated identity supplied by Internet Information Services (IIS). To impersonate the Windows identity supplied by IIS, set the impersonate attribute of the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) configuration element to true as shown in the following example:

<system.web>

<identity impersonate="true" />

</system.web>

In IIS, only Basic Authentication logs users on with a security token that flows across the network to a remote SQL server. By default, other IIS security modes used in conjunction with the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc.ASPX) configuration element settings will not result in a token that can authenticate to a remote SQL Server.

If the Web site is configured to support only anonymous access in IIS, then the security token passed from IIS will be that of the Windows user account for anonymous access as configured in IIS. The anonymous user account can be used to authenticate against a remote SQL Server. However, the default anonymous user account is a local machine account and thus will not exist as an account on the remote SQL Server. You can change the IIS anonymous account to use a domain account, or you can mirror the local machine account on the remote SQL Server by creating a local account on the remote SQL Server with the same user name and password. Additionally the LogonMethod metabase property for IIS6 must be set to an option that allows credentials to flow across the network. For example, the metabase setting MD\_LOGON\_NETWORK\_CLEARTEXT allows logon credentials to flow across the network.

If you are unsure of the Windows identity for your application and whether that identity is logged on with a token that can flow across the network, you can run the following ASP.NET page as part of your application to display the name of the Windows identity and a value indicating whether the identity can flow across the network. Note that the following sample does not take into account whether or not Kerberos delegation has been successfully set up for your domain.

C#

[VB](http://msdn.microsoft.com/en-us/library/bsz5788z(d=printer).ASPX?cs-save-lang=1&cs-lang=vb#code-snippet-2)

<%@ Page Language="C#" %>

<%@ Import namespace="System.Security.Principal" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title>ASP.NET Example</title>

<script runat="server">

public bool WillFlowAcrossNetwork(WindowsIdentity w)

{

foreach (SecurityIdentifier s in w.Groups)

{

if (s.IsWellKnown(WellKnownSidType.InteractiveSid)) { return true; }

if (s.IsWellKnown(WellKnownSidType.BatchSid)) { return true; }

if (s.IsWellKnown(WellKnownSidType.ServiceSid)) { return true; }

}

return false;

}

</script>

</head>

<body>

<%

WindowsIdentity current = WindowsIdentity.GetCurrent();

Response.Write(current.Name + ", " + WillFlowAcrossNetwork(current) + "<br />");

%>

</body>

</html>

The following procedure shows how to access a SQL Server database using Windows integrated authentication in an intranet scenario, where each user has been granted access to the SQL Server individually.

To begin, you need to configure your application in IIS to turn off anonymous access and turn on Windows authentication.

### To configure IIS for Windows integrated authentication

1. In Windows, open the Internet Information Services administration tool.
   * In the Microsoft Windows 2000 Server or Windows Server 2003 operating systems: In the Windows Start menu, point to Programs, then Administrative Tools, and then Internet Services Manager.
   * In the Microsoft Windows XP Professional operating system: open Administrative Tools in the Control Panel.
2. Open the node for your server, and then open nodes until you find the node for your application, which is typically located under Default Web Site.
3. Right-click your application and then click Properties.
4. On the Directory Security tab, click Edit.
5. In the Authentication Methods dialog box, clear the Anonymous Access check box, and then do one of the following:
   * If SQL Server is on the same computer as IIS, select the Integrated Windows authentication check box.
   * If SQL Server is a remote server, select the Basic Authentication check box and clear the Integrated Windows authentication check box.
6. Click all the dialog boxes.

In the application configuration file (Web.config), specify that the application will impersonate the user's credentials supplied by IIS.

### To configure Web.config to impersonate the identity supplied by IIS

* Open the Web.config file for your application and add the following to the system.web element:

<identity impersonate="true"/>

|  |
| --- |
| Note**Note** |
| Elements in Web.config are case sensitive. |

When you create a connection string to access SQL Server, you must include attributes that tell SQL Server that you are using integrated security.

### To configure connection strings for Windows integrated security

* In any connection string for SQL Server, include the attribute Trusted\_Connection=Yes and remove the username and password attributes.

The following shows a typical connection string configured for Windows integrated security:

"workstation id=WebServer1;packet size=4096;

Trusted\_Connection=Yes;data source=SQLServer01";

persist security info=False;initial catalog=northwind"

Set up SQL Server to recognize the users who will be accessing it.

### To configure SQL Server for Windows integrated security

1. From the Windows Start menu, select Microsoft SQL Server, and then select Enterprise Manager.
2. Open the node for the server and expand the node for the database you want to give users permissions for.
3. Right-click the Users node and select New Database User.
4. In the Database User Properties dialog box, enter domain\username in the Login name box, and then click OK. Additionally, configure the SQL Server to allow all domain users to access the database.

# How to: Access SQL Server Using a Mapped Windows Domain User

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/2xzyzb0f(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/2xzyzb0f(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/2xzyzb0f(d=printer,v=vs.80).ASPX)

By default, in Microsoft Windows 2000 and Microsoft Windows XP, ASP.NET applications run in the context of the local user account named ASPNET and in Windows Server 2003 in the context of the local user account named NETWORK SERVICE. These user accounts have limited access rights. However, the ASPNET account is local to the Web server. Because the ASPNET account is local to the Web server, it is not recognized as a user on remote computers. To work around this limitation, you can have your application run in the context of a Windows domain user who is recognized on both the Web server and the computer that is running Microsoft SQL Server.

Mapping your application process to a Windows domain user account requires that you configure the following:

* The Web server.

You must make sure that the Windows domain user account that you specify has sufficient user rights (but no more) to run a Web application.

* Your application.

You must configure the Web.config file for ASP.NET to recognize the domain user account name.

|  |
| --- |
| Note**Note** |
| For information about the Machine.config and Web.config files, see [ASP.NET Configuration Overview](http://msdn.microsoft.com/en-us/library/ms178683.ASPX). |

* A connection string.

When you create connection strings for connection objects in your application, you have to specify that the connection strings will use Windows integrated security.

* SQL Server.

You must add the specified domain user account as a SQL Server login user.

[Configuring a User Account on the Web Server](javascript:void(0))

### To set user rights for the Windows domain user account

1. On the Web server, use Windows administrative tools to make sure that the mapped Windows domain user account has the required user rights.

For detailed information, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX).

1. Run aspnet\_regiis.exe with the -ga switch to grant the common user rights that are required by the identity that you will use for application impersonation.

