Chapter 5
Advanced Web Server Security Configuration

In this Chapter

Now that you are familiar with core web security features in IIS 6.0 such as web service extensions and MIME map settings, we will examine other security options in IIS. We will take an in-depth look at the authentication mechanisms and how IIS user accounts are used. Additionally, we will look at some not-so-often-discussed configuration options that can protect your web applications.

- **Configuring Authentication**
- **Configuring IIS User Accounts**
- **Configuring URLScan**
- **Configuring Your Server to Use SSL**
- **Configuring URL Authorization with the Authorization Manager**
- **Configuring Custom Error Messages**
- **Securing Include Files**
- **Disabling Parent Paths**
- **Configuring IP Address, TCP Port and Host-Header Combinations**

By the end of this chapter you should be familiar with all aspects of the IIS request processing cycle and how settings in IIS can be used to secure your application against various forms of attack. Additional material on the configuration options and their relationship to one another can be found online at www.syngress.com/solutions.

Configuring Authentication

When IIS 6.0 attempts to read a resource from the server's disk, for example, a Hypertext Markup Language (HTML) page, an image, or an active server pages (ASP)/ASP.NET page, it impersonates a Windows user account. That user account's permissions are checked against the NT file system (NTFS) Access Control List (ACL) for the file in question to determine whether the requested action is permitted. In the special case where the end user is not required to supply credentials, IIS 6.0 impersonates the preconfigured "Anonymous User" account.

BY THE BOOK...

IIS provides 7 different authentication mechanisms:

- **Anonymous Authentication** Users do not have to supply credentials and a fixed user account is impersonated.
- **Basic Authentication** Users are prompted to supply a username and password, which are sent unencrypted across the network. Basic authentication is supported by almost all browsers.
- **Digest Authentication** A hash of the user's password is sent across the network. Digest authentication requires domain controllers to be running Windows 2000 or Windows 2003. Digest authentication requires user passwords to be stored using reversible encryption in Active Directory (AD).
- **Advanced Digest Authentication** This is similar to digest authentication in that the same hash process is used for sending the user's password from client to server. With advanced digest authentication however, the user's password is already stored as a Message Digest (MD)5 hash in Active Directory, obviating the need to store the password using reversible encryption. Advanced digest authentication requires a Windows 2003 functional level domain.
- **Integrated Windows Authentication (IWA)** Uses hashing technology to send the user's credentials across the network. IWA offers two authentication systems; NTLM v2 for legacy clients, and Kerberos for Internet Explorer v5 and later. IIS 6.0 supports both NTLM v2 and Kerberos. IWA is the default authentication mechanism in IIS 6.0.

- **UNC (Universal Naming Convention) Authentication** Allows IIS 6.0 to access resources stored on a remote computer using a preconfigured user account specified by the administrator, who has permissions to the remote resource.
- **Microsoft Passport Authentication** A single sign-on technology in which the user's identity is verified by Microsoft's Passport system and authorization to resources is determined by the application.

Any of the above authentication mechanisms can be applied to all websites, an individual website, an individual folder, or a file within a folder. For example, a website can be configured to allow anonymous access, while a single folder within that website can be secured using one of the other authentication mechanisms so that only users with a valid Windows account can get access to the resources in that folder.

Additionally, each resource can have multiple authentication mechanisms enabled. The server and browser will negotiate and choose the most secure method support by both.

The Authentication Process

Regardless of which combination of authentication mechanisms you configure for your website's resources, a browser making an initial request will not send user credentials. That is, the initial request will be made using an anonymous request. If anonymous authentication is configured for the requested resource, then IIS will impersonate the configured anonymous user account (see "Configuring Anonymous Authentication" later in this section).

If anonymous authentication is not enabled, but one of the other authentication mechanisms is enabled, the server and browser will negotiate to select the most secure authentication mechanism enabled on the server and supported by the browser, starting with Integrated Windows Authentication, then Digest/Advanced Digest, and finally Basic Authentication. Passport authentication is not included in this process, as it is a special case. Enabling passport authentication disables all other types of authentication.

Note: If no authentication mechanism is configured, the server will return a Hypertext Transfer Protocol (HTTP) 401.2 "Unauthorized: Logon failed due to server configuration" error.

For subsequent requests to the server, the browser will continue to use the credentials of the previous requests for the new request. For example, if the previous request was anonymous, then the new request will also be anonymous, and if the user supplies a valid Windows username/password using basic authentication, the browser will continue to send that username/password combination for subsequent requests. This behavior will only change if:

- The user closes the browser, in which case the next request to the website will revert to an anonymous request.
- The Web server indicates that the credentials or authentication mechanism are not valid for this new request. In this case, the browser will attempt to negotiate a different authentication mechanism and/or prompt the user for alternate credentials that are valid for the new request.

This has important implications for authentication mechanisms like basic authentication, which does not encrypt user credentials, since all subsequent requests will include the user's credentials, even if the user is not required to authenticate (that is, if anonymous authentication is allowed).

