Lecture 1

A security of the web application

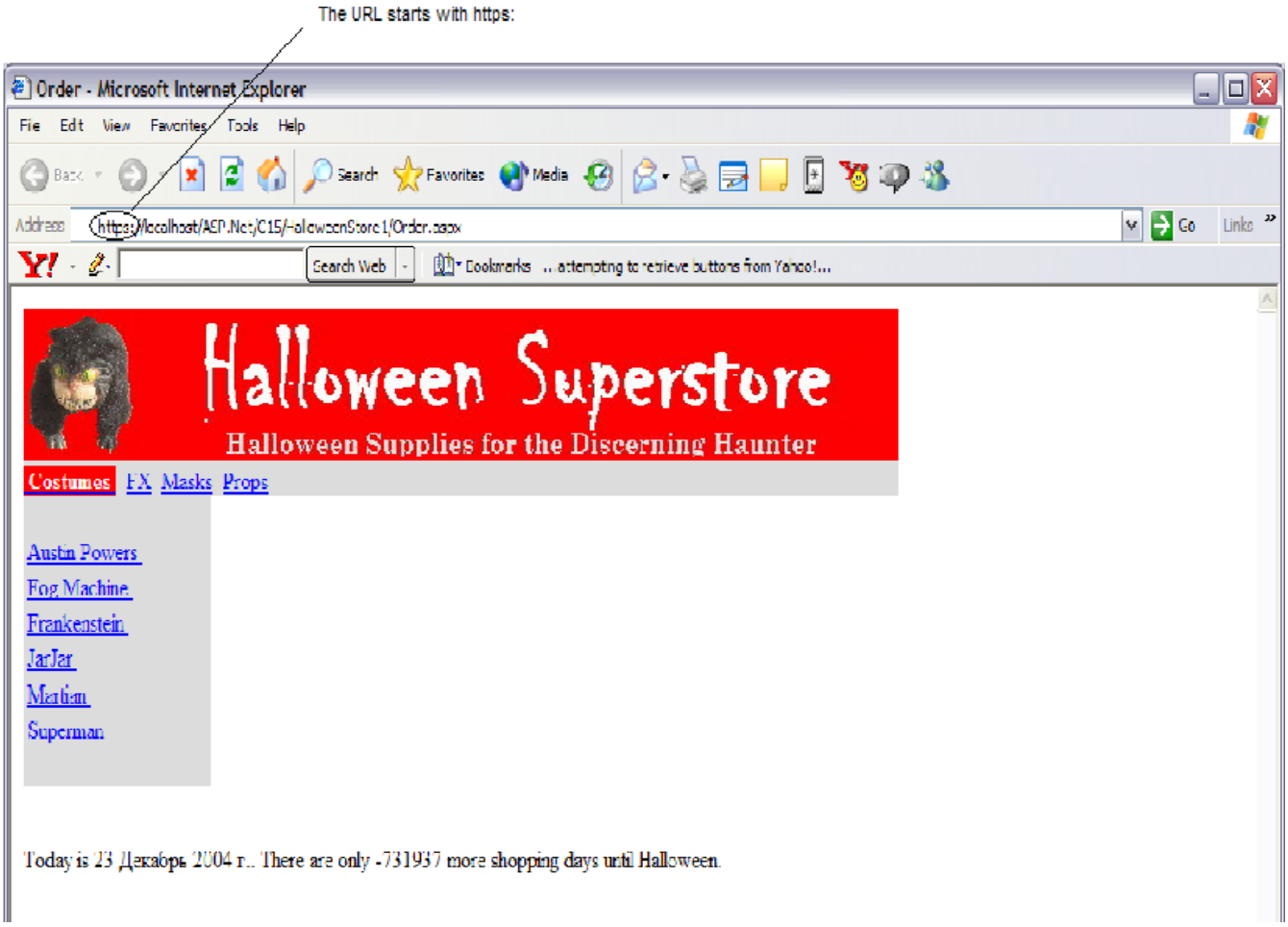
Security is one of the most important concerns for any developer of e-commerce web application. To secure a web application, you must make sure that:

1. Unauthorized users don't have access to it.
2. Private data that's sent between the client and the server can't be intercepted.

Secure Socket Layers

To prevent other from reading data that's transmitted over the Internet, you can use the ***Secure Sockets Layers***, or ***SSL***.

SSL is an Internet protocol that lets you transmit data over the Internet using data encryption.



If it starts with ***https*** rather than ***http***, then you're transmitting data over a secure connection. In addition, a small lock icon appears in the lower right corner of the browser window.

With regular HTTP, all data is sent as unencrypted plain text. As a result, if a hacker intercepts this data, it is easy to read. With a secure connection, however, all data that's transmitted between the client and the server is encrypted. Although a hacker can still intercept this data, he won't be able to read it with breaking the encryption code.

A digital secure certificate

To use SSL to transmit data, the client and server use ***digital secure certificate (DSC)***. Digital secure certificates serve two purposes:

1. They establish the identity of the server or client.
2. They provide the information needed to encrypt data before it's transmitted.

By default, browsers are configured to accept certificate that comes from trusted sources. If a browser doesn't recognize a certificate as coming from a trusted source, however, it informs the user and lets the user view the certificate. Then, the user can determine whether the certificate should be considered valid. If the user chooses to accept the certificate, the secure connection is established.

Sometimes, a server may want the client to authenticate itself with ***SSL client authentication***. Although this isn't as common as ***SSL sever authentication,*** it is used occasionally. For example, a bank might want to use SSL client authentication to make sure it's sending sensitive information such as account numbers and balances to the correct person. To implement this type of authentication, a digital certificate must be installed on the client.

How to determine if a digital certificate is installed on your server

If IIS is running on your local machine, chances are that a certificate hasn't been installed. But if IIS is running on a server on a network, the certificate has been installed and you can view it, using a procedure:

1. Start the IIS program
2. Expand the node for the server and the Web Sites node and then right-click on the *Default Web Site* node and select *Properties* to display the *Properties* dialog box.
3. Click the *Directory Security* tab. If a digital secure certificate is installed, the *View Certificate* button will be enabled. Click this button to display the dialog box.

Types of digital secure certificates

Certificate	Description
Server certificate	Issued to trusted servers so that client computers can connect to them using secure connection.
Client certificate	Issued to trusted clients so that server computers can confirm their identity.

How to get and use a digital secure certificate

If you want to develop an ASP.NET application that uses SSL to secure client connections, you must first obtain a digital secure certificate from trusted source, for example, presented below:

www.verisign.com
www.geotrust.com
www.entrust.com
www.thawte.com

These ***certification authorities***, or ***CA***, verify that person or company requested the certificate is a valid person or company by checking with a ***registration authority***, or ***RA***. To obtain a digital secure certificate, you'll need to provide a registration authority with information about yourself or your company. Once the registration authority approves the request, the certification authority can issue the digital secure certificate.

A DSC from a trusted sources isn't free, and the cost of the certificate will depend on a variety factors such as the level of security. In particular, you'll need to decide what ***SSL strength*** you want the connection to support. SSL strength refers to the level of encryption that the secure connection uses when it transmits data.

SSL strengths

Strength	Pros and Cons
40-bit	Most browsers support it, but it's relatively easier to break the encryption code.
56-bit	It's thousands of times than 40-bits strength and most browsers support it, but it's still possible to break the encryption code
128-bit	It's over a trillion times stronger that 40-bits strength, which makes it extremely difficult to break the encryption code, but it's more expensive and not all browsers support it

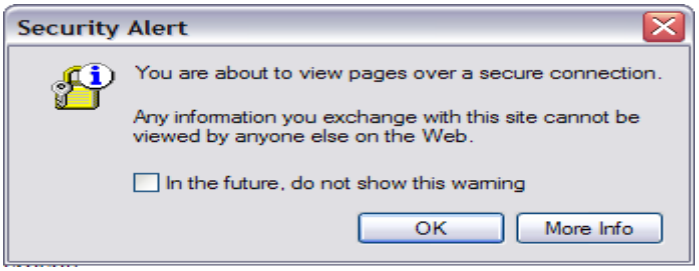
Most certification authorities provide a free trial certificate that can be used for testing purposes. The trial certificate typically expires after a relatively short period of time, such as two weeks or a month. You shouldn't run an ASP.NET application in a production environment using an expired certificate. When you're ready to deploy your application, then you should contact the certificate authority to obtain a valid certificate.

Another way to obtain certificates for testing purposes is to set up your own certification authority using Microsoft Certificate Services, which comes with Windows 2000 Server and Windows 2003. To get more information about Microsoft Certificate Services, you can review the online help for Windows 2000 Server or Windows Server 2003 or you can visit Microsoft's web site.

A secure connection

A picture below shows how to request a secure connection in an ASP.NET application.

