Programmer to Programmer ™

Professional ASP.NET Security

Russ Basiura, Richard Conway, Brady Gaster, Daniel Kent, Sitaraman Lakshminarayanan,
Enrico Sabbadin, Doug Seven, Srinivasa Sivakumar

wrox
Programmer to Programmer™

Wrox technical support at: support@wrox.com
Updates and source code at: www.wrox.com
Peer discussion at: p2p.wrox.com

What You Need to Use this Book

The following is the list of recommended system requirements for running the code in this book:

- Windows 2000 Professional or Windows XP Professional with IIS installed
- Visual Studio .NET Professional or higher
- SQL Server 2000 or MSDE.

This book assumes the following knowledge:

- Good working knowledge of ASP.NET and the .NET Framework
- Familiarity with C#
- Some familiarity with the Visual Studio .NET IDE

Summary of Contents

Windows Authentication

One option for authentication in ASP.NET is to hand over responsibility for authentication to Internet Information Services (IIS). IIS passes ASP.NET a Windows user account with each request it sends for processing. If IIS is set up to require users to log in, this account will refer to the user making the request. Because this authentication option involves Windows user accounts, it is known as Windows authentication and is provided by the Windows authentication module.

Why Would We Use Windows Authentication?

There are three main reasons why we would want to use Windows authentication:

- It involves little work on our part
- Integration with IIS security
- Integration with Windows client machines

The first reason is quite simple – using Windows authentication hands over the responsibility for authentication to IIS so we do not have to implement that functionality ourselves. We let IIS authenticate users and then make use of the user information that IIS provides.

The second reason we might want to use Windows authentication is that it means that our web site authentication is fully integrated with IIS security. We control access to files through windows file access permissions and user accounts are managed through the normal tools.

The final, and main, reason for using Windows authentication is that using IIS for authentication means that it is possible to provide 'invisible' authentication when users are logged in to Windows machines. This is ideal in situations such as intranets where we do not want the user to have to enter their username and password when they enter the intranet site – we just pick up the identity that has already been authenticated.

So why would we *not* want to use Windows authentication?

- Tied to Windows users
- Tied to Windows client machines
- Lack of flexibility

The first problem is that Windows authentication relies on the users we are authenticating having valid Windows accounts. We may well not want to grant users of a web site Windows accounts on our web servers.

The second problem is that some of the authentication methods that IIS uses rely on the user having compatible software on their client machine. This limits our ability to use Windows authentication for users who are using non-Microsoft client software.

The final main problem is that Windows authentication does not give us much control over authentication. We hand the responsibility for authentication to IIS so there is not much we can do to change the way that it works.

How Does Windows Authentication Work?

Just like all other Windows processes, IIS must make an association between an authenticated user requesting a page from its virtual directory store and that user's equivalent Windows account. For those requests that IIS treats as "anonymous" requests (where a user has not entered a username and password), the IUSR account reserved as the account for which IIS will execute all server-side processing is used. To Windows, it appears as though IIS is performing a set of tasks on behalf of the IUSR account for the machine on which IIS is installed.

For pages within sites that have been restricted from anonymous users, IIS performs one of the built-in authentication functions to obtain account information from the client. We will discuss these authentication methods in the following sections. IIS hands the account information it has obtained to Windows so that the account that matches the provided data can be attached to the executing processes required to complete a given HTTP request.

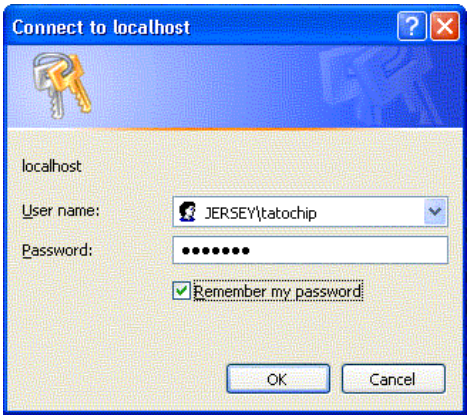
IIS uses one of three possible authentication strategies to authenticate each request it receives:

- Basic Authentication – the username and password are passed as clear text
- Digest Authentication – the username and password are protected with cryptographic techniques
- Integrated Windows Authentication – the identity of a user already logged in to Windows is passed automatically, without the need for a username and password to be entered

We'll discuss these in greater detail now.