Recall from the request processing flow introduced in Chapter 4 that the authentication and authorization phases of the request processing cycle are separate. At the authentication stage, the user has to supply valid user credentials. Whether the authenticated user has appropriate NTFS file permissions to perform the requested action is determined at a subsequent point in the request processing cycle.

Configuring Anonymous Authentication

When anonymous authentication is permitted, users do not have to supply a Windows username or password to access the resource. In order to access the resource, IIS 6.0 impersonates a configured anonymous user account. In this process, IIS logs on to the server as a particular user on your behalf. If NTFS permissions allow the anonymous user account appropriate access (for example, to read a resource, or write to a file), then the action is performed.

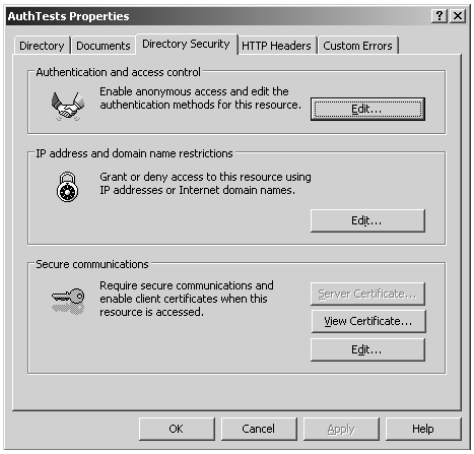
If the configured anonymous user account does not have permissions to access the resource, and an alternate authentication mechanism is enabled, and the browser supports that alternate authentication mechanism, then the user will be prompted to provide a valid Windows username and

password. If no alternate mechanism is specified, IIS will return a "401.3 Access Denied Due to ACL on Resource" error.

By default, the anonymous user account is IUSR_<webservername>. This account is created when IIS 6.0 is installed, and IIS keeps a record of the password for this account. You can change the account that is used for anonymous access, and you can enable or disable anonymous access for all websites, or for specific websites, folders, or files. Use the following steps to disable anonymous access:

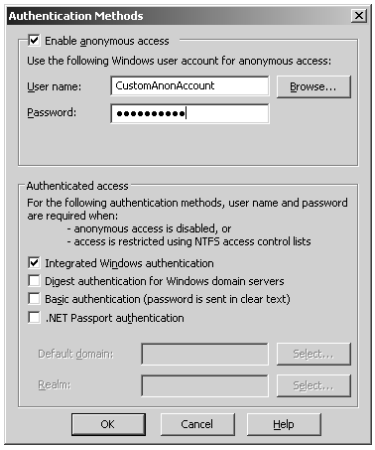
1. Open the IIS Manager. Right-click the website, folder or file you wish to edit and select **Properties**. To change the settings for all websites, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit** (shown in Figure 5.1).

Figure 5.1 Editing Authentication and Access Control



3. To disable anonymous access for the selected resource, disable the **Enable Anonymous Access** option.
4. To change the user account that is impersonated when anonymous access is enabled, enter the desired **User name** or click **Browse** to search for and select the desired username. Enter a **Password** for the user account (Figure 5.2).

Figure 5.2 Editing Anonymous Access



5. Click **OK**.

Using Windows Explorer or a command line tool, ensure that the configured anonymous user account has appropriate NTFS permissions to the website, folder, or individual files that you have just changed the anonymous user account for (for more information on setting NTFS permissions, see Chapter 4). If you use a custom account, ensure that this account has the same minimum privileges that the default IUSR_<machinename> account has. You can find information on these privileges in the "Configuring IIS User Accounts" section of this chapter.

Note that in previous versions of IIS, there was an additional option to allow IIS to control the password. This allowed IIS to impersonate the configured anonymous user account even IIS didn't have the current password for that account. This feature is disabled in IIS 6.0 by default. See "Configuring SubAuthentication" in this section for more information on this password synchronization feature, and how to enable it.

Configuring Basic Authentication

When basic authentication is configured, users are prompted to supply a valid Windows username and password in order to access resources. The username and password are base64 encoded and passed to the server.

Be aware that base64 encoding is not encryption, and can easily be decoded using readily available tools. To secure the transmission of user credentials between client and server, it is recommended that the connection be secured using Secure Sockets Layer (SSL).

After the initial request in which the user supplies access credentials, the browser will automatically continue to send the same user credentials for all subsequent requests for resources on this server (until the browser is closed). Therefore, all subsequent requests for resources should also be secured using SSL.

Basic authentication is part of the HTTP 1.0 specification, and is thus supported by all major browsers. Because of its simplicity, it can be used safely through proxy servers and firewalls. While using Basic Authentication, IIS 6.0 can access network resources (for example, if it has to log in to a remote SQL server) using the authenticated user's credentials

Note that users will not be prompted for a username and password if anonymous authentication is also enabled. When a browser makes a request for a resource, it does not send user credentials (the request is "anonymous"). If anonymous authentication is enabled, IIS 6.0 will impersonate the configured anonymous user account and process the request. To force the browser to prompt the user for credentials, anonymous authentication must be disabled.