A dialog box that may be displayed for a secure connection



A URL that requests a security connection

https://localhost/ASP.NET/HalloweenStore/Checkout1.aspx

A *Web.config* file that defines the *AppPath* settings

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="AppPath"
                value="//localhost/ASP.NET/HalloweenStore/" />
  </appSettings>

  <system.web>
  .
  .
  .
```

Code that retrieves the application path from the *Web.config* file

```
String url = "https:"
        + ConfigurationManager.AppSettings["AppPath"]
        + "CheckOut1.aspx";

Response.Redirect(url);
```

Code that returns to an unsecured connection

```
String url = "http:"
        + ConfigurationManager.AppSettings["AppPath"]
        + "Order.aspx";

Response.Redirect(url);
```

Description

- To request a secure connection, you must use an absolute URL that specifies *https* as the protocol. Once you establish a secure connection, you can use relative URL to continue using the secure connection.
- To return to an unsecured connection after using a secure connection, you must code an absolute URL that specifies the *http* protocol.
- Instead of coding the application's path each URL, you can store it in the *<appSettings>* section of the *web.config* file. That way, if the path changes, you can change it in just one location.
- You can use the *AppSettings* property of the *ConfigurationManager* class within the application to access the elements in the *appSettings* section of the *web.config* file.
- Depending on the security settings in your browser, a dialog box may be displayed before a secure connection is established. A dialog box may be displayed before a secure connection is closed.

How to force a page to use a secure connection

When you build a complete Web Application, you usually include navigation features such as menus or hyperlinks that guide the user from page to page. Unfortunately, users sometimes bypass your navigation features and access pages in your application directly. For example, a user might bookmark a page in your application and return to it later. Other users might simply type the URL of individual pages in your application into their browser's address bar. Some users do this innocently; others do it in an attempt to bypass your application's security features.

Because of that, a page that should use SSL to send or receive sensitive information shouldn't assume that a secure connection has been established. Instead, it should check for a secure connection and establish one if necessary. To do that, you can use the properties of the *HttpRequest* class.

To check for a secure connection, you use the *IsSecureConnection* property. Then, if the connection isn't secure, you can use the *Url* property to retrieve the URL for the page and modify that URL so it uses the *HTTPS* protocol. After you do that, you can use the *Redirect* method to redirect the browser using the new URL. Notice that you typically include this code at the beginning of the `Load` procedure for the page. That way, you can be sure that no other code is executed until a secure connection is established.

Properties of the *Httprequest* class for working with secure connections

Property	*Description*
IsSecureConnection	Returns *True* if the current connection is secure. Otherwise, returns *False*.
Url	The URL of the current request.

A *Load* procedure for a page that forces the page to use a secure connection

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!Request.IsSecureConnection)
   {
      string url = Request.Url.ToString().Replace("http:", "https:");

      Response.Redirect(url);
   }
}
```

Another way to force a page to use a secure connection is by setting a property of the page from IIS. You can also use this technique to change the minimum SSL strength that's required from 40-bit to 128-bit. When you use this technique, a user won't be able to access the page without using a secure connection. In other words, he won't be able to bypass the navigation features provided by your application.

Authentication and authorization

If you want to limit access to your ASP.NET application to certain users, you can use ***authentication*** to verify each user's identity. Then, once you have authenticated the user, you can use ***authorization*** to check if the user has the appropriate privileges for accessing a page to prevent unauthorized users to get accessing to your application.

There are three types of authentication you can use in ASP.NET application:

Windows-based authentication

- Causes the browser to display a login dialog box when the user attempts to access a restricted page.
- Is supported by most browsers.
- Is configured through the IIS management console.
- Uses Windows user accounts and directory rights to grant access to restricted pages.

Forms-based authentication

- Lets developers code a login form that gets the username and password.
- The username and password entered by the user are encrypted if the login page uses a secure connection.
- Doesn't rely on Windows user accounts. Instead, the application determines how to authenticate users.

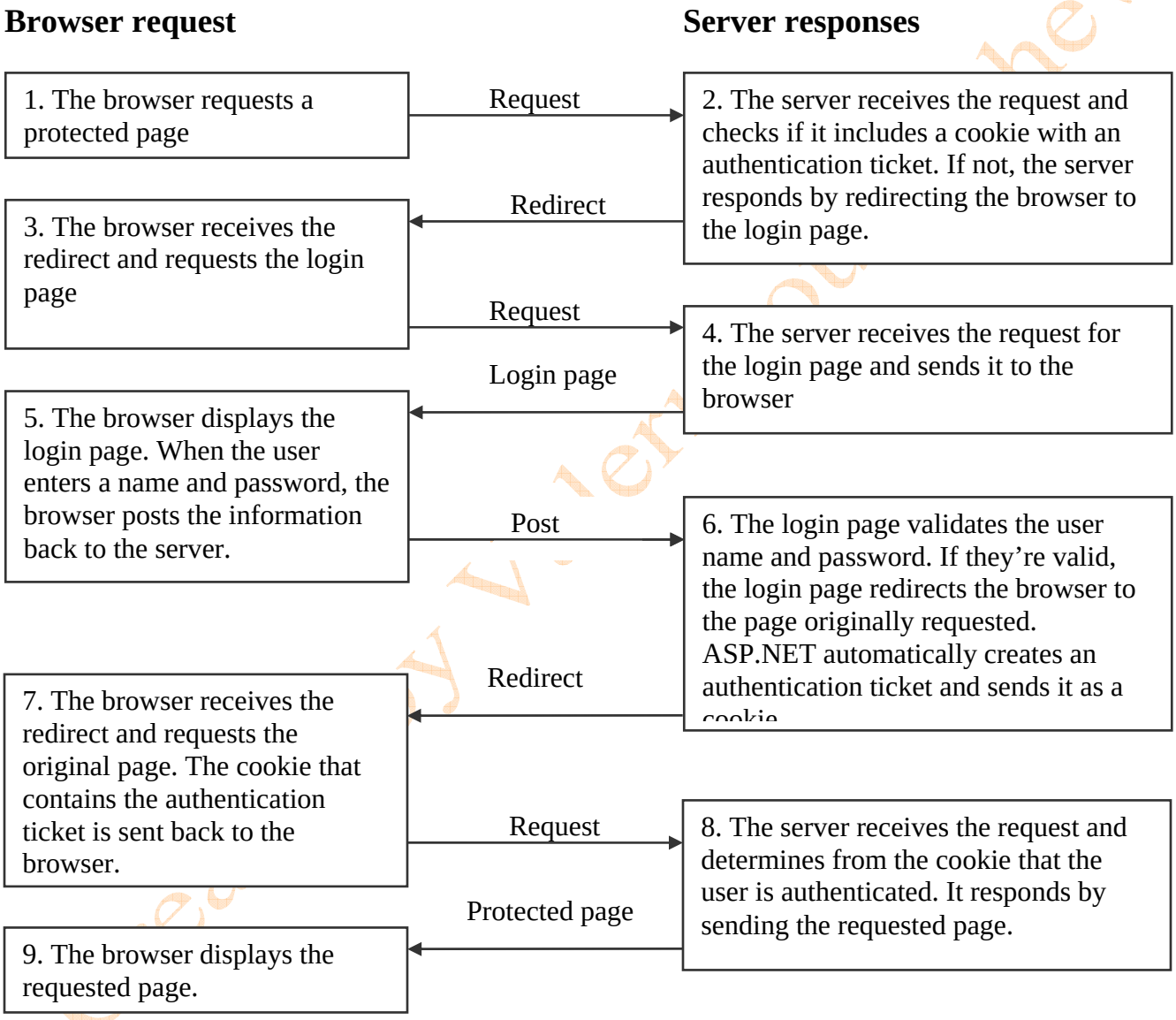
Passport authentication

- ***Passport*** is a centralized authentication service offered by Microsoft.
- Passport lets users maintain a single user account that lets them access any web site that participate in Passport. The advantage is that the user only has to maintain one username and password.

- Passport authentication isn't free. You must sign up for Passport authentication and pay a significant fee to use it in your applications. For more information about passport authentication, visit *www.passport.net*.

Forms-based authentication

Figures below shows a typical series of exchanges that occur between a web browser and a server when a user attempts to access a page that's protected by forms-based authentication.



Description

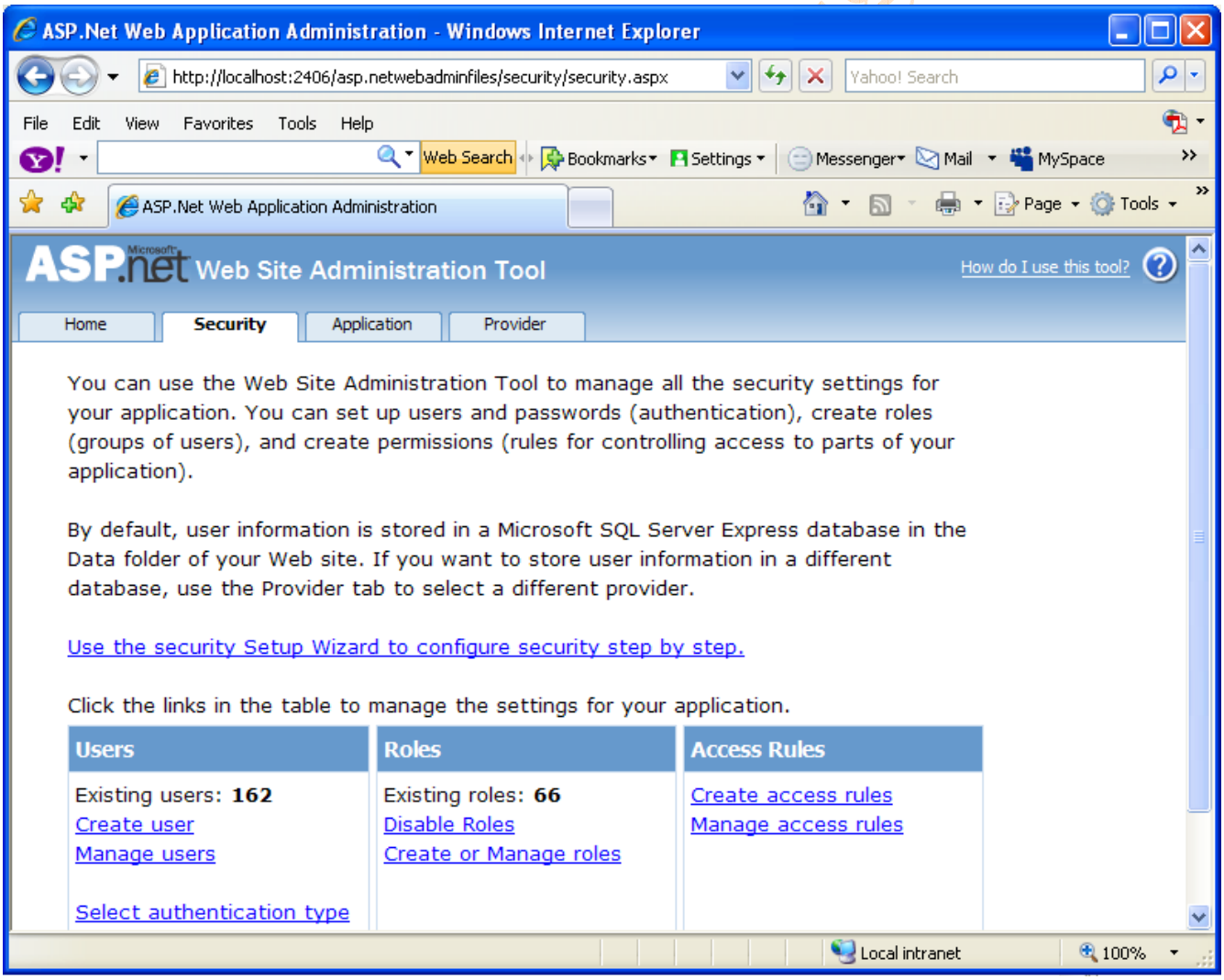
- When ASP.NET receives a request for a page that's protected using forms-based authentication from a user who has not been authenticated, the server redirects the user to the login page.
- To be authenticated, the user's computer must contain an ***authentication ticket***. By default, this ticket is stored as a session cookie.

