**Introduction**

As secure information is the key to efficient web programming, web application programmers always have security concerns. This article will explain how to secure your website using ASP.NET Form **Authentication**.

This article assume that the reader is already familiar with ASP.NET programming.

**Keywords**

* *web.config*: Application configuration files contain settings specific to an application. This file contains the configuration settings that the common language runtime reads (such as the assembly binding policy, remoting objects, and so on), and settings that the application can read [MSDN].
* **Authorization**: The purpose of **authorization** is to determine whether an identity should be granted the requested type of access to a given resource [MSDN].
* **Authentication**: **Authentication** is the process of discovering and verifying the identity of a principal, by examining the user's credentials and validating those credentials against some authority. The information obtained during **authentication** is directly usable by your code. That is, once the identity of the principal is discovered, you can use the .NET Framework role-based security to determine whether to allow that principal to access your code [MSDN].
* **What is authentication ?**  
  Authentication is process of verifying the identity of user.
* **Example :** suppose we have two types of user(normal and admin)to a website.  
  when the user tries to access the website ,we ask them to access the website ,  
  we ask again to log in. This is authentication part.
* **Types of Authentication :** Windows Authenticaton,Forms Authentication.
* **When it take place ?**  
  authentication always proceed to autherization,even if our application lets anonymous  
  users connect and use the application ,it still authenticates them as anonymous.
* **What is Authorization ?**  
  Authorization is process of checking whether the user has access rights to the system.
* **Example :** once we know the user is valis,then we determine to which pages   
  the user should not be able to access admin pages.This is Authorization part.
* **Types of Authorization :** File Authorization,URL Authorization.
* **When it take place ?**  
  Authorization take place after Authentication.

# Difference between Authentication and Authorization

## Authentication

What is ASP.Net AuthenticationAuthentication is the process of verifying the identity of a user by obtaining some sort of credentials and using those credentials to verify the user's identity. If the credentials are valid, the authorization process starts. Authentication process always proceeds to Authorization process.

## ASP.Net Authentication

The ASP.NET authentication scheme that is used to identify users who view an ASP.NET application. An ASP.net application has two separate authentication levels because all requests coming through IIS before it handled by ASP.NET. After IIS authentication schemes ASP.NET implements additional authentication schemes. They are :

**Windows Authentication   
Forms Authentication   
Passport Authentication**

The mode attribute specifies the authentication scheme.

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

**None Authentication**

You can specify "None" as the authentication provider when requests are not authenticated at all or if you plan to develop custom authentication code.

When you need "None" authentication, use the following Web.config configuration:

<system.web>

<authentication mode="None" />

</system.web>

## Authorization

Authorization is the process of allowing an authenticated users to access the resources by checking whether the user has access rights to the system. Authorization helps you to control access rights by granting or denying specific permissions to an authenticated user.

### Asp.Net Authorization

ASP.NET allows two ways to authorize access to a given resources, they are URL authorization and File authorization

**URL authorization**

URL authorization maps users and roles to URLs in ASP.NET applications

**File authorization**

File authorization validate the ACL (access control list) of the .aspx or .asmx handler file to determine whether a user should have access to the file.

**How to implement Authorization ?**

The following example shows a sample implementation of Authorization logic in web.config file.

<authorization>

<allow roles="Administrators" />

<deny users="\*" />

</authorization>

Difference between authentication and authorization.

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| It is the process of verifying the identity of a user. | It is the process of checking whether the user has the access rights to the system. |
| It always proceeds to authorization. | It is the process of allowing an authenticated user access to resources. |
| It has two separate levels because all the requests coming through the IIS before it is handled. | It allows two ways to authorize the access to a given resources. |
| They have additional schemes like windows authentication, forms authentication and passport authentication. | The two ways are URL authorization and File authorization. |

**Background**

I’ve searched so many sites for a code that I can with the help of it, secure websites from unauthorized access. After searching C# books, I found some nice code that helped me to create this simple application. Hope it can help as a basic architecture.

There are three kinds of **authentication** in ASP.NET:

1. **Form**,
2. **Windows**, and
3. **Passport**.

This article will focus on the first type.