Basic authentication can be configured for all websites, or for individual websites, folders, or files. To configure basic authentication, perform the following steps:

1. Open the IIS Manager. Right-click the desired website, folder or file and select **Properties**. To change settings for all websites, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit**.
3. Enable the **Basic Authentication (password is sent in clear text)** option (Figure 5.3).

Figure 5.3 Enabling Basic Authentication

4. IIS will provide a warning concerning the vulnerability of clear text passwords and you will be prompted to confirm your selection. Select **Yes** to enable Basic Authentication.
5. If desired, enter a Windows domain in the **Default domain** field to specify the domain against which the user's credentials will be checked if the user does not supply a domain name when prompted by his or her browser. If you do not supply a name, IIS 6.0 will use the name of the local machine.
6. If desired, enter a Windows domain in the **Realm** field. This entry will be displayed as part of the dialogue box prompting for user credentials in the user's browser. It is recommended that you make this the same as the **Default domain** field.

Configuring Digest Authentication

When digest authentication is configured, users are prompted to supply a Windows username and password. The username is passed in clear text, but the password is hashed by the client. Hashing relies on the use of mathematical algorithms that cannot be reversed. Given a "hashed" value, the original value is impossible to determine from the hash alone. Simple examples of hashing functions include trigonometric functions like Sin() and Cos(). The sine of any value yields a distinct result, however given the result, it is impossible to determine the original value, since the inverse sine of the result yields an infinite number of possible original values.

Digest authentication is defined in RFC 2617, and is an open standard. A number of browsers support digest authentication, including Microsoft Internet Explorer v5 and later, Mozilla v1.4 and later, and Opera v6 and later. Because older browsers do not support digest authentication, you may need to enable basic authentication if you want your website to support these older browsers. Digest authentication is safe to use through proxies and firewalls.

When a browser requests a resource secured with digest authentication, IIS 6.0 will send back a random piece of data called a *nonce*. The browser will generate its own piece of random data (the client nonce, or cnonce). It will then combine the cnonce with the server's nonce, the user's password, and some other data about the request, and generate a hash. The client returns this hash, plus its cnonce, to IIS 6.0. This is called *the digest*. IIS 6.0 will forward this result to the domain controller responsible for the relevant domain. The domain controller will perform the same operation on its copy of the user's password, and if the hashes match, then the user is deemed authenticated.

Because only the cnonce and hash are required to access a resource, digest authentication is susceptible to replay attacks if someone is able to capture packets between the client and the server. This replay window is limited because the server will eventually expire the nonce originally sent to the client, meaning that the hash value is no longer valid to access the resource.

Note that the user will not be prompted to supply a username and password if anonymous authentication is also enabled. When a browser makes a request for a resource, it does not send user credentials (the request is "anonymous"). If anonymous authentication is enabled, IIS 6.0 will impersonate the configured anonymous user account and process the request. To force the browser to prompt the user for credentials, anonymous authentication must be disabled.

The following are required in order to use digest authentication:

- A browser that supports digest authentication (for example, Internet Explorer 5 or later, Mozilla 1.4 or later, or Opera v6 or later).
- The IIS 6.0 server and the user account being used must both reside in the same Windows domain (or trusted domains).
- The user password must be stored using reversible encryption in Active Directory. Digest authentication is not supported for accounts that are local to the IIS 6.0 server. The user account must be a domain account.
- The domain controllers must be running Windows 2000 Server or Windows Server 2003.
- SubAuthentication must be enabled (see "Configuring SubAuthentication" later in this section).
- The process identity of the web application pool that the request is being served from must be running as LocalSystem, not as the default Network Service. See "Configuring IIS User Accounts" in this chapter for information on changing the process identity of a web application pool. Note that setting the process identity to LocalSystem could pose a security risk, as this identity has full access to the entire system.
- The UseDigestSSP metabase key must be set to 0 (false) in the IIS metabase. If this key is not present, then IIS 6.0 will assume that it is 0. This key can be set for all websites, individual websites, folders, or files. If the key is set to 1 (true), IIS 6.0 will

attempt to use advanced digest authentication instead, which may fail if the requirements for advanced digest authentication are not met.

To set this key, you can use the graphical Metabase Explorer tool supplied with the IIS 6.0 Resource Kit. You can download the IIS 6.0 Resource Kit tools from www.microsoft.com/downloads/details.aspx?FamilyID=56fc92ee-a71a-4c73-b628-ade629c89499&DisplayLang=en. Alternatively, you can use the following command line script, ensuring that you have administrative privileges on the IIS 6.0 server:

```
adsutil.vbs set w3svc/UseDigestSSP 0
```

See the IIS online help (accessible from the IIS Manager) for examples on using this script to manipulate the IIS Metabase, and visit www.syngress.com/solutions to view the appendix for additional information.

Digest authentication can be configured for all websites, or for individual websites, folders, or files. To configure digest authentication:

1. Open the IIS Manager. Right-click the website, folder, or file you wish to edit and select **Properties**. To change settings for all websites, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit**.
3. Enable the **Digest authentication for Windows domain servers** option (shown in Figure 5.4).
4. IIS will display a warning stating that digest authentication only works with domain accounts. Click **Yes** to enable digest authentication.