[Mapping to the Windows User Account and Enabling Impersonation](javascript:void(0))

After establishing the correct user rights for the domain user account, configure the application identity impersonation.

|  |
| --- |
| Security note**Security Note** |
| When you place user credentials in your Web.config file, there are potential security threats. Users with access rights to the directory containing the Web.config file can read the file, and thus see the credentials. For details on how to protect against this threat, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.ASPX). |

### To configure the Web application for impersonation

* Open the Web.config file for your application, and then add the following identity impersonation code:

<identity impersonate="true" userName="domain\username" password="\*\*\*\*\*\*\*\*"/>

Substitute the correct password for the value listed in the previous example.

|  |
| --- |
| Note**Note** |
| Elements in the Web.config file are case sensitive. |

[Using Windows Security in the Connection String](javascript:void(0))

Finally, when you create connection strings for database access, configure the connection strings to use Windows integrated security.

### To use Windows integrated security in a connection string

* When you create a connection string for your application, do not include a user name and password. Instead, for the connection string, set the Integrated Security attribute to SSPI.

The following example shows a connection string that includes the appropriate attributes:

data source=myserver;initial catalog=northwind;Integrated Security=SSPI

### To configure SQL Server for integrated security

1. In Windows, click Start, point to Microsoft SQL Server, and then click Enterprise Manager.
2. Open the node for the server, and then expand the node for the database to which you want to grant user rights.
3. Right-click Users, and then click New Database User.
4. In the Database User Properties dialog box, in the Login name box, enter domain\username, and then click OK.

# How to: Access SQL Server as a Local User

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/e2t54ss5(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/e2t54ss5(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/e2t54ss5(d=printer,v=vs.80).ASPX)

By default, when an ASP.NET application runs, the application runs in the context of a special local user account named ASPNET (in Microsoft Windows 2000 and Microsoft Windows XP) or NETWORK SERVICE (in Microsoft Windows Server 2003). This user account has sufficient user rights on the Web server to allow the application to run.

If Microsoft SQL Server and the Web server are on the same computer, you can define the ASPNET or NETWORK SERVICE user account as a local user account on the computer running SQL Server.

The advantage of this method is that it is secure, because the Web application runs within a limited security context. Using a single user name with SQL Server is also efficient because it enables SQL Server to take advantage of connection pooling, which further enhances the scalability of the application.

### To grant SQL Server user rights to the ASPNET or NETWORK SERVICE user account

1. In Windows, click Start, point to Programs, point to Microsoft SQL Server, and then click Enterprise Manager.
2. Expand the node for the server, and then expand the node for the database for which you want to grant user rights.
3. Right-click Users, and then click New Database User.
4. In the Database User Properties dialog box, in the Login name box, enter computername\ASPNET or NT AUTHORITY\NETWORK SERVICE, and then click OK.

# How to: Access SQL Server Using Predetermined Credentials

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ds20z471(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ds20z471(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ds20z471(d=printer,v=vs.80).ASPX)

A reliable way to connect to SQL Server is to pass a user name and password in the connection string. You can use a predetermined user name and password. The recommended method is to store the predetermined user name and password on the server as part of the [connectionStrings](http://msdn.microsoft.com/en-us/library/bf7sd233.ASPX) configuration section and then use protected configuration to encrypt the connection string contents. For details, see to [Overview of Protected Configuration](http://msdn.microsoft.com/en-us/library/hh8x3tas.ASPX). It is also recommended that you restrict access to the Web.config file using NTFS file system permissions.

|  |
| --- |
| Security note**Security Note** |
| Never hard-code credentials as strings into programs in your application. Anyone who can get access to the code file, even the compiled code, will be able to get at the credentials. |

|  |
| --- |
| Security note**Security Note** |
| Always give a predetermined user name the minimal access privileges to a resource. Never use "sa" or any other administrative-level user name. Always use strong passwords. |

### To store credentials in the Web.config file

1. In the Web.config file, create a new add key in the [connectionStrings](http://msdn.microsoft.com/en-us/library/bf7sd233.ASPX) element. The connectionStrings element must appear as a child of the configuration element. For details, see [Configuration Sections Schema](http://msdn.microsoft.com/en-us/library/0hyxd0xc.ASPX).

The following example illustrates an add key that contains a user name and password:

<configuration>

<connectionStrings>

<add name="NorthwindConnection"

connectionString="Data Source=localhost;

Initial Catalog=Northwind;

User Id=ApplicationUserID;

Password=#P%19!ef2" />

</connectionStrings>

</configuration>

1. Encrypt the connection string value using protected configuration as shown in [Walkthrough: Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/dtkwfdky.ASPX).
2. In your application, read the credentials from the [ConnectionStrings](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.connectionstrings.ASPX) property of the [ConfigurationManager](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.ASPX) class.

The following example shows how you can read credentials at run time and concatenate them into a connection string:

VB

Dim settings As ConnectionStringSettings

settings = System.Configuration.ConfigurationManager.ConnectionStrings("NorthwindConnection")

Dim connectionString As String = settings.ConnectionString

C#

ConnectionStringSettings settings;

settings = System.Configuration.ConfigurationManager.ConnectionStrings["NorthwindConnection"];

string connectionString = settings.ConnectionString;

# Web Application Security at Run Time

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/yfe5dwc2(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/yfe5dwc2(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/yfe5dwc2(d=printer,v=vs.80).ASPX)
* [Visual Studio .NET 2003](http://msdn.microsoft.com/en-us/library/yfe5dwc2(d=printer,v=vs.71).ASPX)

Developing your application requires that you work with one set of security issues. The other set of issues — and the ones that are generally more prominent in any discussion of Web security — address the security of your application when it is deployed and running.

Web applications by definition allow users access to a central resource — the Web server — and through it, to others such as database servers. By understanding and implementing proper security measures, you:

* Guard your own resources against unauthorized access.
* Restrict access levels by user or role.
* Establish data integrity and confidentiality, providing a relatively secure environment in which your users are comfortable working with your application.
* Establish control over how your application can get access to restricted resources.
* Ensure that your application's code runs as intended.

This topic provides a general discussion of how you can accomplish these goals, and it includes links to additional topics where you can get more details on the technologies involved.

You can help protect your application from unauthorized access by taking advantage of these types of security features:

* Security features offered by Internet Information Services (IIS) as part of its general Web server function. These include Windows file, computer, and user-level security.
* Security you can build into your ASP.NET application to provide application-specific access.