Form **authentication** is cookie based, as ASP.NET places a cookie in the client machine in order to track the user. If the user requests a secure page and has not logged in, then ASP.NET redirects him/her to the login page. Once the user is authenticated, he/she will be allowed to access the requested page.

deny = \* means deny everyone

deny = ? means deny unauthenticated users

The UrlAuthorizationModule determines whether to grant or deny access to a requested resource for a particular identity based on the URL authorization rules defined in the application's configuration. The authorization rules are spelled out in the [<authorization> element](http://msdn2.microsoft.com/en-us/library/8d82143t.aspx) in the form of <allow> and <deny> child elements. Each <allow> and <deny> child element can specify:

* A particular user
* A comma-delimited list of users
* All anonymous users, denoted by a question mark (?)
* All users, denoted by an asterisk (\*)

The following markup illustrates how to use the URL authorization rules to allow users Tito and Scott and deny all others:

<authentication mode="Forms">

<forms loginUrl="~/Login.aspx" defaultUrl="~/Default.aspx" name=".ASPXFORMSAUTH">

<credentials passwordFormat="Clear">

<user name="kamlesh" password="admin"></user>

</credentials>

</forms>

</authentication>

if (FormsAuthentication.Authenticate(txt\_username.Text, txt\_password.Text))

{

//you can set cookie

FormsAuthentication.SetAuthCookie(txt\_username.Text, false);

//redirect when user is authenticated

FormsAuthentication.RedirectFromLoginPage(txt\_username.Text, false);

}

else

{

//invalid login

}

<authorization>

<allow users="Tito, Scott" />

<deny users="\*" />

</authorization>

The <allow> element defines what users are permitted - Tito and Scott - while the <deny> element instructs that all users are denied.

**Note:** The <allow> and <deny> elements can also specify authorization rules for roles. We will examine role-based authorization in a future tutorial.

The following setting grants access to anyone other than Sam (including anonymous visitors):

<authorization>

<deny users="Sam" />

</authorization>

To allow only authenticated users, use the following configuration, which denies access to all anonymous users:

<authorization>

<deny users="?" />

</authorization>

**Using the code**

In the *web.config*, change the mode of **authentication** to Forms, then add loginUrl="*your default page*".

In this section, you will set the default page of the system. The default page is the page that the system will redirect the user to, whenever a fault happens while the user tries to access a secured page.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/13872/Form-authentication-and-authorization-in-ASP-NET)

<!--***AUTHENTICATION***

*This section sets the* ***authentication*** *policies*

*of the application. Possible modes are "Windows",*

*"Forms", "Passport" and "None"*

*"None" No* ***authentication*** *is performed.*

*"Windows" IIS performs* ***authentication*** *(Basic,*

*Digest, or Integrated Windows) according to*

*its settings for the application.*

*Anonymous access must be disabled in IIS.*

*"Forms" You provide a custom form (Web page)*

*for users to enter their credentials, and then*

*you authenticate them in your application.*

*A user credential token is stored in a cookie.*

*"Passport"* ***Authentication*** *is performed via*

*a centralized* ***authentication*** *service provided*

*by Microsoft that offers a single logon*

*and core profile services for member sites.*

-->

<**authentication** mode="Forms">

<forms loginUrl="Login.aspx">

</forms>

</**authentication**>

This section of the *web.config* determines the users who will be authorized to or denied from the website. The default value <deny users="?" /> means to deny any anonymous (unauthenticated) user trying to access the website. However, this value can be changed. E.g., <deny users="john”, “smith”, “Ahmed” /> means to deny the users: john, smith and Ahmed from accessing this website - it is a black list- or you can say <deny users="\*" /> <allow users="john”, “smith”, “Ahmed” /> which means, deny all users except john, smith, and Ahmed.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/13872/Form-authentication-and-authorization-in-ASP-NET)

<!--***AUTHORIZATION***

*This section sets the* ***authorization*** *policies*

*of the application. You can allow or deny access*

*to application resources by user or role.*

*Wildcards: "\*" mean everyone, "?" means anonymous*

*(unauthenticated) users.*

-->

<**authorization**>

<deny users="?" /> <!-- *Allow all users* -->

<deny users="\*" /> <!-- *block all users* -->

<!-- *<allow users="[comma separated list of users]"*

*roles="[comma separated list of roles]"/>*

*<deny users="[comma separated list of users]"*

*roles="[comma separated list of roles]"/>*

-->

</**authorization**>

**Roles**

In some business websites, multiple employees would need access to a system in order to do specific tasks. However, each employee would have a specific role, and specific operations to do, according to the nature of his/her job or security level. E.g., an HR manager might not allowed to view the data of the seals department.