Figure 5.4 Configuring Digest Authentication

5. If desired, enter a Windows domain in the **Realm** field. This is the Windows domain that will be used to authenticate the user if the user does not supply a domain as part of his or her credentials. It will also be displayed to user as part of the password prompt.

IIS versions prior to IIS 6.0 contain a bug that results in basic authentication being listed prior to digest authentication if both are enabled for a resource. This results in some browsers (including Internet Explorer) choosing basic authentication instead of digest authentication. In IIS 6.0, this bug has been fixed and digest authentication is listed before basic authentication when a list of supported authentication mechanisms is sent to the client.

Configuring Advanced Digest Authentication

Advanced digest authentication is similar to digest authentication in its communication between client and server. However, advanced digest authentication differs from digest authentication in the following ways:

- In advanced digest authentication, the domain controllers (DCs) must be running Windows Server 2003, and the domain functional level must be raised to Windows 2003. Windows 2003 domain controllers store a number of hashes of a user's password when the user password is set. This includes an MD5 hash of the password. These pre-calculated hashes are stored as fields in the AltSecId field of the user object in Active Directory.
- In advanced digest authentication, the user's password does not have to be stored using reversible encryption in Active Directory. This is because the hash sent by the IIS 6.0 server to the domain controller can be compared directly with the pre-calculated MD5 password hash stored in Active Directory.
- IIS 6.0 does not require SubAuthentication, so the process identity web application pool servicing the request for the resource does not have to be LocalSystem.
- The UseDigestSSP metabase property must be set to 1 (true). If this property is set to 0, or not set at all, digest authentication will be used. Digest authentication may fail if the other requirements for digest authentication are not met.

To set the UseDigestSSP key, you can use the graphical Metabase Explorer tool supplied with the IIS 6.0 Resource Kit. You can download the IIS 6.0 Resource Kit tools from: www.microsoft.com/downloads/details.aspx?FamilyID=56fc92ee-a71a-4c73-b628-ade629c89499&DisplayLang=en.

Alternatively, you can use the following command line script, ensuring that you have administrative privileges on the IIS 6.0 server:

```
adsutil.vbs set w3svc/UseDigestSSP 1
```

See the online help system (accessible from the IIS Manager) for examples on using this script to manipulate the IIS Metabase.

Advanced digest authentication can be configured for all websites, or for individual websites, folders, or files. Use the following steps to configure advanced digest authentication:

1. Open the IIS Manager. Right-click the website, folder, or file you wish to edit and select **Properties**. To change settings for all websites, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit**.
3. Enable the **Digest authentication for Windows domain servers** option.
4. IIS will provide a warning stating that digest authentication only works with domain accounts. Click **Yes** to enable advanced digest authentication.

Configuring Integrated Windows Authentication

Integrated windows authentication is deemed by IIS 6.0 to be the most secure method of authenticating clients. When a server is configured to use IWA, the user is prompted to supply credentials; however, credentials are not passed across the network in clear text. Additionally, Internet Explorer can be configured to automatically supply the user credentials of the current user (by default, this is enabled for sites in the *intranet security zone*. For more information on the IE intranet security zone, see http://support.microsoft.com/?id=258063). If the automatically supplied user credentials are not acceptable to the IIS 6.0 server, then the user is prompted to supply alternate credentials.

IWA encompasses two authentication mechanisms:

- NTLM v2, for older clients. NTLM v2 authentication is supported by Internet Explorer v3 and later, as well as some third-party browsers, such as Mozilla v1.4.
- Kerberos v5 authentication, which is supported by Internet Explorer v5 and later.

When a browser requests a resource secured using IWA, the IIS 6.0 server returns two HTTP WWW-authenticate headers; one for Kerberos authentication and one for NTLM v2. The browser then selects the more secure of the two that it supports.

NTLM v2 authentication is similar to digest authentication. When a browser wishes to use NTLM authentication:

- The server sends a nonce to be used in creating a digest of the user's password.
- The browser hashes the user's password using the NTLM v2 algorithm. It then adds the server-supplied nonce to the result of this first hash and creates a digest by hashing this combined string. This is returned to the server.
- The server (or domain controller) already has the user's password stored as an NTLM v2 hash. It merely adds the nonce and performs the same secondary hash the client performed. If the two hashes match, the user is deemed authenticated. The benefit of this mechanism over digest authentication is that it obviates the need to store user passwords using reversible encryption.

One drawback to NTLM v2 authentication is that it requires a number of requests and responses to go back and forth between the client and server. This must be done over a continuously open HTTP connection. Because of this requirement, NTLM v2 authentication does not work through most HTTP proxies.