[Security Process in ASP.NET](javascript:void(0))

IIS provides many security options for Web sites. However, IIS security mechanisms are very generic, in that the same mechanisms are used for all applications. Moreover, IIS security options — for example, using Windows Integrated security — might not always be convenient for your application. (For intranet applications, on the other hand, you might want to use Windows Integrated security for simplicity.)

Therefore, in order to provide access to specific portions of your application, you can use ASP.NET security. ASP.NET security works in conjunction with IIS security but extends it so that you can customize features, such as how to get user credentials.

IIS receives requests first from clients and performs any security checks that have been established for your application using IIS management tools. For example, if the application has been configured in IIS to allow anonymous access, IIS performs no credentials check. After performing this initial check of authentication, IIS sends the request to ASP.NET, which can perform a second level of checking. ASP.NET allows you to specify access restrictions in your application using a variety of criteria: you can restrict access to specific pages, or to specific users, and so on.

### Authentication

The following table describes the authentication methods that are supported by ASP.NET. Several of these overlap with IIS authentication. For details, see [ASP.NET Authentication](http://msdn.microsoft.com/en-us/library/eeyk640h.ASPX).

|  |  |
| --- | --- |
| **Authentication Type** | **Description** |
| Anonymous access | For applications where unknown users will be making requests (typically, public Web applications). Overlaps with IIS. |
| Basic and Digest authentication | (IIS Security Option) In this scenario, users without credentials are prompted to supply a user name and password. |
| Windows Integrated Security (also known as NTLM security) | (IIS Security Option) If the user making the request has already been authenticated in a Windows-based network, IIS can pass the user's credentials through when requesting access to a resource. |
| Certificate authentication | (IIS Security Option) In this scenario, the client has a certificate — a digital identification — that has been obtained from a third-party source. The identity mapped to the client certificate is passed to ASP.NET. |
| Kerberos | (IIS Security Option) The Kerberos authentication protocol defines the interactions between a client and a network Authentication Service known as a Key Distribution Center (KDC). Windows 2000 and 2003 implement a KDC as the authentication service on each domain controller. |
| Windows authentication | (ASP.NET Security Option) Integrates with the previously listed IIS security options. ASP.NET takes the security token created by IIS and makes it available as a [WindowsPrincipal](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal.ASPX) object set as the value of the [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user.ASPX) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.ASPX). |
| Forms authentication | (ASP.NET Security Option) If a user needs to be authenticated, ASP.NET redirects the request to a page that you specify. This page usually contains a form in which you get user name information. (For extra security, the form can be exchanged using HTTPS protocol.) When your application gets the form information, it can perform an application-specific check of the user's credentials. An important point is that the process of authentication is under your control (unlike that for IIS), which allows you to specify what the form looks like and how you choose to store user information.  If a user is successfully authenticated, ASP.NET issues an encrypted cookie containing a token identifying the user for subsequent access. |

Forms authentication is the easiest choice for ASP.NET applications on the public internet because it gives you substantial control over how users are authenticated and it allows you to store authentication in a token on the browser.

For details about IIS security, see the access-control topics in the [Windows Server TechCenter for IIS](http://go.microsoft.com/fwlink/?LinkId=37112) on the Microsoft TechNet Web site. For details about ASP.NET authentication, see [ASP.NET Authentication](http://msdn.microsoft.com/en-us/library/eeyk640h.ASPX).

For details about using Forms authentication with protocol transition and constrained delegation in a Windows Server 2003 domain environment, see [Kerberos Protocol Transition and Constrained Delegation](http://go.microsoft.com/fwlink/?LinkId=50220).

### Authorization

When your Web application runs, it requests resources from the Web server and, often, from other processes as well, such as a database. The ASP.NET process runs in a user context that determines how your application will request those resources. The ASP.NET process runs as a special local user named ASPNET (by default) on Windows 2000 and Windows XP Professional Edition, or it runs as the identity of the application pool for the ASP.NET application on Windows 2003 (by default, the local NETWORK SERVICE account). These accounts run with limited permissions. You can specify a different user context for the ASP.NET process, including the local SYSTEM account (which runs your application in administrator context) or a user whose credentials you explicitly provide, though this is not recommended.

In your ASP.NET application, you can specify that different users have authorized access to different resources. If your application is using Windows authentication, you can use Windows permissions to determine authorization to access specific files or folders on the server.

Alternatively, you can use URL-based authorization, in which authorization can be granted or denied according to different criteria:

* Specific users, or identities, which are based on the credentials provided by the user.
* Roles, which are entities defined to allow multiple users to share privileges based on a common role or function.
* Verbs, which are the HTTP processes (such as GET and POST) for accessing portions of your application.

For example, you can specify that all users can get pages (perform the GET verb) from your application, but that only specific users can post pages to it. Similarly, you might specify that all users are allowed to GET pages but specific roles are denied the right to post.

You can grant URL authorization for the application as a whole or on a directory-by-directory basis. A typical use is to allow all users to view pages in a public directory, but to keep restricted pages in a different directory that is authorized only for specific users or roles.

|  |
| --- |
| **NoteNote** |
| By default, static files, such as images and style sheets, are not subject to ASP.NET authorization when they are served through IIS. You can use IIS security features to restrict access to static files if you do not want all users to be able to access the files. If you use the ASP.NET Development Server to test your ASP.NET application, you will see different behavior because static files are subject to ASP.NET authorization and will not be served to an anonymous user when anonymous access to those files is disabled. Alternatively, you can map static file name extensions in IIS to the ASP.NET ISAPI extension, in which case the ASP.NET authorization rules will apply. |

For more information, see [ASP.NET Authorization](http://msdn.microsoft.com/en-us/library/wce3kxhd.ASPX) and [Basic Security Practices for Web Applications](http://msdn.microsoft.com/en-us/library/zdh19h94.ASPX).

[ASP.NET Configuration Files](javascript:void(0))

You establish ASP.NET security choices using settings in a Web.config file. The file allows you to include predefined elements for various security options, including sections for authentication and authorization. The relevant sections of the Web.config file might look like the following.

<configuration>

<system.web>

<authentication mode="Forms">

<forms loginUrl="login.aspx" />

</authentication>

<authorization>

<deny user="?" />

</authorization>

</system.web>

</configuration>

Your application can contain more than one configuration file. By default, there is a configuration file at the application root that specifies security for the application as a whole — that is, security settings are inherited by subfolders. However, you can also create configuration files in individual folders to create security settings for that folder.