- ASP.NET automatically creates an authentication ticket when the application indicates that the user should be authenticated. ASP.NET checks for the presence of an authentication ticket any time it receives a request for a protected page.
- The authentication ticket cookie can be made persistent. Then, the user will be authenticated automatically in future sessions, until the cookie expires.

Setting up authentication and authorization

By default, all pages of a web site can be accessed by all users whether or not they are authorized. As a result, if you want to restrict access to all or some of the pages of the web site, you need to set up authentication and authorization. With ASP.NET 2.0, you can use the ASP.NET Web Site Administration Tool (WSAT) to set up authentication and authorization.

To start the ASP.NET Web Site Administration Tool, you use the *Website* → *ASP.NET Configuration* command. This starts a web browser that displays the home page for this tool. Then, you can click on the *Security* tab to access a web page like presented below.

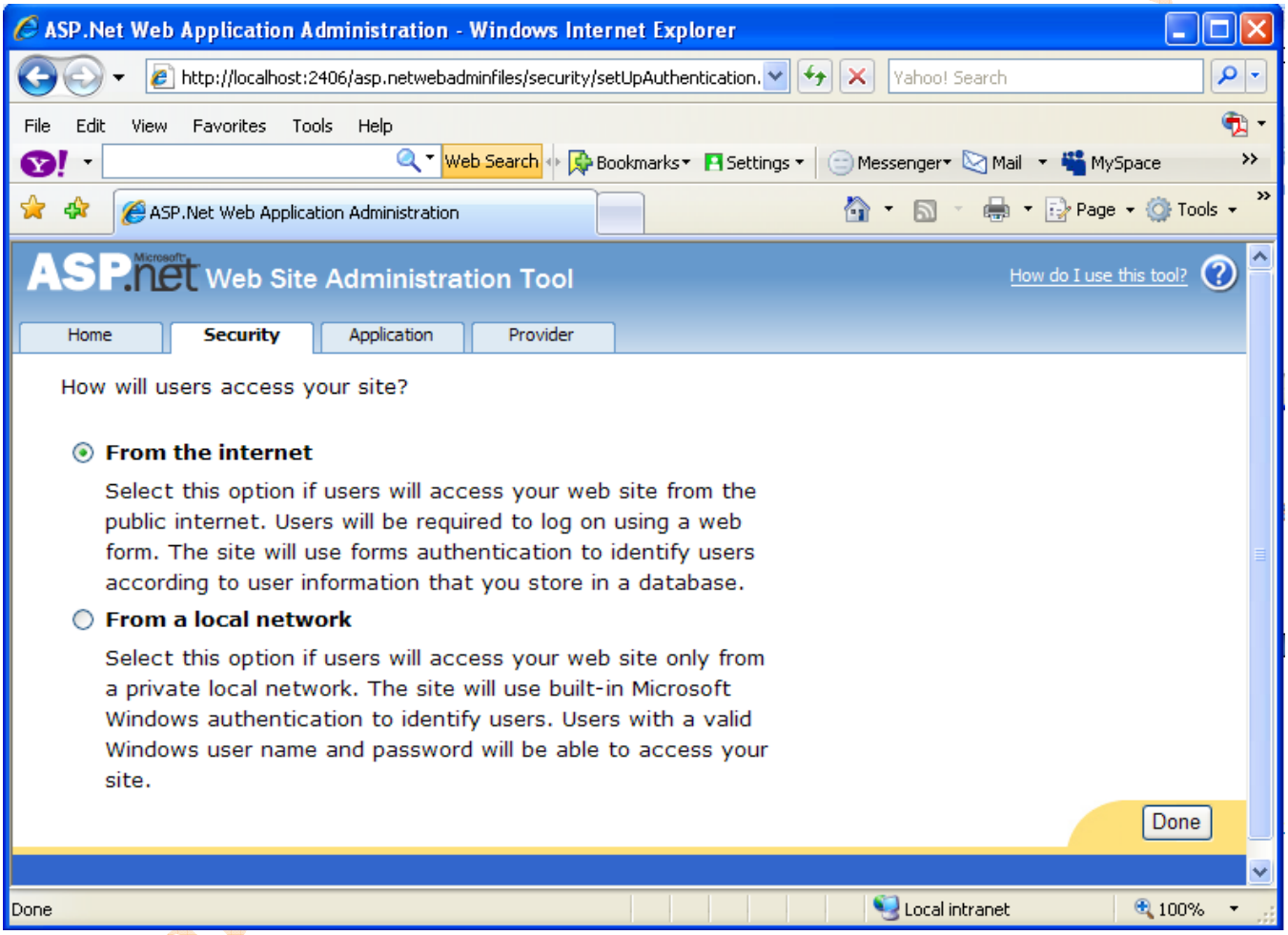


This page lets you set up users, create groups of users, known as roles, and create access rules that control access to parts of your application.

Forms-based authentication

By default, a web site is set up to use Windows authentication. If all users will be accessing your web site through a private local Windows network (an intranet), this option may be the easiest to implement because it uses built-in Windows dialog boxes to allow users to log in.

However, if any users will access your web site from the Internet, you'll need to switch to forms-based authentication. To do that, you can click on the *Select Authentication Type* link from the *Security* tab of the WSAT to display the page shown below:



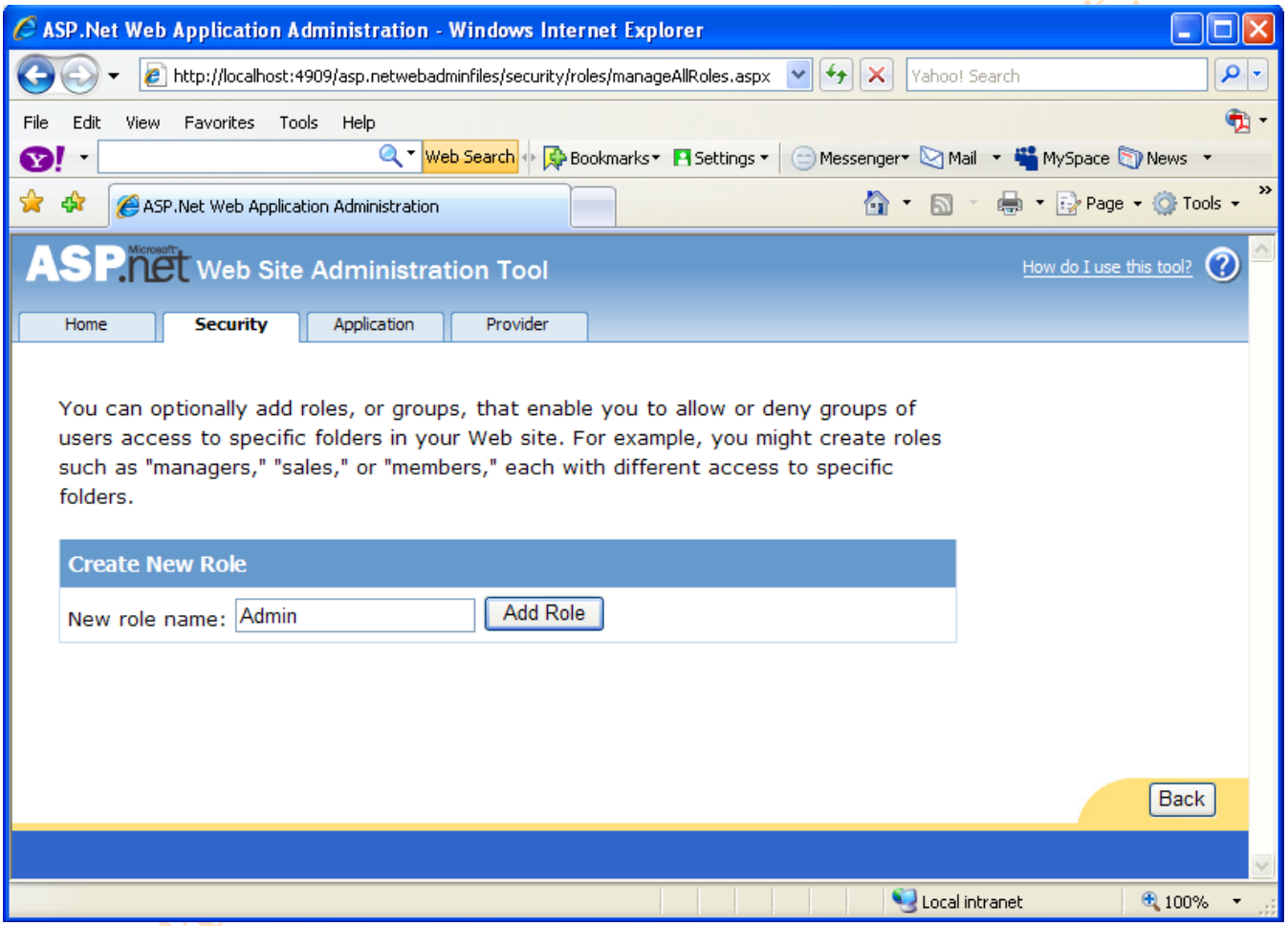
Then, you can select the *From the Internet* option. When you use this option, you'll need to create a web form that allows users to log in.