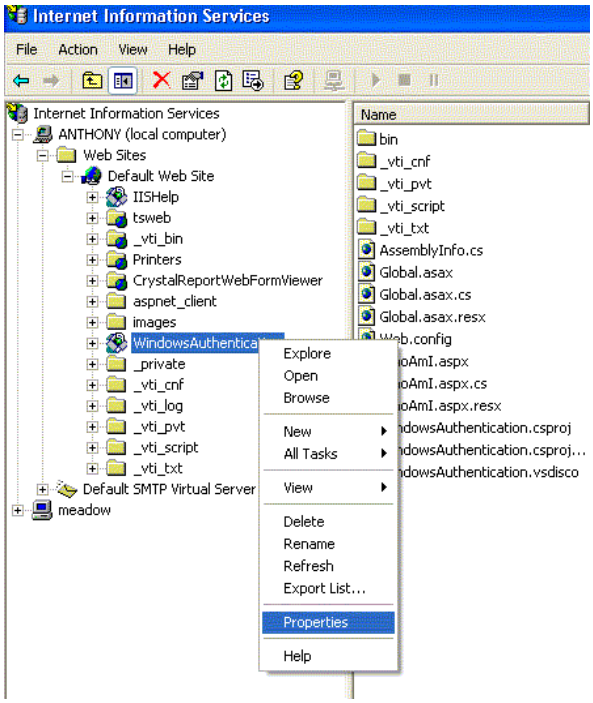
Basic Authentication

Perhaps the most widely supported authentication methodology, **Basic Authentication** serves as a means for Internet application authentication that nearly any web browser supports. During Basic Authentication IIS obtains logon information from an HTTP client via a familiar dialog box that obtains the username and password information from the web client.

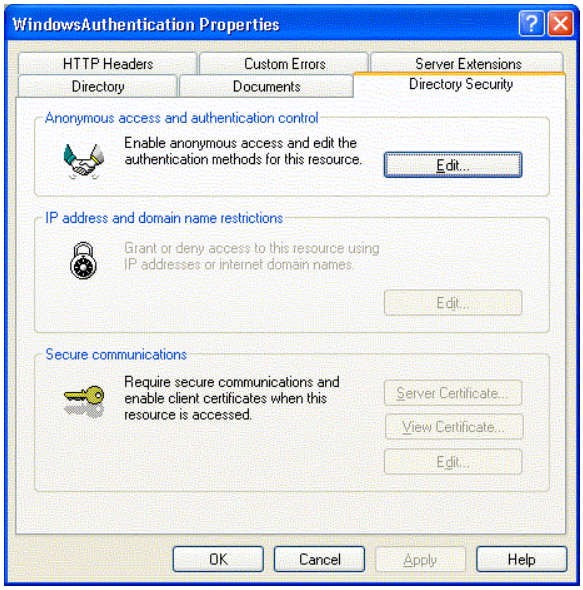


After a user provides this information, the data itself is transmitted to the web server (in this case `localhost`). Once IIS receives the authentication data it is used to attempt to log in to a corresponding Windows account. If an account is located in Windows that matches this data, and the account is allowed access to this particular file or virtual directory, the request is associated with the Windows account and an appropriate response – in most cases, the HTML content rendered by the web server – is returned to the client.