For more information, see [ASP.NET Security Architecture](http://msdn.microsoft.com/en-us/library/yedba920.ASPX).

[XML Web Service Security](javascript:void(0))

XML Web services using .asmx files run as Web applications using ASP.NET and therefore participate in the same security model as any ASP.NET application. For example, an XML Web service might be configured to use Basic authentication or Windows integrated security.

At design time, when you attempt to add a reference to an XML Web service — that is, when you request the XML Web service's discovery documents — the XML Web service will perform standard Web application authentication according to how it was configured. For example, if the XML Web service is configured to use Basic authentication, it will expect to get a user name and password from the requesting client. If the XML Web service is using Basic authentication, for example, the Add Web Reference dialog box will prompt you for credentials.

If you are building an application that includes calls to an XML Web service, you need to be sure that you have appropriate credentials before making the call, or the call will fail. At run time, you can pass credentials to the XML Web service by setting the [Credentials](http://msdn.microsoft.com/en-us/library/system.web.services.protocols.webclientprotocol.credentials.ASPX) property of the client-side object representing the XML Web service before you call its methods.

Because other ASP.NET security options might not be sufficiently flexible, XML Web services can implement a custom authentication solution in which credential information is passed in SOAP headers. In this solution, credentials are passed in an optional part of the message exchanged between client and server. You can then write a custom HTTP module (implementing the [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.ASPX) interface) that can listen for the information in the header and call your authentication code.

As with other ASP.NET applications, the XML Web service might implement specific role-based authorizations to limit access to specific parts of the application.

For details, see [XML Web Services Infrastructure](http://msdn.microsoft.com/en-us/library/sd5s0c6d.ASPX) and [Securing XML Web Services Created Using ASP.NET](http://msdn.microsoft.com/en-us/library/w67h0dw7.ASPX).

[Establishing Data Integrity and Confidentiality](javascript:void(0))

Authentication and authorization establish who your users are and what resources they can access. These security features are designed primarily to help protect your Web application from unauthorized use.

However, there is a separate aspect to security as well, which is to help protect your users' information and provide them with the confidence that they can exchange sensitive information with you. For example, if your application asks users for credit card or other account numbers, personal information, or any data that users might not want others to know, you must provide a way for them to submit this information to you securely.

You can use Secure Sockets Layer (SSL) in IIS to exchange encrypted information using the HTTPS protocol. SSL provides encryption in both directions: your information is transmitted to the user using encryption, and information the user posts to your application is likewise encrypted.

### Establishing SSL and Encryption

To use SSL and encryption, you must obtain a server certificate for your company or identity. The certificate is a digital signature that identifies your site in a way that cannot be impersonated. For Internet (public) applications, you obtain a server certificate from a recognized third-party certification authority. For private (intranet) applications, you can issue a server certificate yourself. You might do this to help secure an internal application, such as a personnel site.

Your server certificate also enables to setup SSL-encrypted connections with your browser users. SSL uses an encryption method called public key encryption. In this form of encryption, there are two keys: a public key used to encrypt data and a private key that you keep secret and use to decrypt information encrypted with the public key. The server certificate you obtain includes a public key. When users want to use SSL, your application sends the certificate and the public key to the browser. The browser and server then use the public key to establish a way to encrypt their information exchange.

|  |
| --- |
| Note**Note** |
| Using SSL requires that the browser support an encryption key that is at least 40 bits long. This level is available in most browsers. However this key length is not considered secure. You can optionally configure your application in IIS to only allow SSL connections with a 128-bit key. |

Once you have obtained a server certificate, you can instruct users to use SSL by having them use https:// as the prefix for getting and posting Web pages. Your Web site can also be configured in IIS to only accept HTTPS connections. IIS and the browser will automatically use an encrypted channel to exchange information.

For details about how to use SSL, see article Q307267, "How to: Secure XML Web Services with Secure Sockets Layer in Windows 2000" in the [Microsoft Knowledge Base](http://go.microsoft.com/fwlink/?LinkId=37115). For information about encryption, see [Cryptographic Services](http://msdn.microsoft.com/en-us/library/92f9ye3s.ASPX). For information about certificates and about configuring SSL, see the [Windows Server TechCenter for IIS](http://go.microsoft.com/fwlink/?LinkId=37112) on the Microsoft TechNet Web site.

[Using .NET Code Security](javascript:void(0))

As a final aspect of security, you should take measures to help ensure that the code in your application is protected from misuse, whether by being inadvertently run in an improper context or by being used in a malicious way. Because ASP.NET is part of the .NET Framework, you can also take advantage of code access security to establish permissions for what code is allowed to do. For information, see [Code Access Security](http://msdn.microsoft.com/en-us/library/c5tk9z76.ASPX).

**ASP.NET Application Security in Hosted Environments**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ssd9kbbc(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ssd9kbbc(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ssd9kbbc(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/ssd9kbbc(d=printer,v=vs.71).ASPX)

The topics in this section provide information about how to configure ASP.NET and the ASP.NET environment to improve the security of your ASP.NET application Web server.

[In This Section](javascript:void(0))

[Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.ASPX)

[ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX)

[ASP.NET Code Access Security](http://msdn.microsoft.com/en-us/library/87x8e4d1.ASPX)

[ASP.NET Trust Levels and Policy Files](http://msdn.microsoft.com/en-us/library/wyts434y.ASPX)

**Configuring ASP.NET Process Identity**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/dwc1xthy(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/dwc1xthy(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/dwc1xthy(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/dwc1xthy(d=printer,v=vs.71).ASPX)

ASP.NET pages are executed within a process, or Windows program. All Windows programs run with a specific security identity. By default, the ASP.NET process runs under a predefined Windows identity. Alternatively, by configuring your application to use impersonation, you can set ASP.NET to run under a different identity or under the Windows identity of the user making the request.

To help improve the security of your ASP.NET applications, you should make sure that the ASP.NET process runs with an identity that has only the minimal permissions required to run your applications. This reduces the vulnerability of any resources that are exposed by ASP.NET, should a security breach occur.

On a Web server running Microsoft Windows Server 2003 and Internet Information Services (IIS) 6.0, the ASP.NET process runs in the application pool for the Web application. The application pool defines the identity that ASP.NET runs under (by default, the NETWORK SERVICE account). On earlier versions of IIS, (in Microsoft Windows 2000 and Windows XP Professional operating systems), ASP.NET runs in the ASP.NET worker process (Aspnet\_wp.exe). The identity that ASP.NET runs under is defined by the identity of the Aspnet\_wp.exe process (by default, the ASPNET account).