ASP.NET provides the concept of roles that gives each role a different view on specific pages.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/13872/Form-authentication-and-authorization-in-ASP-NET)

<location path="HRpages">

<system.web>

<**authorization**>

<allow roles="HR" />

<deny users="\*" />

</**authorization**>

</system.web>

</location>

<location path="salesPages">

<system.web>

<**authorization**>

<allow roles="sales" />

<deny users="\*" />

</**authorization**>

</system.web>

</location>

location here means the folder name which holds the *.aspx* for some specific role. As the example shows,<location path="HRpages"> means that all .aspx files under the *HRpages* folder are protected. <allow roles="HR" /><deny users="\*" /> mean deny every one from accessing pages under *HRpages* except those having the HR role.

**Login.aspx.cs**

This section will show the code that reads the password and the user name from *login.aspx* and redirects the user to a specific page according to his/her role.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/13872/Form-authentication-and-authorization-in-ASP-NET)

private void Submit1\_Click (object sender, System.EventArgs e)

{

if(this.TextBox\_username.Text.Trim()== "HR\_manager"

&& this.TextBox\_password.Text.Trim() == "password")

{

*// Success, create non-persistent* ***authentication*** *cookie.*

Forms**Authentication**.SetAuthCookie(

this.TextBox\_username.Text.Trim(), flase);

Forms**Authentication**Ticket ticket1 =

new Forms**Authentication**Ticket(

1, *// version*

this.TextBox\_username.Text.Trim(), *// get username from the form*

DateTime.Now, *// issue time is now*

DateTime.Now.AddMinutes(10), *// expires in 10 minutes*

false, *// cookie is not persistent*

"HR" *// role assignment is stored*

*// in userData*

);

HttpCookie cookie1 = new HttpCookie(

Forms**Authentication**.FormsCookieName,

Forms**Authentication**.Encrypt(ticket1) );

Response.Cookies.Add(cookie1);

*// 4. Do the redirect.*

String returnUrl1;

*// the login is successful*

if (Request.QueryString["ReturnUrl"] == null)

{

returnUrl1 = "HRpages/HR\_main.aspx";

}

*//login not unsuccessful*

else

{

returnUrl1 = Request.QueryString["ReturnUrl"];

}

Response.Redirect(returnUrl1);

}

}

The object ticket1 is of type Forms**Authentication**Ticket and provides a means of creating and reading the values of a forms **authentication** cookie. The previous code will redirect the user HR\_manager after checking his/her password. If the password is correct then it will create a cookie to track the user and encrypt the content of this cookie.

One of the Forms**Authentication**Ticket constructors takes the following parameters:

* version - the version number.
* name - the user name associated with the ticket.
* issueDate - the time at which the cookie was issued.
* expiration - the expiration date for the cookie.
* isPersistent - true if the cookie is persistent; otherwise, false.
* userData - user-defined data to be stored in the cookie [MSDN].

**Authentication in ASP.NET**

There are two closely interlinked concepts at the heart of security for distributed applications - authentication and authorization. Authentication is the process of obtaining some sort of credentials from the users and using those credentials to verify the user's identity. Authorization is the process of allowing an authenticated user access to resources. Authentication is always precedes to Authorization; even if your application lets anonymous users connect and use the application, it still authenticates them as being anonymous.   
ASP.net provides flexible set of alternatives for authentication. You can perform authentication yourself in code or delegate authentication to other authorities (such as Microsoft Passport). In fact sometimes it seems ASP.net authentication is a bit too flexible; it can be difficult for a new developer to know just where to start. In this article, we review the settings in ASP.net and Internet Information Services (IIS) that control authentication and authorization in ASP.net applications.

An ASP.net application has two separate authentication layers. That is because ASP.net is not a standalone product. Rather it is a layer on top of IIS. All requests flow through IIS before they are handed to ASP.net. As a result, IIS can decide to deny access without the ASP.net process even knowing that someone requested a particular page. Here is an overview of the steps in the joint IIS and ASP.net authentication process.