Creating and managing roles

Roles allow you to apply the same access rules to a group of users. Although roles are optional and are disabled by default, they make it easy to manage authentication. As a result, you'll typically want to enable roles. And since you use roles when you create users and access rules, it is often helpful to set up the roles before you create the users and access rules. That way, you don't have to go back later and edit your users and access rules so they are associated with the correct roles.

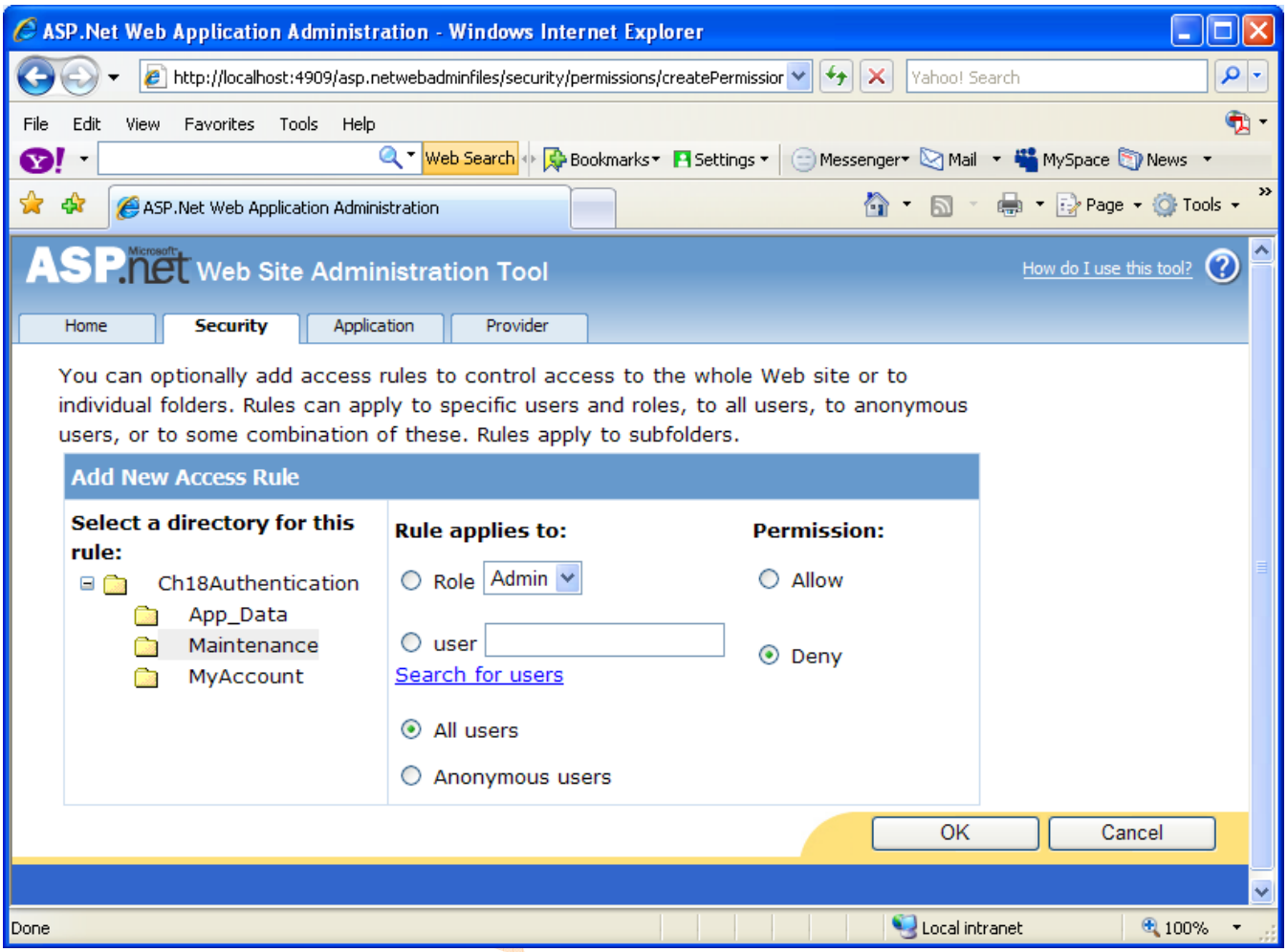
To understand how roles work, let's say you create a role named *Admin* for all employees that will be administrators for the web site, and you assign this role to multiple users. Later, if you want to give all users in the *Admin* role additional permissions, you don't have to give the permissions to each user. Instead, you just need to give the additional permissions to the *Admin* role and all users with that role will get the new permissions.

You can use WSAT to create and manage roles. To do that, you need to click on the *Security* tab to enable roles. Then, you can click on the *Create or Manage Roles* link to display a page like that:



Creating and managing access rules

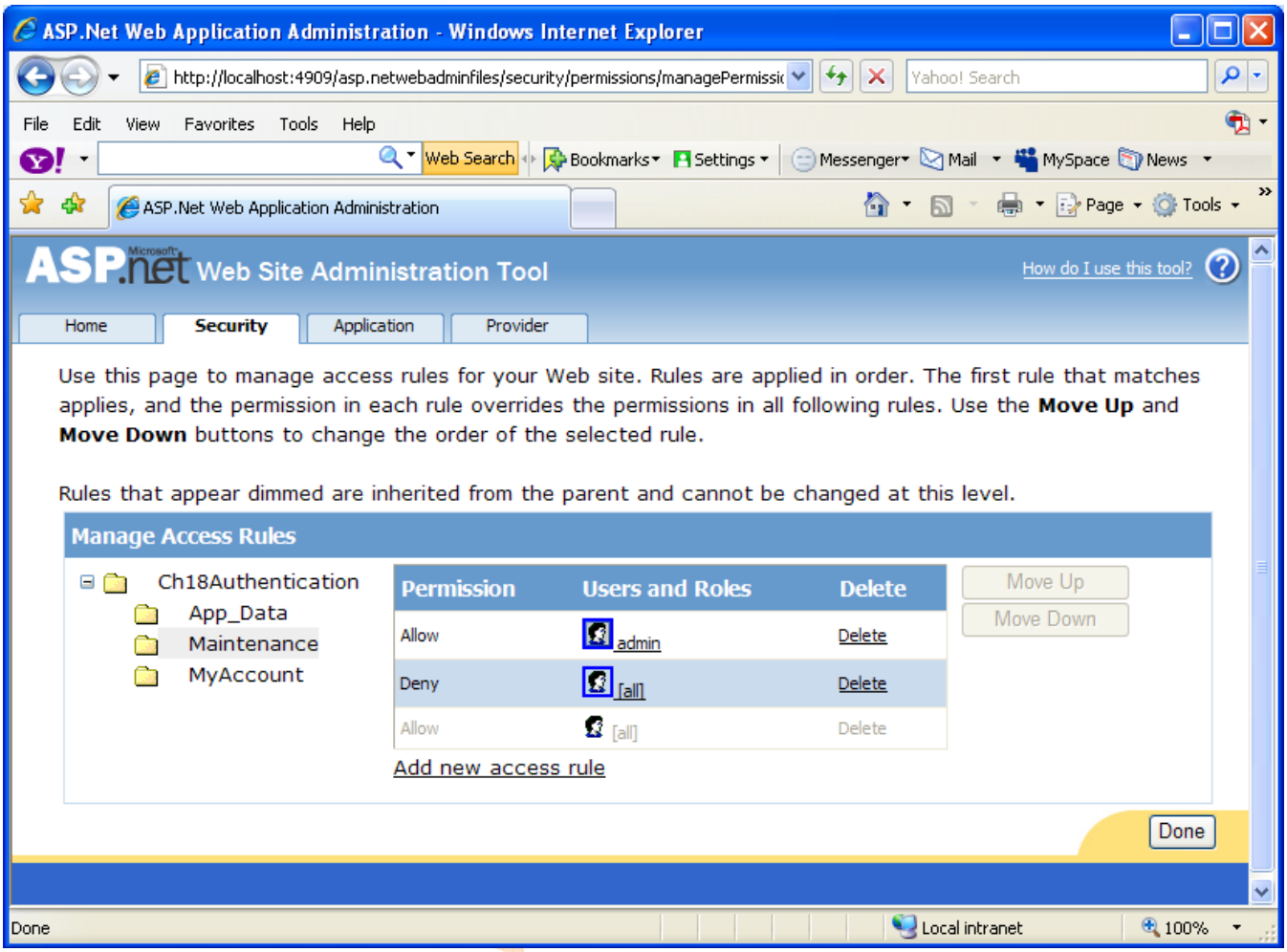
The page, presented below, shows how to create an access rules that restrict access to all or part of a web page application. If you want to apply an access rule to the entire web page application, you can select the root directory for the web application and apply the rule. Then, this rule will apply to all subfolders.