To specify the identity for your ASP.NET application on a server running Windows Server 2003, use IIS Manager to configure the identity of the application pool for your ASP.NET application. The application-pool identity must also be added to the IIS\_WPG group on the server computer.

To run the ASP.NET worker process with its own account under Windows 2000 or Windows XP Professional, you can make settings in the Web server computer's Machine.config file. The file is located in the following folder:

<drive>:\Windows\Microsoft.NET\Framework\v2.0.50727\Config\machine.config

In the Machine.config file, apply the following attributes to the [<processModel>](http://msdn.microsoft.com/en-us/library/7w2sway1.ASPX) configuration element:

* userName   The name of the Windows account under which the process will run.
* password   The clear-text password for the account. There are security risks associated with storing clear-text passwords in a configuration file. If you store credentials in the configuration file, you should encrypt the contents of the [<processModel>](http://msdn.microsoft.com/en-us/library/7w2sway1.ASPX) configuration element using the ASP.NET Set Registry console application (Aspnet\_setreg.exe). For information about how to use the ASP.NET Set Registry console application, see article 329290, "How to use the ASP.NET utility to encrypt credentials and session state connection strings," in the Microsoft Knowledge Base at the [Microsoft support Web site](http://support.microsoft.com/). For applications that are published on the Internet, use an alternative means of running the application under a specific identity, such as impersonating the anonymous identity supplied by IIS. For more information, see [Using IIS Authentication with ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/134ec8tc.ASPX).

The following example shows how to set these attributes, in the [<processModel>](http://msdn.microsoft.com/en-us/library/7w2sway1.ASPX) element of a configuration file, to run the worker process under a local user account.

<system.web>

<processModel enable="true"

userName="LOCALMACHINE\IUSR\_ProcessUser"

password="%Hco94\*#QW12"/>

</system.web>

In addition to setting the userName attribute to the name of an existing Windows identity, you can set it to the predefined names System or Machine. The System account runs the worker process under the same identity as IIS itself (typically SYSTEM).

|  |
| --- |
| **Security noteSecurity Note** |
| It is not recommended that you run your applications under the System account, because the account has elevated privileges and can therefore represent a security risk if the ASP.NET process is compromised. |

The Machine account runs the worker process with a special account named ASPNET that has limited permissions. With either identity, the process does not have to supply credentials to the operating system.

|  |
| --- |
| **NoteNote** |
| In order for your ASP.NET applications to function properly, you must ensure that the process identity has access to the Access Control Lists (ACLs) that are listed in [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e.ASPX). Additionally, on servers running IIS 5.0, the process identity must be granted read access to the IIS metabase, which can be accomplished using the Aspnet\_regiis.exe tool and specifying the -ga option (for example, aspnet\_regiis -ga "UserDomain\ApplicationUser"). |

**ASP.NET Required Access Control Lists (ACLs)**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/kwzs111e(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/kwzs111e(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/kwzs111e(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/kwzs111e(d=printer,v=vs.71).ASPX)

The following table shows which type of file and folder permissions the identity of an ASP.NET Web application must have in order to function properly. Some permissions are required only by the account that the ASP.NET process is running as, while others are required by any impersonated account also. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.ASPX).

|  |  |  |  |
| --- | --- | --- | --- |
| **Location** | **Access type** | **Account** | **Comments** |
| %SystemRoot%\Microsoft.NET\Framework\versionNumber\Temporary ASP.NET Files | Read/write | Process or configured impersonation. | This is the location for dynamically compiled files. Beneath this location, application code generation takes place in a discrete directory for each application. You can configure the root location using the tempDir attribute of the [<compilation>](http://msdn.microsoft.com/en-us/library/s10awwz0.ASPX) configuration section. |
| %SystemRoot%\assembly | Read | Process or configured impersonation. | This is the location of the [global assembly cache](http://msdn.microsoft.com/en-us/library/yf1d93sz.ASPX) (GAC). |
| %SystemRoot%\System32 | Read | Process | Contains system DLLs loaded by the .NET Framework. |
| %SystemRoot%\Temp | Read/write/delete | Process | Used for Web services support. |
| User profile directory | Read/write | Process | Used by the GAC cache lock files and the security configuration caching mechanism of the common language runtime. If the user profile directory for the account does not exist, ASP.NET uses the default user profile directory. |
| Web application directory | Read | Process or configured impersonation. | This is the location for application files. |
| Web application directory\App\_Data | Read/write | Process or configured impersonation. | This is the default location for data files in an ASP.NET Web application. If your application uses the App\_Data subdirectory, the ASP.NET process must be able to write to the directory and for some databases, to be able to create temporary files in the subdirectory. |
| %SystemRoot%\Microsoft.NET\Framework\version and subdirectories | Read | Process or configured impersonation | ASP.NET must be able to access the system assemblies referenced in the Machine.config file in the CONFIG subdirectory under %SystemRoot%\Microsoft.NET\Framework\version\. |

**ASP.NET Code Access Security**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/87x8e4d1(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/87x8e4d1(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/87x8e4d1(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/87x8e4d1(d=printer,v=vs.71).ASPX)

One of the benefits of using ASP.NET to host multiple Web sites is support in the common language runtime (CLR) for code access security (CAS) to help protect Web applications. Code is assigned to a security zone classification based on evidence about the code's origin, such as a strong name for an assembly or the code's URL of origin. For information about Code Access Security, see [Code Access Security Basics](http://msdn.microsoft.com/en-us/library/33tceax8.ASPX).

You can configure code access security for an individual assembly by making it a strong-named assembly and by adding security policy for that assembly. However, many ASP.NET assemblies are dynamically generated during page compilation and therefore are not strongly named, so you must configure security policy for those assemblies indirectly. Additionally, because ASP.NET supports no-compile applications, assembly-based evidence is not supported. Because ASP.NET applications include the concept of directory structures, it is much easier to configure code access security based on categories of ASP.NET applications as opposed to manually configuring the .NET Framework to work separately with each ASP.NET application on a computer.

For each application, ASP.NET lets you assign a configurable trust level that corresponds to a predefined set of permissions. By default, applications are assigned a trust level according to the evidence they present.