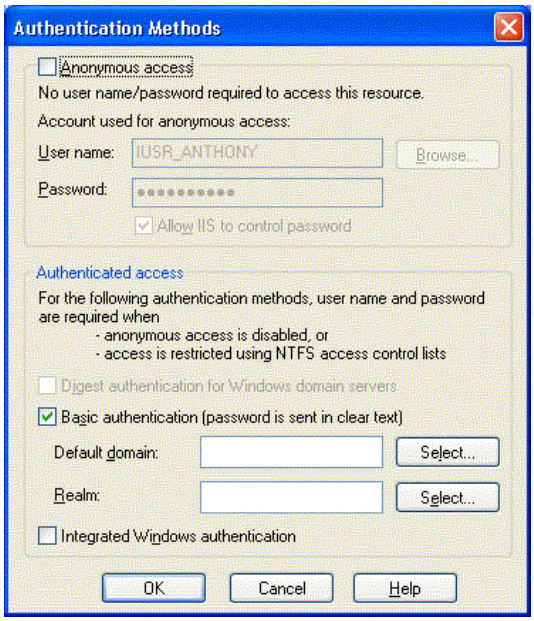
The process of enabling an IIS virtual directory with the Basic Authentication is straightforward, thanks to a simple Windows interface. By using the Microsoft Management Console (MMC) snap-in for Internet Information Services, any virtual directory or entire web site can be set up to authenticate users via Basic Authentication. To do so, locate the virtual directory you wish to secure, right-click on it, and select properties.



Once you select the properties item a property page will appear containing various configuration possibilities for the specified virtual directory. By clicking on the Directory Security tab in the properties page, you can see a series of sections related to the process of securing a specified directory.



The topmost section of this window is responsible for authentication functionality. By clicking on the button labeled Edit in this section, the final dialog appears, displaying a set of choices that can be made pertaining to how this virtual directory will authenticate its visitors.

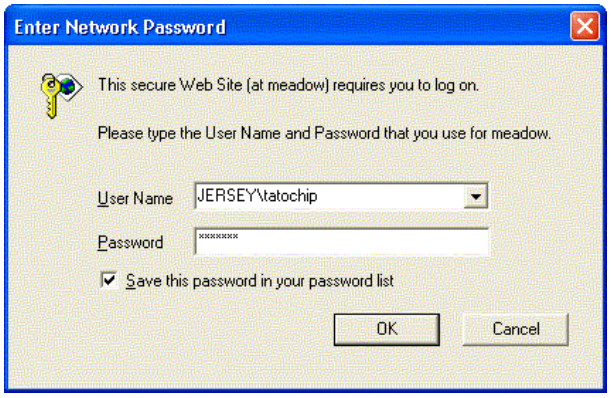


By inspecting the Authentication Methods dialog, you can see that this particular virtual directory has already been set up to authenticate users with Basic Authentication. In addition to this high-level instruction, options exist for specifying a particular Windows domain or realm against which users could be authenticated.

> **It is important to note that Basic Authentication, though virtually universal to most HTTP application platforms, is an insecure method of authentication. Username and password credentials obtained via Basic Authentication are transmitted between the client and server as clear text. The data itself is encoded (not encrypted) into a Base-64 string that intruders can easily hijack and decode. For this reason, Basic Authentication should only be used in conjunction with an HTTP wire encryption tool such as Secure Socket Layers (SSL). In this way, the data that would otherwise be clearly visible to any network sniffing utility will be encrypted using complex algorithms.**

Digest Authentication

Digest authentication, like Basic Authentication, requires the user to provide account information via a dialog that is displayed when a virtual directory is requested from IIS.



Unlike Basic Authentication, however, Digest authentication passes a digest (hence the name of this scheme) of the password, rather than the password itself. This means that the password itself is never sent across the network, preventing it from being stolen.

The process of Digest Authentication works like this:

1. The unauthenticated client requests a restricted resource.

2. The server responds with an HTTP 401 response. This response includes a 'nonce' value. The server ensures that each nonce value is unique.

3. The client uses the nonce, the password, and some other values, to create a hash (remember from Chapter 3 that a hash is an essentially unique value that is generated from the password but cannot be used to obtain the password). This hash value, known as the digest, is sent back to the server along with the plain text username.

4. The server uses the nonce value, its stored password for the username, and the other values to create a hash. It then compares this hash to the one provided by the client. If they match then the passwords match (since the other values used to compute the hash are the same).

Since the nonce value changes with each authentication request, the digest is not very useful to an attacker – the original password cannot be extracted from it and the digest cannot be used for 'replay' attacks where the value is re-sent.