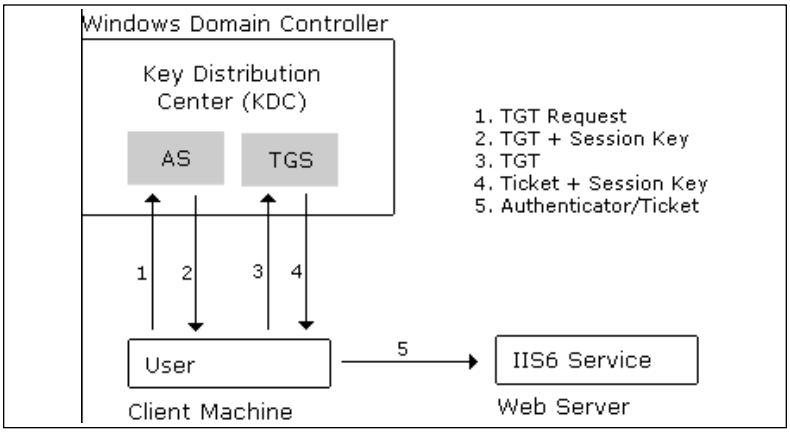
Kerberos v5 authentication is an open, industry-standard, ticket-based authentication method first developed at the Massachusetts Institute of Technology (MIT). It uses challenge/response technologies, timestamps for nonces, and a ticket granting service to facilitate a single sign-on. Kerberos v5 is a much more complex authentication mechanism than NTLM v2.

When a client wishes to access a resource secured with Kerberos authentication:

- It first contacts the Kerberos authentication server (AS). The AS, using a secret known to both the AS and client (namely a hash of the user's password) transmits a temporary *ticket granting ticket* (TGT) to the client. The TGT can then be used instead of a hash of the user's password for subsequent accesses to network resources. This obviates the need to cache the hash of the user's password (which increases security by not requiring the user to enter his or her password for each network access, and increases performance by allowing the application to cache the TGT). The TGT is valid only for a limited time, thereby reducing its usefulness to attackers in case it is stolen.
- The client then contacts the ticket granting services (TGS), to get a ticket to access the service hosting the secured resource (that is, the website hosted by IIS 6.0). The TGS transmits a ticket to the browser again using a shared secret (the TGT). Additionally, the TGS transmits a session key to the browser.
- The AS and TGS are together known as the Kerberos Distribution Center (KDC). In a Windows domain, domain controllers host the KDC.
- The browser then contacts IIS 6.0 with the ticket received from the TGS. The ticket is encrypted with a key known to IIS6, and contains a session key. The browser also transmits a timestamp encoded with the session key. The server uses the extracted session key to decode the timestamp and ensure that the time matches the web server's time (a slight discrepancy is allowed). This prevents replay attacks, since an attacker cannot generate an updated encrypted timestamp.

The Kerberos authentication method is depicted in Figure 5.5.

Figure 5.5 Client Authentication Using Kerberos v5



For more information on Kerberos authentication, the following URLs may be useful:

- **Kerberos Explained** www.microsoft.com/msj/0899/kerberos/kerberos.aspx
- **Windows 2000 Kerberos Authentication** www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/kerberos.mspx
- **JSCI Kerberos FAQ** www.wedgetail.com/jcsi/kerberos/FAQ.html

Note that the user will not be prompted to supply a username and password if anonymous authentication is also enabled. When a browser makes a request for a resource, it does not send user credentials (the request is "anonymous"). If anonymous authentication is enabled, IIS 6.0 will impersonate the configured anonymous user account and process the request. To force the browser to prompt the user for credentials, anonymous authentication must be disabled. Additionally, as mentioned previously, Internet Explorer can be configured to automatically send the credentials of the current user. In this instance, the user is not prompted for credentials, but the browser does send them to the server.

When using Kerberos authentication, IIS 6.0 can access a remote resource (for example, to log in to a remote SQL server) using the authenticated user's credentials when delegation is configured. This is not possible when using NTLM v2 authentication unless IIS 6.0 resides in a Windows 2003 domain.

To use Kerberos authentication, the following requirements must be met:

- Clients must support Kerberos authentication. This requires Internet Explorer v5 or later. Additionally, the client operating system must be Windows 2000 or Windows XP or Windows Server 2003. Windows NT 4 and earlier, and Windows 9x do not natively support Kerberos authentication.
- In Internet Explorer, the **Use Integrated Windows Authentication (requires a restart)** option must be enabled. This option is not enabled by default when using Internet Explorer v5 on Windows 2000. For more information see http://support.microsoft.com/?id=299838.
- Client machines must be able to contact the KDC or Windows domain controllers to get their Kerberos tickets. For this reason, Kerberos authentication is often described as being stopped by

firewalls, since firewalls typically do not allow computers on the unsecured side to communicate with DCs located on the secured side.

- The Service Principal Name (SPN) must be registered with Active Directory. By default, the NetBIOS name (http://servername) of the IIS 6.0 server is registered under the default application pool identity (network service) that it runs under. To make alterations or add new SPNs you use the Setspn.exe tool. If you change the account that is used as the process identity of the application pool servicing http://servername you need to reregister the SPN: Setspn.exe –A http/<servername> Domain\NewUserAccount.