[Homogenous Application Domains](javascript:void(0))

By default, ASP.NET partial-trust application domains are homogeneous. Homogeneous application domains are partial-trust application domains that define a shared permission set for running code. This results in a constrained set of possible permissions for code that is running in a partial-trust application domain. It also means that the application-domain trust boundary is itself associated with a partial-trust grant set. In a homogeneous application domain that is hosted in ASP.NET, code that can be loaded is associated with one of two permission sets. Code either runs with full trust (code ffrom the GAC always runs in full trust), or code runs with the partial-trust permission set that is defined by the current trust-level setting.

After you install ASP.NET the computer contains two sets of ASP.NET partial-trust policy files. The ASP.NET 4 CAS model uses one set. The other set is used when you configure your application to use the CAS model that is provided in versions earlier than ASP.NET 4. For more information, see [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).

[The trust Configuration Element](javascript:void(0))

The [trust](http://msdn.microsoft.com/en-us/library/tkscy493.ASPX) configuration element specifies a trust level. The setting can apply to the computer (machine) level. In that case, every ASP.NET application runs at that trust level. Or the setting can apply to any application root directory in the hierarchy. In that case, the trust level applies to the specific ASP.NET application.

If you want to set policy for a specific application, you can do so by editing the application Web.config file as shown in the following example:

<trust level="High" />

However, if you want to set policy at a higher level (for example, the root of a Web site that has child applications) you can disallow child applications from changing the trust level setting by using a location element as shown in the following example:

<location path="ContosoSite" allowOverride="false">

<trust level="High" />

</location>

It is recommended that you set the level attribute of the trust configuration element to High for sites that are trusted. For sites that are not trusted, such as a Web server that hosts sites that run code from an external customer, we recommend that you set the level attribute of the trust configuration element to Medium.

The trust element provides a [LegacyCasModel](http://msdn.microsoft.com/en-us/library/system.web.configuration.trustsection.legacycasmodel.ASPX) attribute, which is set to false by default. Setting this attribute to true configures an ASP.NET application to use most (although not all) the ASP.NET CAS behavior from versions of earlier than ASP.NET 4.

The following table lists the default supported attributes for the [trust](http://msdn.microsoft.com/en-us/library/tkscy493.ASPX) configuration element.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Supported values** |
| level | Specifies the security zone under which the application will run. | Full, High, Medium, Low, and Minimal.The default is Full. |
| TrustSection.LegacyCasModel | Gets or sets a value that indicates whether legacy code access security is enabled. | true or false. The default is false. |
| originUrl | Specifies a URL or a URL pattern that is allowed connect access. Types in the [System.Net](http://msdn.microsoft.com/en-us/library/system.net.ASPX) namespace use this attribute as part of the security checks that are performed when types in the namespace make HTTP connections. If present, this attribute can be used to check permissions for some objects, such as a [WebRequest](http://msdn.microsoft.com/en-us/library/system.net.webrequest.ASPX) instance, that allow connectivity to various network locations. For example, you could configure this attribute with the host name of servers in a Web farm so that ASP.NET pages could call Web services deployed in the same Web farm as the Web application.  Note**Note**  By default, the ASP.NET 4 CAS configuration enables you to explicitly grant unrestricted [WebPermission](http://msdn.microsoft.com/en-us/library/system.net.webpermission.ASPX) permissions. This eliminates the need to use the [OriginUrl](http://msdn.microsoft.com/en-us/library/system.web.configuration.trustsection.originurl.ASPX) attribute. | Well-formed HTTP URLs, or the regular expression-based syntax that is supported by WebPermissionAttribute. |

The following table lists permission types supported by the CLR and the default policy for each permission under different trust levels.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Permission** | **Full** | **High** | **Medium** | **Low** | **Minimal** |
| [AspNetHostingPermission](http://msdn.microsoft.com/en-us/library/system.web.aspnethostingpermission.ASPX) | Full | High | Medium | Low | Minimal |
| [ConfigurationPermission](http://msdn.microsoft.com/en-us/library/system.configuration.configurationpermission.ASPX) | Unrestricted | Unrestricted | No permission | No permission | No permission |
| [DnsPermission](http://msdn.microsoft.com/en-us/library/system.net.dnspermission.ASPX) | Unrestricted | Unrestricted | Unrestricted | No permission | No permission |
| [EnvironmentPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.environmentpermission.ASPX) | Unrestricted | Unrestricted | [Read](http://msdn.microsoft.com/en-us/library/system.security.permissions.environmentpermissionaccess.ASPX): TEMP, TMP, OS, USERNAME, COMPUTERNAME | No permission | No permission |
| [FileIOPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission.ASPX) | Unrestricted | Unrestricted | [Read](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.ASPX), [Write](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.ASPX), [Append](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.ASPX), [PathDiscovery](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.pathdiscovery.ASPX):Application Directory | [Read](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.ASPX), [PathDiscovery](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermissionaccess.pathdiscovery.ASPX):Application Directory | No permission |
| [IsolatedStorageFilePermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragefilepermission.ASPX) | Unrestricted | Unrestricted | [AssemblyIsolationByUser](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragecontainment.ASPX), Unrestricted [UserQuota](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermissionattribute.userquota.ASPX) | 1 MB [UserQuota](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermissionattribute.userquota.ASPX) (can be changed for individual sites), [AssemblyIsolationByUser](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragecontainment.ASPX) | No permission |
| [PrintingPermission](http://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermission.ASPX) | Unrestricted | [DefaultPrinting](http://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermissionlevel.ASPX) | [DefaultPrinting](http://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermissionlevel.ASPX) | No permission | No permission |
| [ReflectionPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermission.ASPX) | Unrestricted | [ReflectionEmit](http://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermissionflag.ASPX)  and [RestrictedMemberAccess](http://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermissionflag.ASPX). | [RestrictedMemberAccess](http://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermissionflag.ASPX) | No permission | No permission |
| [RegistryPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.registrypermission.ASPX) | Unrestricted | Unrestricted | No permission | No permission | No permission |
| [SecurityPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission.ASPX) | Unrestricted | [Execution](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [ControlPrincipal](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [ControlThread](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [RemotingConfiguration](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX) | [Execution](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [ControlPrincipal](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [ControlThread](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX), [RemotingConfiguration](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX) | [Execution](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX) | [Execution](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.ASPX) |
| [SmtpPermission](http://msdn.microsoft.com/en-us/library/system.net.mail.smtppermission.ASPX) | Unrestricted | [Connect](http://msdn.microsoft.com/en-us/library/system.net.mail.smtpaccess.ASPX) | [Connect](http://msdn.microsoft.com/en-us/library/system.net.mail.smtpaccess.ASPX) | No permission | No permission |
| [SocketPermission](http://msdn.microsoft.com/en-us/library/system.net.socketpermission.ASPX) | Unrestricted | Unrestricted | No permission | No permission | No permission |
| [WebPermission](http://msdn.microsoft.com/en-us/library/system.net.webpermission.ASPX) | Unrestricted | Unrestricted | Unrestricted | No permission | No permission |
| [SqlClientPermission](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlclientpermission.ASPX) | Unrestricted | Unrestricted | Unrestricted | No permission | No permission |
| Event Log | Unrestricted | No permission | No permission | No permission | No permission |
| Message Queue | Unrestricted | No permission | No permission | No permission | No permission |
| Service Controller | Unrestricted | No permission | No permission | No permission | No permission |
| Performance Counters | Unrestricted | No permission | No permission | No permission | No permission |
| Directory Service | Unrestricted | No permission | No permission | No permission | No permission |