However, it's more common to allow all users including anonymous users to be able to access the pages in the root directory. That way, all users can view your home page and any other pages that you want to make available to the general public. Then, you can restrict access to the pages in your application that are stored in subfolders.

The next page shows how to manage the access rules for a folder. To do that, you can select the folder to display all of the access rules for the folder. Then, you can move rules up or down, which is important since they're applied in the order in which they're displayed. Or, you can delete any rules that you no longer want to apply.

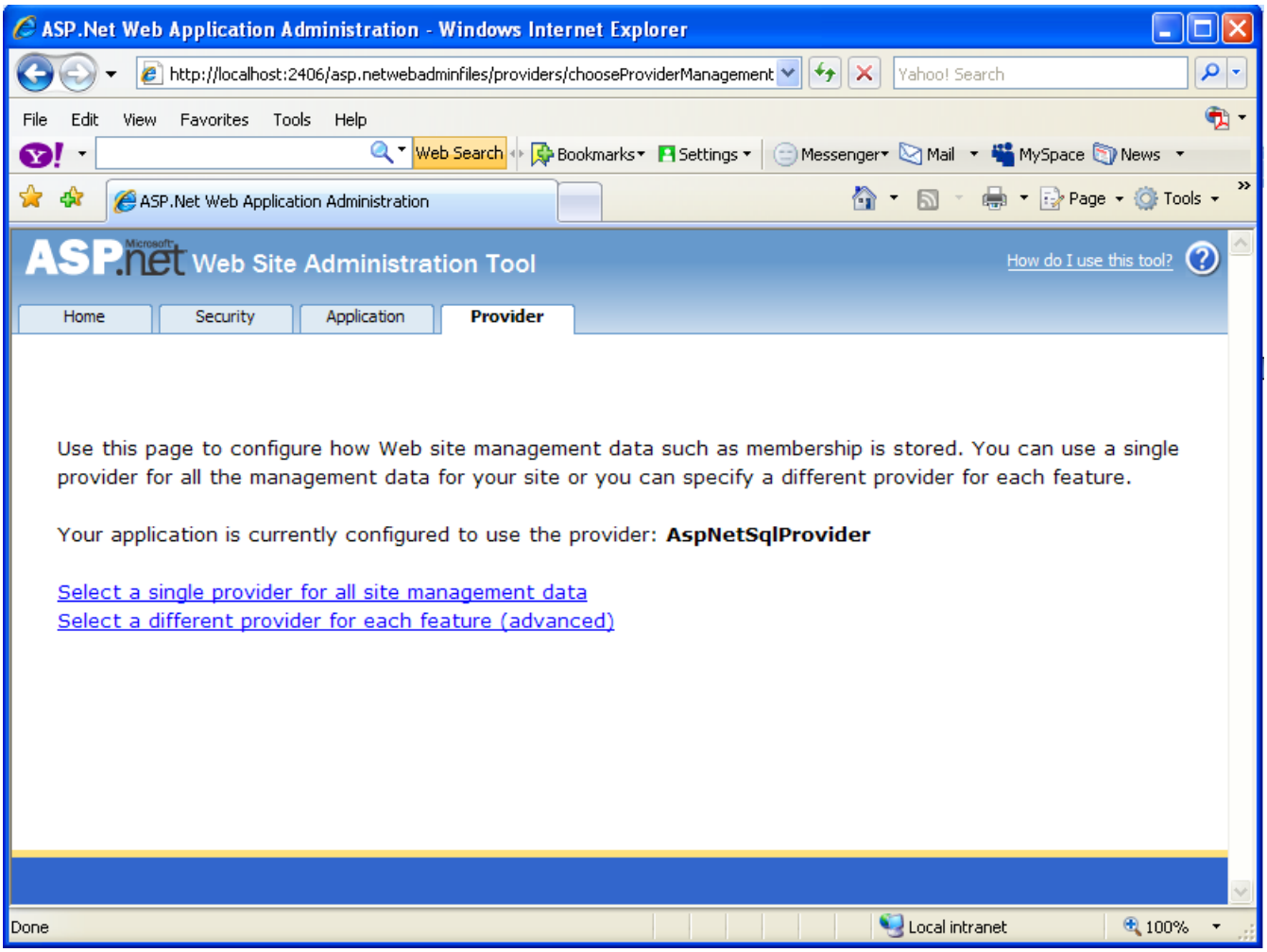
On that page, you can see the rules that restrict access to the *Maintenance* folder. Here, the bottom rule (which is dimmed) is the rule that's automatically applied to all pages of a web site if you don't create new rules. This rule allows access to all users including anonymous users. Then, the middle rule overrides the bottom rule and denies access to all users including authenticated users. However, the top rule overrides the middle rule and allows access to users in the *Admin* role. As a result, only authenticated users in the *Admin* role are able to access the page in the *Maintenance* folder.



Modifying the membership and role provider

When you add roles and users, a class known as a ***data provider*** contains the code that reads and writes the data for the users and roles. A data provider that works with membership data is often called a ***membership provider***, and a data provider that works with roles is often called a ***role provider***. By default, a data provider named *AspNetSqlProvider* is used to store both membership and role data in a SQL Server Express database named *AspNetDb.mdf* that's stored in the *App_Data* folder of your web site. This database is created for you automatically, and it usually works the way you want, at least for prototyping.

However, the data provider architecture lets you use different data providers if you want to. If, for example, you want to use one data provider for the membership data and another for role data, you can use the *Provider* tab shown here



to select separate providers. The only provider for membership data is *AspNetSqlMembershipProvider*. Two providers are available for role data: *AspNetSqlRoleProvider* and *AspNetWindowsTokenRoleProvider*. If you use the Windows token provider, the role information will be based on Windows accounts. Because of that, you won't create roles and users as described in the previous parts of the lecture. In addition, you won't use forms-based authentication. Instead, you'll need to use Windows-based authentication.

If the data providers that ship with ASP.NET 2.0 aren't adequate for your application, you may need to write a custom data provider. If, for example, you need to store membership data in an Oracle database, a MySQL database, a MySQL database, or even on a midrange or mainframe computer, you can write a custom membership provider to do that. Or, if you need to work with existing membership data that's stored in a SQL Server database, you can write a custom provider to work with that data.

To write a membership provider, you need to code a class that implements the abstract *MembershipProvider* class. To write a role provider, you need to code a class that implements the abstract *RoleProvider* class. After you implemented all of the necessary properties and methods of these classes, you can edit the *machine.config* or *web.config* file to add the providers to the list for all applications or the list for one application. Then, you can use the *Provider* tab to select the data providers for memberships and roles. Once you do that, the rest of the authentication features should work as described in that lecture.