  If the website is accessed using a Domain Name System (DNS) or Windows Internet Name Service (WINS) name that differs from the NetBIOS name, then this must also be registered manually using the setSPN tool. Replace the server name with the DNS or WINS name that the website is being accessed with. For more information on using the Setspn.exe tool, see: http://support.microsoft.com/?id=294382.

  Setspn.exe is part of the Windows 2000 Resource Kit, and is available for download from www.microsoft.com/windows2000/techinfo/reskit/tools/existing/setspn-o.asp.

Note that if you have multiple web applications within a website, and you assign them to web application pools that have differing process identities, Kerberos delegation will not work. Kerberos delegation requires a given SPN, being the website's host name (for example, www.myCompany.com) to be associated with a single user account. Web applications allocated to web application pools running under differing process identities share the same SPN, but do not run under the same user account, and Kerberos delegation will fail.

Integrated Windows authentication can be configured for all websites, or for individual websites, folders, or files. Use the following steps to configure IWA:

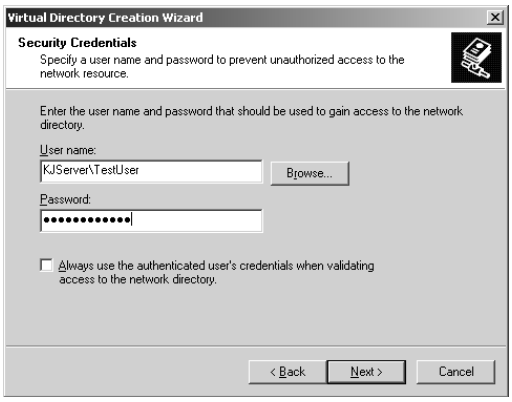
1. Open the IIS Manager. Right-click the website, folder, or file you wish to edit and select **Properties**. To change settings for all websites, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit**.
3. To enable IWA, enable the **Integrated Windows Authentication** option.

Configuring UNC Authentication

UNC authentication (also known as UNC passthrough authentication) allows you to configure IIS to use a specified user account for accessing resources on a remote share. When you create a virtual directory or a website that obtains its content from a remote share, IIS prompts you to supply a username and password for the remote share. This will be used when a user requests a resource from your website. To configure UNC authentication:

1. Open the IIS Manager, and locate the folder under which you will create a new virtual directory.
2. Right-click the folder and select **New** | **Virtual Directory**.
3. Enter an **Alias** for the virtual directory. This will be the folder name used by visitors to your site. For example, if you enter **UNCTest**, users will access this folder as http://yourserver/UNCTest/.
4. Click **Next**.
5. Use the **Path** field to enter a UNC path to a remote server, for example, **\\remoteServer\shareName**.
6. Click **Next**.
7. Disable the **Always use authenticated user's credentials when validating access to the network directory** option.
8. Enter the **User name** and **Password** that will be used to access the remote share (shown in Figure 5.6). Note that if the username and password you supply are not valid for the remote share, users will receive an "HTTP 500 Internal Server Error: Invalid Username or Password" error message.

Figure 5.6 Configuring UNC Authentication



9. Re-enter the password when prompted, then click **Next**.
10. Choose the web permissions that should be allowed for the virtual directory. The default is to allow **Read** (for static files) and **Scripts** (for dynamic content).

Passport Authentication

Passport authentication is a single sign-on authentication mechanism that is a proprietary Microsoft technology. When passport authentication is enabled on a resource, all other methods are disabled. When users access the resource, IIS checks for a passport authentication ticket cookie. If the cookie is not present, or if the credentials are not valid for the resource, the user is redirect to a Microsoft passport logon server. After authenticating, the user is redirected back to the original URL.

Enabling passport authentication requires that you sign up with the Microsoft passport service. For more information on Microsoft passport authentication, see: www.microsoft.com/net/services/passport/business.asp. For more information on enabling passport authentication on an IIS server, see: www.microsoft.com/resources/documentation/IIS/6/all/proddocs/en-us/sec_auth_passport.asp.

Configuring SubAuthentication

SubAuthentication is the mechanism by which IIS can synchronize the passwords it uses with passwords stored in Active Directory or the local security accounts database. SubAuthentication was installed by default in earlier versions of IIS, but it is not installed by default with IIS 6.0 because it constitutes a potential security vulnerability. A user with privileges to administer a website can set the anonymous user to an account with elevated privileges (for example, a domain administrator account) without supplying a corresponding password, by enabling the **Allow IIS to Control Password** option.

SubAuthentication may have to be enabled if you want IIS to synchronize passwords, or if you want to use digest authentication. To enable SubAuthentication:

1. Enter the following at the command prompt and press **Enter**:

```
rundll %windir%\system32\iissuba.dll,RegisterIISSUBA
```

2. Set the process identity for the application pool in question to **LocalSystem** (see "Configuring IIS User Accounts" in this chapter for more information on configuring web application

pool identities). Note that setting the process identity to LocalSystem could pose a security risk, as this identity has full access to the entire system.