When a permission level is available but is not explicitly mentioned in security policy, applications that are running with Full permissions can always use it. Applications that are running with lower trust levels will not be able to use resources unless you grant them explicit permissions by changing the security policy.

As the table shows, application with High permission sets have read/write permission for files in their application directories, and Low trust applications have read-only permission for files in their application directories. Because the [FileIOPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission.ASPX) type relies on a physical path (for example, C:\SampleAppPath), ASP.NET uses a tokenized statement in the policy files that is replaced at run time with relevant path information for the application.

|  |
| --- |
| **NoteNote** |
| By default, ASP.NET grants unrestricted [WebPermission](http://msdn.microsoft.com/en-us/library/system.net.webpermission.ASPX) for ASP.NET Medium trust application. |

# ASP.NET Trust Levels and Policy Files

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/wyts434y(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/wyts434y(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/wyts434y(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/wyts434y(d=printer,v=vs.71).ASPX)

Trust levels for ASP.NET applications are defined using policy files. Trust levels are associated with policy files by using the [securityPolicy](http://msdn.microsoft.com/en-us/library/zhs35b56.ASPX) configuration element, which is valid at the computer (machine) level, application level, and in an application Web.config file. You can add or remove custom trust levels by adding entries to the configuration section that specify the trust level name and the policy file to use. The default trust files are installed in the Config directory under the folder that contains the Aspnet\_isapi.dll file. This same location is used for the Machine.config configuration file and for run-time security policy files.

[ASP.NET Full Trust Level](javascript:void(0))

The trust levels are Full, High, Medium, Low, and Minimal. The Full is equivalent to having full trust in the local computer. The ASP.NET host does not apply any additional policy to applications that are running at the full-trust level.

The following example shows the [securityPolicy](http://msdn.microsoft.com/en-us/library/zhs35b56.ASPX) section of a configuration file that maps the full trust level to a policy file.

<system.web>

<securityPolicy>

<trustLevel name="Full" policyFile="internal"/>

</securityPolicy>

</system.web>

[ASP.NET Partial-Trust Levels](javascript:void(0))

By default, in ASP.NET partial-trust application domains, security policy does not intersect with machine-level, user-level, or enterprise-level CAS policies. ASP.NET partial-trust application domains are homogeneous. Homogeneous application domains assign a single common permission set for running user code and simplify CAS decisions. For more information, see [Homogeneous Application Domains](http://msdn.microsoft.com/en-us/library/dd984947.ASPX#homogeneousapplicationdomains) in [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).

In a homogeneous application domain hosted in ASP.NET, code that can be loaded is associated with one of the following permission sets:

* Code runs with full trust (code from the GAC always runs in full trust).
* Code runs with the partial-trust permission set that is defined by the current trustLevel setting.

|  |
| --- |
| Note**Note** |
| This behavior is different from partial-trust applications in versions of ASP.NET earlier than ASP.NET 4. |

By default, the trust level for application domains for ASP.NET is full trust. The partial-trust behavior in ASP.NET takes effect when the trustLevel element's name attribute is set to a value other than Full.

The policy files that define the partial-trust permission sets for High, Medium, Low, and Minimal are all located in the CONFIG subdirectory of the .NET Framework installation directory. The following example shows the [securityPolicy](http://msdn.microsoft.com/en-us/library/zhs35b56.ASPX) section of a configuration file that maps trust levels to different policy files.

<system.web>

<securityPolicy>

<trustLevel name="High" policyFile="web\_hightrust.config"/>

<trustLevel name="Medium" policyFile="web\_mediumtrust.config"/>

<trustLevel name="Low" policyFile="web\_lowtrust.config"/>

<trustLevel name="Minimal" policyFile="web\_minimaltrust.config"/>

</securityPolicy>

</system.web>

The policy files are named by using the following pattern:

web\_[trustlevelname]trust.config

For example, the partial-trust permission set for Medium trust is in the file that is named web\_mediumtrust.config.

|  |
| --- |
| Note**Note** |
| The fact that the application-domain boundary is now itself partially trusted is the most common CAS change that typically requires you to modify full-trust code in order to make it work in ASP.NET 4. |

If you do not want applications to be able to specify their own trust level, you can include a [location](http://msdn.microsoft.com/en-us/library/b6x6shw7.ASPX) element and set the allowOverride attribute to false. You might do this on a server that hosts multiple applications and that must limit the trust level of the hosted applications.

|  |
| --- |
| Important note**Important** |
| In the default root Web.config file for the server, all the partial-trust related configuration sections are inside a location element whose allowOverride attribute is set to true. To lock down the computer, you change the value of the allowOverride attribute to false. |

[Modifying Trust-Level Files](javascript:void(0))

Although ASP.NET homogeneous application domains constrain code to either full trust or to the named ASP.NET partial-trust permission set, you can influence how a permission set is associated with an assembly. You can customize how a permission set is associated with running code. For information about how to customize a permission set, see [Homogeneous Application Domains](http://msdn.microsoft.com/en-us/library/dd984947.ASPX#homogeneousapplicationdomains) in [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).