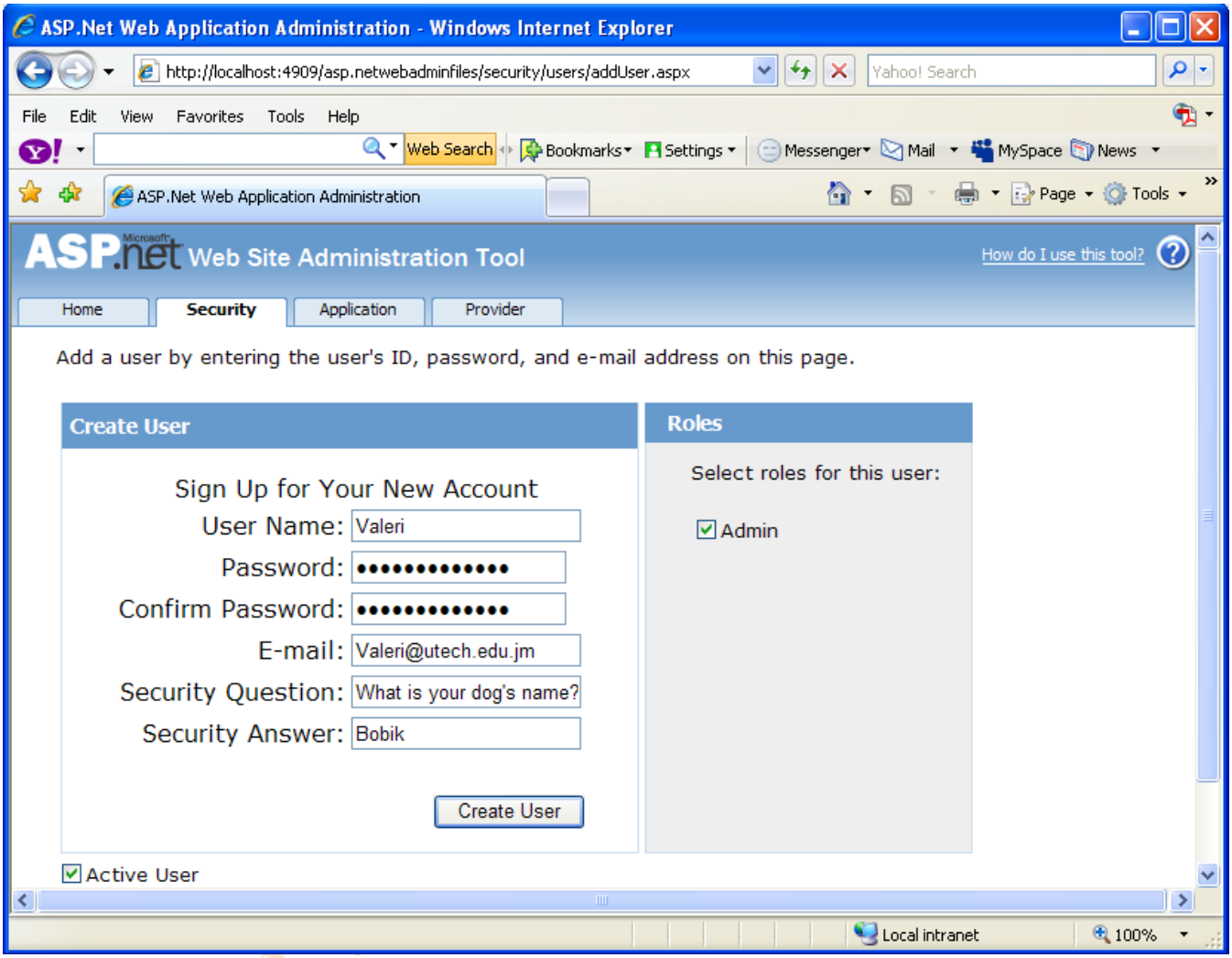
Elements in the web.config file that relax the default password policy

```
<membership defaultProvider="AspNetSqlMembershipProviderRelaxed">
  <providers>
    <add name="AspNetSqlMembershipProviderRelaxed"
        type="System.Web.Security.SqlMembershipProvider,
              System.Web, Version=2.0.0.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a"
        ConnectionStringName="LocalSqlServer"
        enablePasswordRetrieval="false"
        enablePasswordReset="true"
        requiresQuestionAndAnswer="true"
        applicationName="/"
        requiresUniqueEmail="false"
        passwordFormat="Hashed"
        maxInvalidAttemptWindow="5"
        passwordAttemtsWindow="10"
        passwordStrengthRegularExpression=""
        minRequiredPasswordLength="6"
        minRequiredNonalphanumericCharachters="0" />
  </providers>
</membership>
```

This figure also shows how you can modify the attributes of a data provider change the way that the data provider behaves. In particular, it shows how to relax the strict password requirements for an application. To do that, you can copy the *Membership* element from the *machine.config* file (which applies to all web applications) to the *web.config* file (which applies to your application). Then, you can edit the *Name* attribute of the *Add* element to create a unique name for the modified provider, and you can add and set the *minRequiredPasswordLength* and *minRequiredNonalphanumericCharachters* attributes. In that case, these attributes are set so each password requires 6 characters and zero special characters. Last, you can add the *defaultProvider* attribute to the *Membership* element so your application uses this provider.

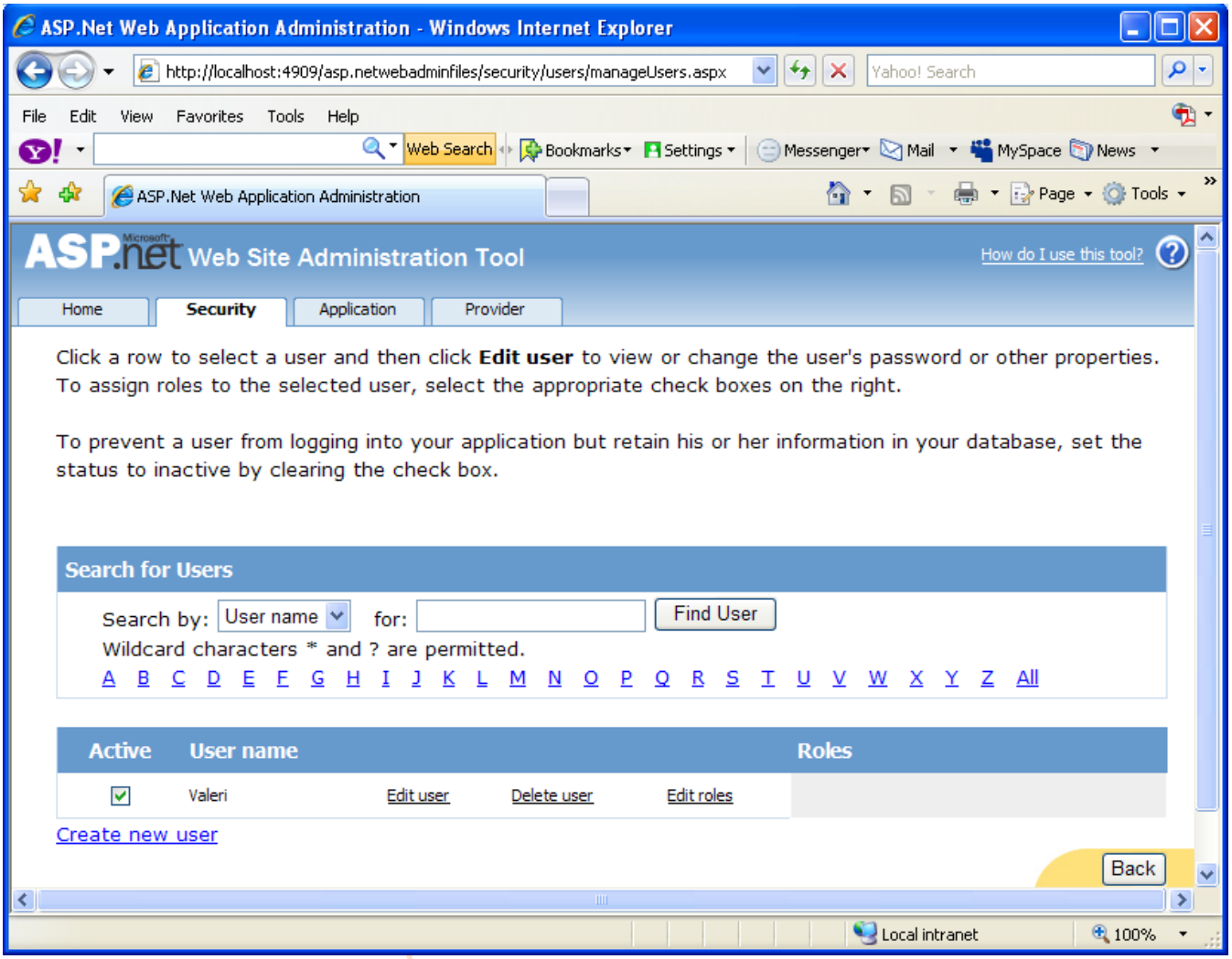
Creating and managing users

This page displayed how to use the WSAT to create a user:



The default password policy for an ASP.NET 2.0 application requires that you entered at least seven characters with one of them being a non-alphanumeric character. If that's too strict for your web site, though, you can relax this policy by adding two attributes to the membership provider, which you'll learn in later lectures.

Once you've created one or more users, you can use the second screen displayed below. To edit a user's e-mail address, you can click on the *Edit User* link. (However, the link doesn't let you edit the username, password, or security question and password.) To change the roles assigned to a user, you can click on the *Edit Roles* link for the user and work with the boxes that are displayed in the *Roles* column on the table. To prevent a user from logging into your application but retain his or her information in your database, you can set the status to inactive by clearing the *Active* check box to the left of the username.



Web Site Administration Tool Overview

The Web Site Administration Tool (WSAT) lets you view and manage the web site configuration through a simple web interface. To access the WSAT, on the website menu, click ASP.Net Configuration.

Web site configuration settings are stored in an XML file named *web.config*, which is located in the root folder of the web site. The WSAT lets you change your site configuration without having to manually edit the *web.config* file. The first time that you use the WSAT to administer a specific web site, if no *web.config* file exists, the WSAT creates one. By default, the WSAT also creates a database in the *App_Data* folder of the web site to store application services data, such as membership and roles information. For most settings, changes that are made in the WSAT take effect immediately and are reflected in the *web.config* file.

Inherited Settings

Default settings for a web site are automatically inherited from any configuration files that exist for the computer or for the Web server as a whole. For example, the web server might have default settings that apply to all sites on that server. Using the WSAT, you can create and modify settings for your specific

web site that are not inherited, and you can override the inherited settings as allowed by the site-wide settings. If the setting has been inherited and cannot be overridden, it appears dimmed, to indicate that it is disabled, in the WSAT.

Requirements

The WSAT is included with the Microsoft Visual Web Developer Web development tool. In order to use the WSAT to administer a web site, the user credentials for the user account under which you are running Visual Web Developer must have *Read* and *Write* permissions to the *web.config* file and the *App_Data* folder of the application being administered. If you cannot manage the configuration for the Web site using the WSAT, contact the system administrator.

Features

The WSAT features a tabbed interface that groups related configuration settings within each tab. The tabs and the configuration settings that the tabs manage are described in the following sections.

***Security* Tab**

Use the *Security* tab of the WSAT to manage rules for securing specific resources in the Web application. ASP.NET uses a security system that lets you restrict access to specific user accounts or the roles to which the user accounts belong. With the *Security* tab, you manage user accounts, roles, and access rules for the web site. Before using the *Security* tab for the first time, use the *Security Setup Wizard* to configure basic security settings for the web site.