1. IIS first checks to make sure the incoming request comes from an IP address that is allowed access to the domain. If not it denies the request.
2. Next IIS performs its own user authentication if it configured to do so. By default IIS allows anonymous access, so requests are automatically authenticated, but you can change this default on a per - application basis with in IIS.
3. If the request is passed to ASP.net with an authenticated user, ASP.net checks to see whether impersonation is enabled. If impersonation is enabled, ASP.net acts as though it were the authenticated user. If not ASP.net acts with its own configured account.
4. Finally the identity from step 3 is used to request resources from the operating system. If ASP.net authentication can obtain all the necessary resources it grants the users request otherwise it is denied. Resources can include much more than just the ASP.net page itself you can also use .Net's code access security features to extend this authorization step to disk files, Registry keys and other resources.

As you can see several security authorities interact when the user requests and ASP.net page. If things are not behaving the way you think they should, it can be helpful to review this list and make sure you have considered all the factors involved

**Authentication providers**

Assuming IIS passes a request to ASP.net, what happens next? The answer depends on the configuration of ASP.net itself. The ASP.net architecture includes the concept of and authentication provider a piece of code whose job is to verify credentials and decide whether a particular request should be considered authenticated. Out of the box ASP.net gives you a choice of three different authentication providers.

* The windows Authentication provider lets you authenticates users based on their windows accounts. This provider uses IIS to perform the authentication and then passes the authenticated identity to your code. This is the default provided for ASP.net.
* The passport authentication provider uses Microsoft's passport service to authenticate users.
* The forms authentication provider uses custom HTML forms to collect authentication information and lets you use your own logic to authenticate users. The user's credentials are stored in a cookie for use during the session.

Selecting an authentication provider is as simple as making an entry in the web.config file for the application. You can use one of these entries to select the corresponding built in authentication provider:

<authentication mode="windows">  
authentication mode="passport">  
<authentication mode="forms">

ASP.net also supports custom authentication providers. This simply means that you set the authentication mode for the application to none, then write your own custom code to perform authentication. For example, you might install an ISAPI filter in IIS that compares incoming requests to list of source IP addresses, and considers requests to be authenticated if they come from an acceptable address. In that case, you would set the authentication mode to none to prevent any of the .net authentication providers from being triggered.

The fig below illustrates the authorization and authentication mechanisms provided by ASP.NET and IIS.

**Windows authentication and IIS**

If you select windows authentication for your ASP.NET application, you also have to configure authentication within IIS. This is because IIS provides Windows authentication. IIS gives you a choice for four different authentication methods:

Anonymous, basic digest, and windows integrated

If you select anonymous authentication, IIS doesn't perform any authentication, Any one is allowed to access the ASP.NET application.

If you select basic authentication, users must provide a windows username and password to connect. How ever this information is sent over the network in clear text, which makes basic authentication very much insecure over the internet.

If you select digest authentication, users must still provide a windows user name and password to connect. However the password is hashed before it is sent across the network. Digest authentication requires that all users be running Internet Explorer 5 or later and that windows accounts to stored in active directory.

If you select windows integrated authentication, passwords never cross the network. Users must still have a username and password, but the application uses either the Kerberos or challenge/response protocols authenticate the user. Windows-integrated authentication requires that all users be running internet explorer 3.01 or later Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. Kerberos is a solution to network security problems. It provides the tools of authentication and strong cryptography over the network to help to secure information in systems across entire enterprise

**Passport authentication**

Passport authentication lets you to use Microsoft's passport service to authenticate users of your application. If your users have signed up with passport, and you configure the authentication mode of the application to the passport authentication, all authentication duties are offloaded to the passport servers.

Passport uses an encrypted cookie mechanism to indicate authenticated users. If users have already signed into passport when they visit your site, they'll be considered authenticated by ASP.NET. Otherwise they'll be redirected to the passport servers to log in. When they are successfully log in, they'll be redirected back to your site

To use passport authentication you have to download the Passport Software Development Kit (SDK) and install it on your server. The SDK can be found at http://msdn.microdoft.com/library/default.asp?url=/downloads/list/websrvpass.aps. It includes full details of implementing passport authentication in your own applications.