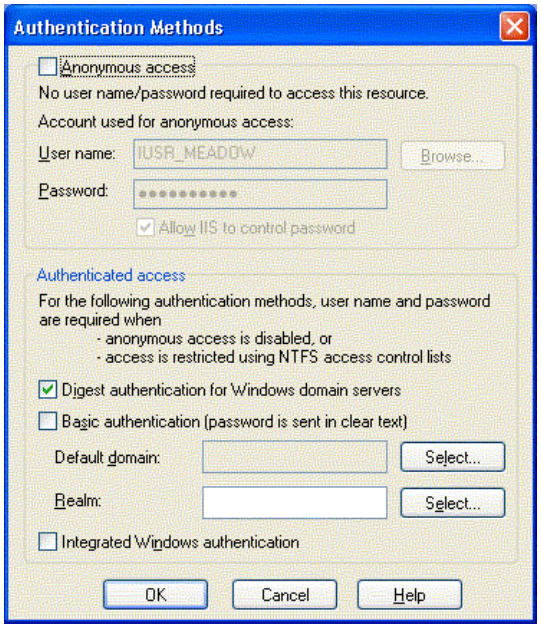
Limitations of Digest Authentication

In theory, Digest authentication is a standard – web servers and web browsers should all be able to use digest authentication to exchange authentication information. Unfortunately, Microsoft interpreted a part of the Digest authentication specification in a slightly different way from other organizations, such as the Apache Foundation (the Apache web server) and the Mozilla project (the Mozilla web browser). This means that, until the problems are solved, Microsoft products that use Digest authentication will not be able to use it in conjunction with non-Microsoft products.

Another limitation of Digest authentication in IIS is that it will only function when the virtual directory being authenticated via Digest is running on or controlled by a Windows Active Directory domain controller.

Configuring Digest Authentication

If the virtual directory you wish to authenticate via Digest authentication resides on a Windows Active Directory domain controller, the option to enable Digest authentication will be enabled in the Authentication Methods dialog of the IIS console in the MMC. The screen capture below displays the appropriate settings should you decide to enable Digest authentication.

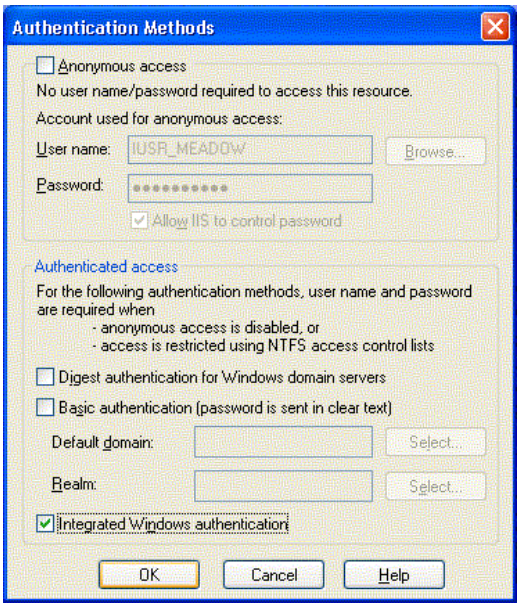


Integrated Windows Authentication

Perhaps the most simply implemented authentication scheme, integrated Windows authentication provides WAN – or LAN – based intranet applications with an authentication practice that is virtually invisible to the client user. Users who have logged into a Windows NT, 2000, or Active Directory domain exchange authentication digests with a domain controller, and a respective set of credentials are provided to the client workstation. When an HTTP request originates from this same workstation, those credentials are transmitted with it to IIS. This tight integration between applications served with IIS and the underlying Windows security framework is a major reason for the widespread adoption of and support for integrated Windows authentication.

Configuring Integrated Windows Authentication

Like the previous authentication options, integrated Windows authentication can be enabled in the Authentication Methods dialog of the IIS administration tool. The following image illustrates the appropriate settings necessary to enable a virtual directory with integrated Windows authentication.



During integrated Windows authentication IIS sends two HTTP headers – Negotiate and NTLM – to the requesting client. If the client is Internet Explorer 2.0 or higher (integrated Windows authentication is not supported in non-Internet Explorer clients) the client is capable of handling the Negotiate header, IIS is notified, and a Kerberos authentication ticket is generated.