ASP.NET security is based on the concepts of user accounts, roles, and access rules and lets you restrict access to your web application resources to only the user accounts that you specify. Security settings are established using a combination of configuration settings and data stored in a database (or other data store). User accounts and roles that you create are stored in the database and access rules are stored in the *Web.config* file.

You can configure your application to use the following types of security, which depend on how the Web site will be used:

- *Forms-based authentication (From the Internet)*
  Forms-based authentication is used for Web sites that are made available to the Internet. Forms-based authentication uses the ASP.NET membership system to manage individual user accounts and groups (roles). User account information is stored in a local database or in a Microsoft SQL Server database. You can use the ASP.NET logon controls to create a logon page where users can enter their credentials.
- *Integrated Microsoft Windows authentication (From a local network)*
  Windows authentication interacts with Windows security, using the logon credentials that users provide when they log on to Windows. Therefore, Windows authentication is suited for intranet scenarios, where users have logged on to a Windows-based network. You do not have to create a logon page, because users are automatically logged on to your application with their Windows credentials.

Use the Users section of the *Security* tab to complete the following tasks:

- Create, edit, and delete registered user accounts for the Web site.
- View a list of all registered user accounts for the Web site.
- Change the authentication method that is used by the Web site.

Note:

You can create and manage user accounts, if you chose the *From the Internet* option for your authentication type (if you are using forms-based authentication). If you chose the *From a local network* option as your authentication type (if you using integrated Windows authentication), you cannot manage individual user accounts. If you change the authentication type, any user information that you have created will be lost. Additionally, access rules might no longer work in the way that you configured them. Generally, you should select an authentication type only when you first configure the web site.

Use the *Roles* section of the *Security* tab to group user accounts, which makes it easier to assign permissions (authorization).

Use the *Access Rules* section of the *Security* tab to allow or deny access for specified pages to specific user accounts or to all user accounts that belong in a specified role. Typically, you use an access rule to restrict pages for some user accounts.

Creating Users

You can create and manage user accounts, if you have set the authentication type to *From the Internet* (forms authentication). To change authentication types, click *Select* authentication type.
To create user accounts *Click Create* user, and then specify the following information.

- ***User Name*** Enter the name for the user account to create.
- ***Password*** Enter the password for User Name. Passwords are case sensitive.
- ***Confirm Password*** Re-enter Password.
- ***E-mail*** Enter the e-mail address for User Name. The Web Site Administration Tool does not confirm whether the address that you enter is a valid e-mail address, but it does validate that the e-mail address conforms to the correct format for e-mail addresses.
- ***Security Question*** Enter a question to ask the user when they need to reset or recover their password.
- ***Security Answer*** Enter the answer to Security Question.
- ***Active User*** Select this option to enable this user account as an active (current) user of the site. If you do not select this option, the user information is stored in the database, but the user cannot log on to the Web site.
- ***Roles*** Select the roles for User Name. You create roles separately.

Creating Roles

To create roles

1. On the *Security* tab, click *Enable* roles.
2. Click *Create* or *Manage* roles.
3. In the *New* role name box, enter a name for the role to create, such as ***Administrator***, ***Member***, or ***Guest***, and then click *Add Role*.

To add user accounts to roles

1. On the *Security* tab, click *Manage Users*, and then click *Edit User*.
2. Under *Roles*, select the roles for the user account.

To create access rules

1. On the *Security* tab, click *Create* access rules.
2. Specify the following options:
   - Select a directory for this rule

     You can choose to create a rule that applies to the whole site or to only a specific subdirectory. In the directory structure display for the web site, select the directory to which the rule applies.
3. Under *Rule* applies to, specify how to apply the rule.
   - *Role*

     Select *Role*, and then in the list, select the name of the role to which the access rule applies.
   - *User*

     Select *User*, and then enter the name of the user account to which the access rule applies. If you are using ASP.NET membership (Web site security is set to *From the internet*), you can also use the Search for users feature.
   - *All users*

     Select this option to apply the rule to all visitors to the web site.

     Note:

     Be careful when you create a rule with the *All users* option. Because rules are applied in order, you can unintentionally create a rule that prevents all users from accessing a folder.
   - *Anonymous users*

     Select this option to apply this rule to anonymous (non-registered) user accounts only.
4. Typically, you choose the *Anonymous users* option to restrict (deny) access for users who are not logged on.
   - *Permission*

     Select *Allow* to give access to the specified directory for the specified user account or role.

     Select *Deny* to not allow access to the specified directory for the specified user account or role.

     For example, to prevent users who are not logged on (anonymous) from viewing pages in a folder, click the folder, select *Anonymous users*, and then select *Deny*.
5. Sometimes, you might have to create multiple rules for the same folder in order to establish the correct permissions. For example, you might create a rule that denies access to anonymous user accounts and a second rule that denies access to user accounts in the role of *Guest*. That way, only users who are logged on and in another group can access the folder.

Behind the Scenes

The Web Site Administration Tool manages security information in the following two places:

- The *Web.config* file at the root of the web site.
- The site provider database that is used to store user and group information.

Web.config Settings

The *Web.config* settings that are managed through the *Security* tab are the *<authorization>, <roleManager>,* and *<authentication>* sections.

The following code example is the *Web.config* file that is created by the Web Site Administration Tool within a restricted subdirectory of the web site. Access to the restricted subdirectory is allowed for administrators and denied for anonymous users.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authorization>
      <allow roles="administrators" />
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Database

When you use the default data provider, the WSAT creates entries in the default ASP.NET database. By default, the WSAT creates a database in the *App_Data* folder of the web site. However, using the *Provider* tab, you can specify that application information for user accounts and roles be kept in another database (for example, retrieving role information from the Windows user database). For detailed information, see Web Site Administration Tool Provider Tab below.
When ASP.NET automatically creates a database in the *App_Data* folder of the web site it assigns a name *ASPNETDB.MDF.* Sometimes due several reasons it doesn't creates it. Or you need to move your web application to another location an you don't want inherit security information about users, their roles, rules, etc. to new destination. In this case you need to in new location create a database, which support all security issues, described above, yourself.

The ASP.NET SQL Server Registration tool is used to create a Microsoft SQL Server database for use by the SQL Server providers in ASP.NET, or to add or remove options from an existing database. The ***Aspnet_regsql.exe*** file is located in the *[drive:]\%windir%\Microsoft.NET\Framework\version* folder on your web server.
You can run *Aspnet_regsql.exe* without any command-line arguments to run a wizard that will walk you through specifying connection information for your SQL Server installation, and installing or removing the database elements for the membership, role management, profile, Web Parts personalization, and health monitoring features. (Setting session state and SQL cache dependency are not covered by the wizard.) You can also run *Aspnet_regsql.exe* as a command-line tool to specify database elements for individual features to add or remove, using the options listed in the technical documentation: *http://msdn.microsoft.com/en-us/library/ms229862(VS.80).aspx*.

The ASP.NET creates a database usually in system area on your machine for storing SQL Sever files in *C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL*Data folder. Next, you need attach the file, created by the *Aspnet_regsql.exe* utility using features of the SQL Management Studio Tool. After that you need update a *Web.config* file adding following elements:

```
<connectionStrings>
  <remove name="LocalSqlServer" />
  <add name="LocalSqlServer"
    connectionString="Data Source=VPOUGATCHEV;
                Initial Catalog=HalloweenASPNETDB;
                Integrated Security=True;
                MultipleActiveResultSets=False;
                Packet Size=4096;
                Application Name=&quot;
                Microsoft SQL Server Management Studio Express&quot;"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

When you install ASP.NET, the *machine.config* file for your server includes configuration elements that specify SQL Server providers for each of the ASP.NET features that rely on a provider. These providers are configured, by default, to connect to a local user instance of SQL Server Express 2005. If you change the default connection string used by the providers, then before you can use any of the ASP.NET features configured in the machine configuration, you must install the SQL Server database and the database elements for your chosen feature using *Aspnet_regsql.exe*. If the database that you specify with *Aspnet_regsql.exe* does not already exist (*ASPNETDB* will be the default database if one is not specified on the command line), then the current user must have rights to create databases in SQL Server as well as to create schema elements within a database.
Each SQL Server provider can use the same SQL Server database to store the data for that particular feature. Each feature can be used individually, or in conjunction with other features. For example, you could use role management by itself or in conjunction with the user information managed by membership.

Application **Tab**

Use the *Application* tab of the Web Site Administration Tool (Web Site Administration Tool) to manage commonly used settings related to the web application and site.

Use the *Application* tab to manage the following:

- ***Application settings name/value pairs.***
  Application settings are name/value pairs that represent configurable values text in the Web application. Use application settings to store custom application configuration information, such as file paths, XML Web service URLs, commonly used text, or any information that you want to maintain in a central location and be able to change easily.
  Important Note:
  Application settings are stored in the configuration file as plain text. Therefore, you must take appropriate security measures when you use application settings. Do not store sensitive information, such as user names, passwords, or database connection strings in application settings.

- ***Simple Mail Transfer Protocol (SMTP) settings.***
  If the Web site requires the ability to send e-mail (for example, to send users a password), you must specify the SMTP server that your site uses.
- ***Application status.***
  You can take your application offline (shut it down) to perform maintenance or to bring a new Microsoft SQL Server Express edition database online.
- ***Debugging and tracing.***
  Debugging and tracing let you diagnose and fix problems with the Web application, and are primarily intended to be used during development. Enabling debugging causes the pages in the web site to be compiled with information that the .NET Framework or the Microsoft Visual Studio debugger can use to step through your code. Enabling tracing causes pages to generate information about individual web requests, the HTTP headers that are sent with the requests, the state of controls on the page, and other details about page processing.

Configuring Application Settings

When you create an application setting, you specify a name and value for the setting to be created. This lets you access the value of the setting in your application by using the ***AppSettings*** property of the ***ConfigurationSettings*** class. For example, if you define an application setting named *AppName* with a value that represents the name of your application, you might use it to set the heading of a page as shown in the following code example.

[Visual Basic]