**Forms authentication**

Forms authentication provides you with a way to handle authentication using your own custom logic with in an ASP.NET application. The following applies if you choose forms authentication.

1. When a user requests a page for the application, ASP.NET checks for the presence of a special session cookie. If the cookie is present, ASP.NET assumes the user is authenticated and processes the request.
2. If the cookie isn't present, ASP.NET redirects the user to a web form you provide
3. You can carry out whatever authentication, checks you like in your form. When the user is authenticated, you indicate this to ASP.NET by setting a property, which creates the special cookie to handle subsequent requests.

**Configuring Authorization**

After your application has authenticated users, you can proceed to authorize their access to resources. But there is a question to answer first: Just who is the user to whom your are grating access? It turns out that there are different answers to that question, depending on whether you implement impersonation. Impersonation is a technique that allows the ASP.NET process to act as the authenticated user, or as an arbitrary specified user

ASP.NET impersonation is controlled by entries in the applications web.config file. The default setting is "no impersonation". You can explicitly specify that ASP.NET shouldn't use impersonation by including the following code in the file

<identity impersonate="false"/>

With this setting ASP.NET does not perform impersonation. It means that ASP.NET will runs with its own privileges. By default ASP.NET runs as an unprivileged account named ASPNET. You can change this by making a setting in the processModel section of the machine.config file. When you make this setting, it automatically applies to every site on the server. To user a high-privileged system account instead of a low-privileged, set the userName attribute of the processModel element to SYSTEM. Using this setting is a definite security risk, as it elevates the privileges of the ASP.NET process to a point where it can do bad things to the operating system.

When you disable impersonation, all the request will run in the context of the account running ASP.NET: either the ASPNET account or the system account. This is true when you are using anonymous access or authenticating users in some fashion. After the user has been authenticated, ASP.NET uses it own identity to request access to resources.

The second possible setting is to turn on impersonation.

<identity impersonate="true"/>

In this case, ASP.NET takes on the identity IIS passes to it. If you are allowing anonymous access in IIS, this means ASP.NET will impersonate the IUSR\_ComputerName account that IIS itself uses. If you aren't allowing anonymous access,ASP.NET will take on the credentials of the authenticated user and make requests for resources as if it were that user. Thus by turning impersonation on and using a non-anonymous method of authentication in IIS, you can let users log on and use their identities within your ASP.NET application.

Finally, you can specify a particular identity to use for all authenticated requests

<identity impersonate="true" username="DOMAIN\username" password="password"/>

With this setting, all the requests are made as the specified user (Assuming the password it correct in the configuration file). So, for example you could designate a user for a single application, and use that user's identity every time someone authenticates to the application. The drawback to this technique is that you must embed the user's password in the web.config file in plain text. Although ASP.NET won't allow anyone to download this file, this is still a security risk if anyone can get the file by other means.

**Best practices**

Now that you know what the choices are for ASP.NET authentication, here are some points that tell which to choose.

* If there is nothing sensitive about the application, stick with no authentication in ASP.NET and anonymous authentication in IIS. That lets anyone who can reach the host computer use the application.
* If you have to authenticate users, there are several choices. If all users have accounts on your network, use Windows authentication in ASP.net with one of the strong IIS authentication settings. If users don't have network accounts, own custom authentication scheme is preferred, means forms authorization.
* If different users must have different privileges, impersonation in ASP.net configuration files needs to be turn on.

configuration>

    <system.web>

        <authentication mode="Forms">

            <forms name="TestAuthCookie" loginUrl="login.aspx"timeout="30">

                <credentials passwordFormat="Clear">

            <username="user1" password="pass1"/>

            <username="user2" password="pass2"/>

        </credentials>

        </forms>

    </authentication>

    <authorization>

        <deny users="?"/>

    </authorization>

    <compilation targetFramework="4.0"/>

    <pages controlRenderingCompatibilityVersion="3.5"ClientIDMode="AutoID"/>

    </system.web>

</configuration>

Now that we have specified some valid logon accounts, we need to actually specify that we want to password protect. For this example I have decided to password protect the entire web site starting at the root, so the optional attribute will not be used. We set the authorization to deny all non-authenticated users (deny users=”?”).