### Granting ASP.NET Full Trust to Assemblies

The ASP.NET 4 fullTrustAssemblies section enables you to explicitly establish a list of assembly identities that will always be granted full trust. The securityPolicy configuration section in the Web.config file contains a child fullTrustAssemblies configuration section. The [FullTrustAssembliesSection](http://msdn.microsoft.com/en-us/library/system.web.configuration.fulltrustassembliessection.ASPX) section is a standard collection that supports add, remove, and clear operations, where you can specify one or more assembly identities that will be granted full trust at run time. The following example shows to configure ASP.NET full trust assembly in the fullTrustAssemblies configuration section.

<system.web>

<securityPolicy>

<fullTrustAssemblies>

<add assemblyName="MyCustomAssembly"

version="1.0.0.0"

publicKey="publicSigningKey"/>

</fullTrustAssemblies>

</securityPolicy>

</system.web>

Each entry in the fullTrustAssemblies element identifies an assembly by its assembly name and assembly version, and by a 320-character string that is the hexadecimal character representation of the public half of the signing key. Notice that the assembly location is not specified in the definition. It is up to the individual hosting environment (such as ASP.NET 4) to find and load assemblies. If an assembly that is loaded matches the information that is contained in one of the add elements in fullTrustAssemblies, the assembly is granted full trust.

### Configuring Partial Trust Visible Assemblies

In the .NET Framework 4, the CLR includes a variation of the [AllowPartiallyTrustedCallersAttribute](http://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute.ASPX) (APTCA) attribute that is referred to as conditional APTCA (C-APTCA). Conditional APTCA enables assemblies that are marked with the APTCA attribute to retain APTCA characteristics in only certain hosted environments. For more information about C-APTCA, see "Conditional APTCA" in [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).

Host environments can supply to the CLR a list of conditional-APTCA assemblies whose APTCA characteristics should be honored. ASP.NET supplies a hard-coded list of all the ASP.NET assemblies to the CLR. If ASP.NET did not do this, Web applications would fail immediately when the first line of ASP.NET internal code tried to run in a partial-trust application domain. Since it is impossible for ASP.NET to know in advance all possible conditional-APTCA assemblies, ASP.NET includes a configuration section where conditional-APTCA assemblies can be added.

The securityPolicy configuration section in the Web.config file has a child configuration section named partialTrustVisibleAssemblies. This is a standard collection that supports add, remove, and clear operations, and where you can specify one or more assembly identities that should be treated as APTCA (if they are also marked for conditional APTCA).

The following example shows to configure ASP.NET a partial-trust assembly in the partialTrustVisibleAssemblies configuration section.

<system.web>

<securityPolicy>

<partialTrustVisibleAssemblies>

<add assemblyName="MyCustomAssembly"

publicKey="publicSigningKey"

/>

</partialTrustVisibleAssemblies>

</securityPolicy>

</system.web>

Each entry in the partialTrustVisibleAssemblies section identifies an assembly by assembly name. Each entry is also identified by a 320-character string that is the hexadecimal character representation of the public half of the signing key that is used on the conditional-APTCA attributed assembly. You do not have to specify a version attribute. Only the assembly name and public key token are required by the CLR.

For more information see, "Customizing the ASP.NET 4 Conditional APTCA List" in [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).

[Creating Policy Files That Have Custom Permission Sets](javascript:void(0))

You can change the policy files or create new ones by using custom permission sets. For example, you can copy the contents of the Web\_hightrust.config file and assign permission to make OLEDB connections by first adding the [OleDbPermission](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbpermission.ASPX) class to the SecurityClasses section of the policy file, as shown in the following example:

<SecurityClass Name="OleDbPermission"

Description="System.Data.OleDb.OleDbPermission, System.Data, Version=4.0.0.0,

Culture=neutral, PublicKeyToken=b77a5c561934e089" />

You can then specify the parameters for the specified [OleDbPermission](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbpermission.ASPX) type, including restrictions for OLEDB connection strings. Next, you can specify which permission sets include the OleDbPermission security class. You do this by adding an IPermission element to the PermissionSet element that has the name attribute set to ASP.Net in the trust-policy file, as shown in the following example:

<PermissionSet

class="NamedPermissionSet"

version="1"

Name="ASP.Net">

<IPermission

class="OleDbPermission"

version="1"

Unrestricted="true" />

</PermissionSet>

Some permissions, such as the OleDbPermission permission, enable you to specify additional restrictions that narrow the access that is granted or denied. For example, the OleDbPermission permission enables you to grant access to make connections using the OLE DB .NET Framework data provider, but with restrictions on which OLEDB connection strings are allowed. The following example shows how to specify that only OLEDB connections to the Access database named catalog.mdb are allowed.

<IPermission class="OleDbPermission" version="1">

<add ConnectionString=

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\access\_data\catalog.mdb""

KeyRestrictions=""data source=;user id=;password=;"

KeyRestrictionBehavior="AllowOnly"/>

</IPermission>

You can save your updated trust-policy file and replace it with the current Web\_hightrust.config file. Or you can create a new trust-policy file and specify it as the policy file for the High trust level, or create a new trust level as shown in the following example:

<trustLevel name="HighCustom"

policyFile="web\_highcustom.config"/>

Using this trust level definition, you can set the partial-trust policy for an ASP.NET application to use HighCustom as shown in the following example:

<trust level="HighCustom" />

To preserve the default settings, ASP.NET includes two copies of each file that contains trust-level settings. One copy is named with the file name extension .config. The .config file contains the settings for each trust level that is used by the system. The second copy is named with the file name extension .config.default and contains the default settings for the related trust level. If the current trust-level settings have been modified and you want to restore the default settings, you can replace the contents of the .config file with the contents of the .config.default file.

[Using ASP.NET 2.0 CAS Behavior](javascript:void(0))

You can configure ASP.NET 4 applications to use ASP.NET 2.0 CAS behavior.

In ASP.NET 4, the trust element provides the legacyCasModel attribute, which is set to false by default. Setting this attribute to true configures an ASP.NET application to use most (although not all) the ASP.NET CAS behavior from versions earlier than ASP.NET 4. For information about using ASP.NET CAS behavior in versions earlier than ASP.NET 4, see [Homogeneous Application Domains](http://msdn.microsoft.com/en-us/library/dd984947.ASPX#homogeneousapplicationdomains).

In the .NET Framework 4.0, regardless of the CAS model that is in effect for an ASP.NET application, partial-trust code is not allowed to perform security asserts. For more information, see "Configuring ASP.NET 4 Applications to Use the ASP.NET 2.0 CAS Model" in [Code Access Security in ASP.NET 4 Applications](http://msdn.microsoft.com/en-us/library/dd984947.ASPX).