```
labelPageHeading.Text = ConfigurationManager.AppSettings("AppName")
```

[C#]

```
labelPageHeading.Text = ConfigurationManager.AppSettings["AppName"];
```

Taking Applications Offline and Online

If you want to do maintenance on the web site, you can take it offline. This shuts the process down that is running the web site so that the site no longer serves pages. You can then edit pages or other files without the possibility that a page will be requested while you are in the middle of your edits, potentially resulting in an error. Taking an application offline is also useful, if you are working with SQL Server Standard edition and want to swap or add an *.mdf* database file to the application. If the application is running, it cannot attach to new *.mdf* files.

When you have finished editing files or adding or changing *.mdf* files, you can put the application back online.

Configuring Debugging and Tracing

You must have Administrative credentials on the computer where the web application runs to change its debugging and tracing settings.

To manage the debugging and tracing features, click *Configure debugging and tracing*, and then set the following options:

- *Enable Debugging*
  Select this option to enable debugging for all pages in the web site.
- *Capture tracing information*

Select this option to enable tracing for all pages in the web site. If this check box is selected, the remaining options on the page are enabled.

If you have set the option to capture tracing information, you can make the following choices:

- *Display tracing information on individual pages*
  Select this option to append tracing information to the bottom of web pages on your site. If you do not display tracing information about individual pages, the information is still captured. To view trace information, you can request the *Trace.axd* page for the web site, which acts as a tracing information viewer.
- *Local requests only*
  Select this option to view trace information only for request from the host Web server (localhost).
- *All requests*
  Select this option to view trace information from any computer.

Under *Select* the sort order for trace results, specify how you want to display trace information, as follows:

- *By time*
  Select this option to display trace information in the order that it is captured.
- *By category*
  Select this option to display trace information alphabetically within user-defined categories.
- *Number of trace requests to cache*
  Enter the number of trace requests to store on the server. You can view cached information by using the trace viewer *(http://*server/application*/trace.axd).* The default is 10. If you do not select the Most recent trace results option, and the limit is reached, trace is automatically disabled.

Under *Select* which trace results to cache, specify how you want to display trace output, as follows:

- Most recent trace results
  Select this option to display the most recent trace output and to discard older trace data beyond the limits that are indicated by *Number* of trace requests to cache.
- *Oldest trace results*
  Select this option to display trace data for requests until *Number* of trace requests to cache is reached.

Note:

When you are finished troubleshooting the web site, disable debugging and tracing to get maximum performance from your web application.

Security Note:

When tracing is enabled for a page, trace information might appear on any browser that makes a request for the page from the server. Because tracing displays sensitive information, such as the values of server variables, it can represent a security threat. Make sure that you disable page tracing for the page before publishing the Web application to a production server.

To configure a custom error page for the web site, click *Define* default error page and then specify the following:

- *Use the default error message*
  Select this option to use the default error page.
- *Specify a URL to use as the default error page*

Select this option to use a custom error page, and then choose the page to use.

Behind the Scenes

The *Application* tab provides a simple Web interface for managing configuration settings that are stored in the *Web.config* file for your application.

Application Settings

The settings that are managed through the *Application Settings* feature of the *Application* tab exist in the *<appSettings>* section of the *Web.config* file for the web application. This is a predefined configuration section provided by the .NET Framework. The highlighted lines in the following code are an example configuration file that is generated after you use the Web Site Administration Tool to create an application setting named *ApplicationName*.

```
<configuration>
  <appSettings>
    <add key="ApplicationName" value="MyApplication" />
  </appSettings>
</configuration>
```

Taking Applications Offline and Online

The offline setting exists within the *<httpRuntime>* section of the *Web.config* file for the Web site. The highlighted lines in the following code are generated after you use the Web Site Administration Tool to take an application offline.

```
<configuration>
  <system.Web>
    <httpRuntime enable="False" />
  </system.Web>
</configuration>
```

As long as the ***httpRuntime*** setting is disabled, ASP.NET does not create an *AppDomain* object for your application upon receiving a request. In effect, the Web application cannot be started.

Debugging and Tracing

The settings managed through the Debugging and Tracing feature of the *Application* tab exist within the *<trace>*, *<compilation>*, and *<customErrors>* sections of the *Web.config* file. The following code is an example configuration file that is generated after you use the Web Site Administration Tool to enable both tracing and debugging and to establish a default custom error page.

```
<configuration>
  <system.Web>
    <customErrors defaultRedirect="~/myErrorPage.aspx" />
    <trace enabled="True" pageOutput="True" localOnly="True"
        traceMode="SortByCategory" requestLimit="10" mostRecent="True" />
    <compilation debug="True" />
  </system.Web>
```

```
</configuration>
```

In this example, both debugging and trace are enabled, tracing is displayed on pages that are requested from the Web server only, trace results are sorted by category, and the 10 most recent trace results are cached for display. Additionally, the default error page is *myErrorPage.aspx*.

Provider **Tab**

Use the *Provider* tab of the Web Site Administration Tool to manage how ASP.NET stores the data for application features, such as user accounts, roles, and other settings.

ASP.NET uses provider classes to manage data storage for various features, such as membership and role management. A provider class is a component that exposes specific ASP.NET functionality, such as managing user accounts in membership. Each of these application features requires data storage. Although each provider class performs the same application features, each provider class can store data in different ways. For example, different provider classes for ASP.NET membership might store user account information in the following different ways:

- In a Microsoft SQL Server database.
- In *Microsoft Windows Active Directory* directory service.
- In a custom user database.

Each of these provider classes still performs the same core set of tasks for managing user accounts and you interact with the provider classes in your application in the same way.

Configuring Providers in the Web Site Administration Tool

The Web Site Administration Tool lets you use different provider classes for application features, such as membership and roles. ASP.NET is installed with several provider classes for each type of application feature. For example, ASP.NET has a provider class for membership that stores information in a SQL Server database ***(AspNetSqlMembershipProvider)***, and another that stores membership information in Active Directory.

Managing Provider Settings

You can use the Web Site Administration Tool to change and test providers for the Web site. Your application can use a single provider for application features or it can use several different providers.

By default, the Web Site Administration Tool uses the ***AspNetSqlProvider*** provider for all application features.

Use the Web Site Administration Tool to manage providers for the web site in the following ways:

- Change from the default ***AspNetSqlRoleProvider*** provider to the alternate ***AspNetWindowsTokenRoleProvider*** provider, if you would prefer to use local *Windows* groups for role authorization.
- Specify whether to use the same provider for all application features or to use a different provider for each application feature. Generally, you select individual providers, only if you want precise control over where the information is stored or if you have to use a different provider for just one feature, such as roles.

Selecting a Single Provider vs. Different Providers

You can configure the Web site to use the same provider for all application features. In this case, all data for membership and roles and so on, is stored in a single data store (typically a single database). Alternatively, you can select a different provider for each application feature.
In the Web Site Administration Tool, on the *Provider* tab, your first choice is whether you want to use the same provider for all features or to select a provider individually for each application feature. You can choose the following options:

- To specify a single provider for all application features, click *Select* a single provider for all site management data.
- To specify providers for specific application features, click *Select* a different provider for each feature (advanced), and then select a provider for each application feature.

Configuring the SQL Server Provider

If you want to use the SQL Server provider to store application feature data in a SQL Server database, you must first configure SQL Server by creating the appropriate database. ASP.NET includes a command-line utility named *aspnet_regsql.exe* that performs this task for you.

The *aspnet_regsql.exe* executable is located in the *WINDOWS\Microsoft.NET\Framework\versionNumber* folder on the Web server. The *aspnet_regsql.exe* utility is used to both create the SQL Server database and add or remove options from an existing database.

You can run the *aspnet_regsql.exe* executable without any command-line arguments to run a wizard that will help you with specifying connection information for SQL Server and installing or removing the database elements for all supported features. You can also run the *aspnet_regsql.exe* executable as a command-line utility to configure database elements for individual features.

To run the *aspnet_regsql.exe* wizard, run the *aspnet_regsql.exe* executable without any command-line arguments, as shown in the following example.

```
[%system root%]\Microsoft.NET\Framework\versionNumber\aspnet_regsql.exe
```

To view online Help for additional options that are available with the *aspnet_regsql.exe* utility, use the ***/?*** option.

Behind the Scenes

The *Provider* tab manages configuration settings that are stored in the *Web.config* file for your application. Specifically, the settings that are managed through the *Provider* tab exist in the *<membership>* and *<roleManager>* sections of your configuration file.

The following code is the *Web.config* file that is generated by the Web Site Administration Tool after specifying that the ***AspNetWindowsTokenRoleProvider*** provider be used for role management.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<configuration>
    <system.web>
        <membership defaultProvider="AspNetSqlMembershipProvider" />
        <roleManager enabled="true" defaultProvider="AspNetWindowsTokenRoleProvider" />
        <authentication mode="Forms" />
    </system.web>
</configuration>
```