3. Set the **AnonymousPasswordSync** metabase property to **1** (true). To set this key, you can use the graphical Metabase Explorer tool supplied with the IIS 6.0 Resource Kit. You can download the IIS 6.0 Resource Kit tools from www.microsoft.com/downloads/details.aspx?FamilyID=56fc92ee-a71a-4c73-b628-ade629c89499&DisplayLang=en.

   Alternatively, you can use the following command line script to set this property, ensuring that you have administrative privileges on the IIS 6.0 server:

   ```
   adsutil.vbs set w3svc/AnonymousPasswordSync 1
   ```

   See the IIS online help (accessible from the IIS Manager) for examples on using this script to manipulate the IIS Metabase.

To disable SubAuthentication, enter the following at a command prompt and press **Enter**:

```
rundll %windir%\system32\iissuba.dll,UnregisterIISSUBA
```

Configuring Delegation

Delegation is the process by which a service may impersonate a user account and log on to network resources on behalf of that user. Kerberos supports this process if both the computer impersonating, and the user account being impersonated are configured to be *trusted for delegation*.

In a Windows 2003 domain, delegation can be limited to specific services. So, if you enable IIS 6.0 to be able to impersonate specific users, you can limit the services that IIS 6.0 can connect to when it impersonates. In a Windows 2000 domain, this restriction cannot be set.

Perform the following steps to enable delegation in a Windows 2003 domain:

1. Open the **Active Directory Users and Computers** Administrative tool on a Domain Controller, or any machine where this tool has been installed.
2. Locate the computer account for the server that IIS 6.0 is running on. Right-click it and select **Properties**.

3. On the Delegation tab choose either **Trust this computer for delegation to any service (Kerberos Only)** or **Trust this computer for delegation to specified services only**. If you choose the latter, enable either **Use Kerberos Only** or **Use any authentication protocol**.
4. If you have chosen to allow delegation for specific services only, click the **Add** button. In **Add Services**, select **Users or Computers**. Enter the target computer name in the **Enter the object names to select** field. In the **Add Services** section, add the service(s) that IIS 6.0 can connect to.
5. Locate the user account(s) that will be trusted for delegation. Right-click and select **Properties**.
6. On the Delegation tab, enable the **Trust this user for delegation to any service (Kerberos only)** or **Trust this user for delegation to specified services only** option. If you choose the latter, enable either **Use Kerberos Only** or **Use any authentication protocol**. If you choose Use any authentication protocol, IIS 6.0 can use a Kerberos token to access a remote resource on the user's behalf even if the initial authentication to IIS by the browser was via NTLM or digest authentication. More information on Windows 2003 protocol transition can be found on the Microsoft website: www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/security/constdel.mspx.
7. If you have chosen to allow delegation for specific services only, click the **Add** button and select **Users and Computers**. Enter the name of the computer that the user will be trusted to delegate for.
8. In **Add Services**, select the service or services that will be trusted for delegation, then click **OK**.

REALITY CHECK...

IIS 6.0 offers a wide variety of standard and proprietary authentication mechanisms. The following may be used as a useful summary of benefits and drawbacks:

- Basic authentication is the most widely supported, as it is part of the HTTP 1.0 specification. It works safely through HTTP proxies and firewalls. However, user credentials are

not encrypted, and so alternate methods (such as SSL) should be used to ensure that user credentials cannot be intercepted. If IIS 6.0 has to access a network resource using the authenticated user's credentials, then this can be done when using basic authentication.

- Digest/advanced digest authentication is an improvement over basic authentication, and is an open standard defined in RFC 2617. It relies on the client supporting HTTP v1.1. The user password is hashed, and thus cannot be decrypted. However, it is vulnerable, in a time-limited sense, to replay attacks. Digest/advanced digest authentication works safely through HTTP proxies and firewalls. However there are additional server requirements above basic authentication.
- Integrated Windows authentication is deemed by IIS 6.0 to be the most secure method of authenticating clients. It comprises two authentication systems; NTLM v2, which is supported by IE v3 and later (and Mozilla v1.4 and later), and Kerberos, which is supported by IE v5 and later running on Windows 2000, Windows XP or Windows Server 2003. Depending on how delegation is configured, it may be possible for IIS 6.0 to connect to remote resources on the user's behalf.

Configuring IIS User Accounts

IIS 6.0 uses a number of built-in Windows accounts, as well as a number of IIS-specific user accounts. The user accounts that are actively used depend on whether IIS 6.0 is running in IIS5 isolation mode or in IIS 6.0 worker process mode (see Chapter 1 for more information on these modes).

BY THE BOOK...

IIS 6.0 provides two major application processing modes: IIS 6.0 worker process isolation mode and IIS 5.0 isolation mode (for backward compatibility with IIS 5.0 applications).

In IIS 6.0 worker process isolation mode web applications are assigned to web application pools, which can each be configured to use a separate process identity. The core IIS 6.0 services run under LocalSystem.

In IIS 5.0 isolation mode, web applications can either run inside the core IIS processes (running as LocalSystem), or out of

process in a separate dllhost.exe process. These processes can be assigned separate process identities.