A Brief Introduction to Kerberos

Kerberos authentication tickets and more importantly the support for the Kerberos authentication protocol are new to the Windows architecture with the release of Windows 2000. The Kerberos protocol specifies a shared-secret authentication paradigm, thus dictating that both the server and the client know the password for a given account that will be used during the authentication process. The Kerberos protocol itself provides specific methods of storage, retrieval, and encryption. Kerberos provides a much quicker authentication option than historic Windows challenge-response (NTLM) models, which are used during integrated Windows authentication processes should the client application fail to understand the Negotiate HTTP header.

Kerberos is supported in Windows 2000 networks and Internet Explorer browsers version 5.0 and higher.

The ASP.NET Windows Authentication API

We have seen how we can set up IIS to use one of the three authentication methods to identify users. When any of these methods is activated, the identity of the user making a request will be passed to ASP.NET along with the request (if the user is anonymous then the account that is configured for anonymous access will be passed).

ASP.NET provides a set of classes that allow us to make use of the authentication that IIS performs. These classes are found in the `System.Security.Principal` and `System.Web.Security` namespaces.

In this section, we will take a look at each of these classes and what part they play.

The WindowsAuthenticationModule Class

`WindowsAuthenticationModule` is an HTTP Module that deals with receiving the authentication information that IIS sends along with the request and populating `Context.User` with an object that represents the authenticated user.

When the Windows authentication module is activated in the `web.config` file (see configuring ASP.NET for Windows authentication, below), it creates a `WindowsPrincipal` object, including a `WindowsIdentity` object, with each request. We will look at these two classes in more detail in the next two sections.

`WindowsAuthenticationModule` performs this task by handling the `HttpApplication.AuthenticateRequest` event.

There is only really one feature of the `WindowsAuthenticationModule` class that we ever need to program against. This is the `Authenticate` event that it exposes. This event is raised during the authentication process and can be used to manipulate what goes on. We will be looking at this later in this chapter, when we discuss how we can customize Windows authentication.

The WindowsPrincipal Class

The `WindowsPrincipal` class is an implementation of the `IPrincipal` interface that we saw in the last chapter. As we discussed in that chapter, the principal is the security context for the user that allows us to decide what the user may and may not do.

`WindowsPrincipal` does not define any completely new members in addition to those required by the `IPrincipal` interface. As required, it provides access to the `IIdentity` object associated with the principal through the `Identity` property. It also implements the `IsInRole` method.

`WindowsPrincipal` implements three different overloads of `IsInRole` that all check whether the user is in a specified Windows user group. The required `IsInRole(string)` overload is implemented so that it accepts the name of the user group to be checked. `IsInRole(int)` expects an integer Role Identifier (RID) that refers to a user group. Finally, an overload is provided that expects a member of the `WindowsBuiltInRole` enumeration (this can be found in `the System.Security.Principal` namespace along with `WindowsPrincipal`). We will look at how we can use these overloads of `IsInRole` later in the chapter.

The WindowsIdentity Class

`WindowsIdentity` is an implementation of the `IIdentity` interface that we looked at in the last chapter.

In addition to the intrinsic properties exposed by the `IIdentity` interface, the `WindowsIdentity` class offers more properties that can be inspected at run time to gain additional information. By extending the functionality found in the `IIdentity` interface, the `WindowsIdentity` objects present a more Windows-focused format.

`WindowsIdentity` provides a number of properties that allow us to check certain features of the identity:

- `IsAnonymous` allows us to determine whether the user is anonymous (has not identified themselves to Windows).
- `IsGuest` allows us to check whether the user is using a Guest account. (These accounts are designed for public access and do not confer very many privileges.)
- `IsSystem` allows us to determine whether the user is acting as part of the operating system, with a highly privileged system account.

`WindowsIdentity` also provides some static methods that create `WindowsIdentity` instances:

- `GetAnonymous` creates a `WindowsIdentity` that represents an anonymous user.
- `GetCurrent` creates a `WindowsIdentity` that represents the identity tied to the current security context (that is, the user whose identity the current code is running under).

The `Token` property of `WindowsIdentity` allows us to access to the authentication token for the identity.