That’s all that is needed for the config.web file. If someone tries to access the site and the user has not already authenticated, they will be redirected to the login.aspx page.

This is only half the required process though. We now need to create the login.aspx page to actually allow the user to authenticate to our application.

Here is the complete source of the sample login.aspx page:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <%@ Page Language="C#" AutoEventWireup="true" CodeFile="login.aspx.cs" %>  <html>  <head runat="server"><title>Sample</title><head>  <body>      <form id="form1" runat="server">          Username: <asp:TextBox ID="txtUsername" runat="server" /><br>          Password:<asp:TextBox ID="txtPassword" runat="server" /><br>          <asp:Button ID="Button1" runat="server" onclick="Button1\_Click"Text="Login" /><br>          <asp:Label ID="lblStatus" runat="server" Text="Please login" />      </form>  </body>  </html> |

And here is the complete source of the login.aspx.cs file:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | using System;  using System.Web.UI.WebControls;  using System.Web.Security;    public partial class Default3 : System.Web.UI.Page  {      protected void Button1\_Click(object sender, EventArgs e)      {          if (FormsAuthentication.Authenticate(txtUsername.Text, txtPassword.Text))          {              lblStatus.Text = ("Welcome " + txtUsername.Text);              FormsAuthentication.RedirectFromLoginPage(txtUsername.Text, true);          }          else          {              lblStatus.Text = "Invalid login!";          }        }  } |

Let’s look at the login.aspx page first. This is fairly straight-forward HTML format. These aren’t actually straight HTML tags, but rather ASP.Net HTML controls that will render HTML page to the client browser (you can tell the difference because the runat=”server” tag at on the control). This is a form that accepts a username and password. When the submit button is clicked, this page executes the code within the login.aspx.cs page located in the subroutine named “Button1\_Click”

Inside the Button1\_Click method we use the FormsAuthentication object. The first line of the sub actually passes the entered username and password over to the object, which in turn compares this information to the values in the web.config file. If the values match, then the next line changes the label (just so we can see visually that it worked) then writes a cookie to the browser and redirects the user back to the original URL which was requested. The second value listed (“true”) tells the browser to persist the cookie. So if this user authenticates, closes their browser, opens it again, and tries the secure URL – they will still be authenticated.

If the username and password entered did not match, an error message is displayed to the screen and the visitor is allowed to enter a new username and password to try again.

This is a simple example and I don’t cover any of the advanced configurations or options, but with this sample code, you should have a basis to work with if you want to implement security in ASP.Net.

Authentication is the process of validating a user based on a set of credentials such as username, password, and e-mail address. Suppose you own a small Web development company that uses ASP.NET, and you want to give your users a secured area from where they can download or view additional resources such as tutorials. You would have to store crucial user data such as usernames and passwords (preferably in a database such as Microsoft Access or SQL Server) and then authenticate users based on those credentials with a help of the relevant ASP.NET code. This process involves a huge amount of work for developers, including such tasks as creating tables, stored procedures, and so on.

ASP.NET offers simpler ways to validate users—with little work required. By applying ASP.NET programming logic, you can store user data in XML files and then validate users using those files. If you have a limited number of users, you can store the credentials in a Web configuration file (Web.Config) instead. This article shows you how to apply ASP.NET user authentication using either a Web.Config file or an XML file. If you haven't already, you'll need to install Microsoft's ASP.NET Web Matrix, a free editor available for download from [http://www.asp.net](http://www.asp.net/).

### Authenticating Users Using a Web.Config File

Web.Config is the main configuration file that ASP.NET applications use for storing global parameters such as connection strings for databases, passwords, and so forth. You should save this file inside the root directory of your ASP.NET application. To perform authentication using the Web.Config file, you need to create a file as shown in Listing 1.1:

**Listing 1.1** Web.Config

<configuration>

<system.web>

<authentication mode = "Forms">

<forms>

<credentials passwordFormat = "Clear">

<user name ="abc" password = "123"/>

</credentials>

</forms>

</authentication>

<authorization>

<deny users = "?"/>

</authorization>

</system.web>

</configuration>

The usernames and passwords should be supplied inside the credentials tag, and the authentication mode should be set to Forms. The contents of Listing 1.1 are case sensitive and should be entered as shown. Further, the authorization section denies access to all anonymous users. Hence, only users whose credentials match those given in the Web.Config file can access the relevant Web page. The following table shows different kinds of tags and symbols you can use inside the authorization tag and their meanings.