Implementing Windows Authentication

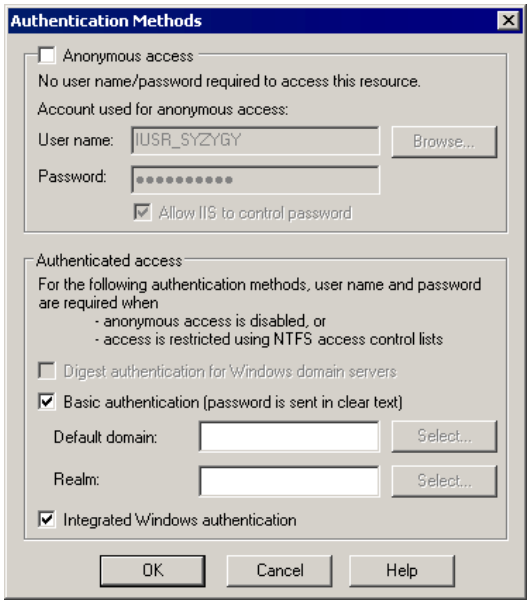
In order to use Windows authentication in an ASP.NET application and have access to the details of the user currently authenticated by IIS, there are two steps we need to take:

- Configure IIS to perform authentication.
- Configure ASP.NET to use the IIS authentication information.

Once we have done this, we will be able to access the authenticated identity of the user in our ASP.NET application and use it to make decisions.

Configuring IIS to Perform Authentication

We can use any of the IIS authentication methods that we discussed earlier. The important thing is that we turn off anonymous authentication, forcing users to be authenticated by IIS. We do this by unchecking the Anonymous Access checkbox in the Authentication Methods window (reached through the Directory Security tab of the web site properties window).



We must then select an authentication method to be used in order that users can be recognized by IIS and our ASP.NET application.

We can leave anonymous authentication as an option for users if we want to allow users to use some parts of our application without logging in. For example, we may want to allow anonymous authentication in the root folder, but require users to log in to use an administration folder. We can achieve this by restricting access to the folder in question using the usual Windows security configuration options.

Configuring ASP.NET to Use Windows Authentication

Because using Windows authentication involves handing over responsibility for authentication to IIS, we do not have to do much to configure ASP.NET to use it.

The configuration required is as simple as ensuring that the `<authentication>` element in the `web.config` is set to use the windows mode:

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

  <system.web>
    <compilation
         defaultLanguage="c#"
         debug="true"/>
    <customErrors mode="Off" />

    <authentication mode="windows">

    ...

  </system.web>
</configuration>
```

ASP.NET will now use the information that IIS provides about the current user to populate `Context.User` and we will be able to access this information in our application.

Accessing Windows User Information in ASP.NET

When Windows authentication is active, the `User` property of the `HttpContext` that is used to service the request is populated with a `WindowsPrincipal` object. This means we can access the `WindowsPrincipal` in our page classes through `Context.User`, `Page.User`, or simply `User`. These properties all refer to the same `WindowsPrincipal` object.

We can access the `WindowsIdentity` object for the user through the `WindowsPrincipal.Identity` property. This allows us to access more information about the user.

Here is a simple example that displays the username of the current user. First we create a simple `.aspx` page that includes a label to display the username:

```
<%@ Page language="c#" Codebehind="default.aspx.cs" AutoEventWireup="false"
Inherits="simpleDemo._default" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
      <HEAD>
            <title>default</title>
      </HEAD>
      <body>
            <form id="default" method="post" runat="server">
                  Current User:
                  <asp:Label id="UserNameLabel" runat="server"></asp:Label>
            </form>
      </body>
</HTML>
```

We then add some code in the code behind to display the username:

```
            private void Page_Load(object sender, System.EventArgs e)
            {
                  UserNameLabel.Text = User.Identity.Name;
            }
```

This should give us a result like the following:

If we want to access any of the members of `WindowsIdentity` that are specific to `WindowsIdenity` (rather than inherited from `IIdentity`), we must cast the `User.Identity` object to `WindowsIdentity`. For example, if we add a label to our `.aspx` page from the previous example and want to use it to display whether the current is anonymous by using the `WindowsIdentity.IsAnonymous` property, we can use the following code:

```
private void Page_Load(object sender, System.EventArgs e)
{
      UserNameLabel.Text = User.Identity.Name;
      AnonymousLabel.Text =
      ((WindowsIdentity)User.Identity).IsAnonymous.ToString();
}
```

`((WindowsIdentity)User.Identity)` gives us a `WindowsIdentity` object from `User.Identity`. We then access the `IsAnonymous` property.

Checking User's Roles

We will be looking more at general techniques for roles-based authorization in Chapter 12. Something worth noting now is that an enumeration of standard Windows roles is provided, which we can use when we call `WindowsPrincipal.IsInRole`.

For example, if we want to add code to our previous example to check whether the user is a member of the Power User role, we can add the following code (assuming that we have created a label to indicate whether the user is a power user):

```
    private void Page_Load(object sender, System.EventArgs e)
    {
      UserNameLabel.Text = User.Identity.Name;
      AnonymousLabel.Text =
((WindowsIdentity)User.Identity).IsAnonymous.ToString();
      PowerUserLabel.Text = ((WindowsPrincipal)User).IsInRole
(WindowsBuiltInRole.PowerUser).ToString();
    }
```

Note that we have to cast the `IPrincipal` in `User` to `WindowsPrincipal` in order to do this – `IPincipal` does not define an overload for `IsInRole` that accepts a `WindowsBuiltInRole` as a parameter.

Customizing Windows Authentication

Sometimes we might want to add functionality to Windows authentication. For example, we might want to use our own custom `IPrincipal` object rather than the `WindowsPrincipal` object that Windows authentication uses to populate `Context.User` by default (so that we can add information or functionality to the principal object), or we might want to do additional activities such as logging when authentication takes place.

Fortunately, as we mentioned earlier in this chapter, the Windows authentication module exposes an `Authenticate` event that we can handle to have our own code execute when authentication takes place.

Here is a simple event handler that we can add to `Global.asax` that will log each authentication to a text file:

```
public void WindowsAuthentication_OnAuthenticate(object
                        sender,WindowsAuthenticationEventArgs e)
{
  System.IO.FileStream fs = new System.IO.FileStream(Server.MapPath("")
            + @"\authentications.txt", System.IO.FileMode.Append);

  string ip = e.Identity.Name + " "
    + e.Context.Request.UserHostAddress.ToString()
    + ", ";

  byte[] b = System.Text.Encoding.ASCII.GetBytes(ip);

  fs.Write(b,0,b.Length);
  fs.Close();
}
```

First we create a `FileStream` to write to a file in our application folder:

```
System.IO.FileStream fs = new System.IO.FileStream(Server.MapPath("")
          + @"\authentications.txt", System.IO.FileMode.Append);
```

We then build a string by extracting the username and the source IP address for the request:

```
  string ip = e.Identity.Name + " "
    + e.Context.Request.UserHostAddress.ToString()
    + ", ";
```

Note that we use the `WindowsAuthenticationEventArgs` object, `e`, to access the objects we need to access to get the information for the log. We use `e.Identity` to access the `WindowsIdentity` object of the authenticated user (this is the one that is added to `Context.User`), We use `e.Context.Request` to access the `HttpRequest` object for the request.

Finally, we write the text to the file and close it:

```
  byte[] b = System.Text.Encoding.ASCII.GetBytes(ip);

  fs.Write(b,0,b.Length);
  fs.Close();
```

Each time the Windows authentication module performs authentication (once per request), this event handler will write the username and source IP address into a file.

> **Note – This method of logging is only really suitable for an application that expects a very low amount of usage. Higher trafficked systems would use a more sophisticated logging system. However, this example does show how such systems can be linked into the Windows authentication module.**

Summary

In this chapter we have looked at how we can use Windows authentication to tie our authentication to that done by IIS. We have seen:

- How to set IIS up to perform Basic, Digest, or Integrated Windows authentication
- How to configure ASP.NET for Windows Authentication
- How we can access the details of the user authenticated by IIS
- How we can customize Windows authentication