|  |  |
| --- | --- |
| **Tag** | **Meaning** |
| <deny users = "?"/> | Denies access to all anonymous users |
| <deny users = "\*"/> | Denies access to both anonymous and authenticated users |
| <allow users = "?"/> | Allows access to all anonymous users |
| <allow users = "\*"/> | Allows access to both anonymous and authenticated users |

The next step is to create an ASP.NET page that contains the real code for verifying a user. Listing 1.2 is a sample code snippet:

**Listing 1.2** Login.aspx

if IsValid then

if FormsAuthentication.Authenticate(txtUsername.Text,

txtPassword.Text) Then

lblStatus.Text = "Username and Passwords are correct"

else

lblStatus.Text = "Invalid Username and Passwords"

end if

end if

This code uses the Authenticate method of the FormsAuthentication class to validate a user. The Authenticate method takes the corresponding text fields as parameters. If the entered data is incorrect, it executes the statement inside the else part. You can also redirect a user to another Web site if the entered data is correct by slightly modifying Listing 1.2 to the code shown in Listing 1.3:

**Listing 1.3**

if IsValid then

if FormsAuthentication.Authenticate(txtUsername.Text,

txtPassword.Text) Then

Response.Redirect("http://www.developer.com")

else

lblStatus.Text = "Invalid Username and Passwords"

end if

end if

|  |
| --- |
| Authentication means validating a user based on a set of credentials, such as e-mail, username, and password. Authorization occurs after authentication. Authorization requires specifying access restrictions and permissions for your users. Hence, these terms are different but interrelated. |

### Authenticating Users Using an XML File

Even though you can easily authenticate users by using a Web.Config file, it is not advisable for sites with a large number of users. It is also very difficult to implement an automated system that directly adds users to the Web.Config file. A Web developer should manually add new usernames and passwords to the file for each new user. To avoid this hassle, ASP.NET provides a facility for authenticating users using an XML file. For this purpose, you have to create both a Web.Config file (Listing 1.4) and an XML file (Listing 1.5):

**Listing 1.4** Web.Config

<configuration>

<system.web>

<authentication mode = "Forms">

<authorization>

<deny users = "?"/>

</authorization>

</system.web>

</configuration>

**Listing 1.5** pwd.xml

<passwordlist>

<user>

<name>bob</name>

<pwd>123</pwd>

</user>

<user>

<name>mark</name>

<pwd>456</pwd>

</user>

<user>

<name>peter</name>

<pwd>789</pwd>

</user>

</passwordlist>

|  |
| --- |
| XML is a case-sensitive language. |

The next step is to create an ASP.NET page. Because it has to check two credentials (username and password), you have to add two TextBox controls and a Button control to the form. Double-click the button control and add the code given in Listing 1.6:

**Listing 1.6**

If IsValid then

If XMLAuthentication(txtUsername.Text,txtPassword.Text) Then

Response.Redirect("http://www.developer.com")

End If

End If

Listing 1.6 passes the two control IDs as parameters to the XMLAuthentication method. This method will contain the real code to authenticate users from your XML file. Further, if the username and password match with that of the XML file, the user will be redirected to the developer.com home page. The source code for this method is given in Listing 1.7:

**Listing 1.7**

Dim dstPwd as DataSet

Dim dtblPwd as DataTable

Dim users() as DataRow

dstPwd = New DataSet()

dstPwd.ReadXML(MapPath("Pwd.xml"))

dtblPwd = dstPwd.Tables(0)

users = dtblPwd.Select("name = '"& strUsername & "' ")

if users.Length > 0 Then

if users(0)("pwd") = strPwd Then

Return True

Else

lblStatus.Text = "Invalid Password"

End If

Else

lblStatus.Text = "Username does not exist"

End If

Return False

End Function

In Listing 1.7, the XML file is loaded by using the built-in ReadXML() method, and the XMLAuthentication method checks both the username and password. The method displays the relevant messages in the label control.