The default accounts used by IIS 6.0 in worker process isolation mode for running any process that executes user-supplied code are low privilege accounts, which helps reduce the possible damage that a malicious attacker can inflict should an application be compromised.

IIS 6.0 Running in Worker Process Mode

When IIS 6.0 is running in worker process mode, websites and web applications (by default, a website is always configured as a web application) run inside web application pools. Each web application pool is represented by a w3wp.exe process, which is visible in Task Manager. Each w3wp.exe process has a process identity. This is the user context that the worker process runs under. This identity is required because a w3wp.exe process can be running even if there are no requests coming in from browsers.

IIS 6.0 provides the following three preconfigured user accounts that can be used as the process identity for a web application pool. You can also provide your own user account, which we will examine shortly.

- **LocalSystem** The built-in LocalSystem account has a high level of access rights. It is part of the Administrators group and can access the entire system. Running a worker process as LocalSystem can be a security risk; if the worker process or an application running inside that worker process is compromised, attackers may have full access to the system. Some IIS configurations (for example, enabling digest authentication or enabling SubAuthentication) require the relevant worker process to run as LocalSystem.
- **Network Service** The built-in Network Service account has far fewer access rights to the system than LocalSystem. This is the default process identity when creating new web application pools. The Network Service user account is able to access the same network resources as the computer it is running on.
- **Local Service** The built-in Local Service account has the same privileges as Network Service on the local machine, but is

unable to access the network. Use this account if the worker process does not need access to resources outside the local computer

- **IIS_WPG Group** The accounts mentioned are all members of the IIS_WPG group. This group is assigned the minimum permissions required for a worker process to start. If you manually create a separate user account to use as a worker process identity, ensure that it is added to this group, otherwise the worker process may fail to start.

These accounts have the following user rights (as shown with an "x" in Table 5.1). This table also lists the IUSR_<machinename> account, which will be discussed shortly.

Table 5.1 User Rights for Common IIS 6.0 User Accounts

User Right	Local System	Network Service	Local Service	IIS_WPG Group	IUSR_<machinename>
Full Access	x				
Replace a Process Level Token (SeAssignPrimaryTokenPrivilege)		x	x		
Adjust Memory Quotas for a process (SeIncrease Quota-Privilege)		x	x		
Generate Security Audits (sseAuditPrivilege)		x	x		
Bypass Traverse Checking (SeChangeNotifyPrivilege)		x	x	x	x
Access this computer from a network (SeNetworkLogonRight)		x	x	x	x
Logon as a Batch Job (SeBatchLogonRight)		x	x	x	x

Continued

Table 5.1 User Rights for Common IIS 6.0 User Accounts

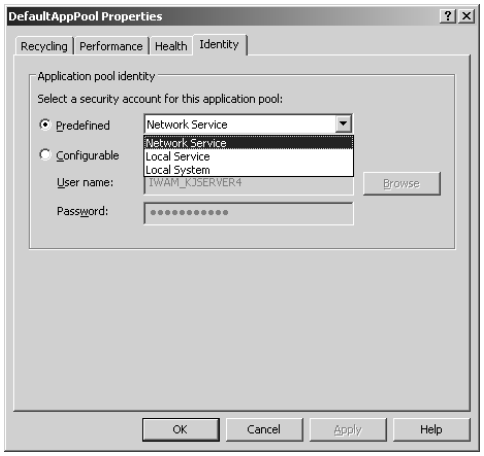
User Right	Local System	Network Service	Local Service	IIS_WPG Group	IUSR_<machinename>
Logon as a Service (SeInteractiveLogon-Right)		x			
Allow Logon Locally (SeInteractiveLogon-Right)		x			x

Changing the Process Identity of a Web Application Pool

Perform the following steps to change the process identity of a web application pool:

1. Open the IIS Manager and expand the **Application Pools** node. Right-click the web application pool that you wish to change the identity of and select **Properties**.
2. On the **Identity** tab, select one of the three preconfigured accounts from the drop-down list (shown in Figure 5.7) or specify a custom account by selecting the **Configurable** option.

Figure 5.7 Configuring a Web Application Pool Identity



3. Click **OK**.

Other User Accounts – IUSR_<machinename>

The IUSR_<machinename> account is also important when running IIS 6.0 in worker process isolation mode. When anonymous authentication is configured, the user requesting the resource from IIS does not have to provide Windows user credentials. Instead, IIS impersonates the configured anonymous user account, which is IUSR_<machinename> by default.

Note that by default, ASP.NET applications do not use IUSR_<machinename> for anonymous requests. ASP.NET applications use the process identity of web application pool they are in. This can be changed to the IIS anonymous user account in the web.config file by adding:

```
<identity impersonate="true">
```

All other requests (for static files, or ASP applications) use IUSR_<machinename>. Table 5.1 lists the user rights held by IUSR_<machinename>.

You can change the account that is used for anonymous access for all websites, or for individual websites, folders, or files. To do so, perform the following steps:

1. Open the IIS Manager. Right-click the website, folder, or file you wish to edit and select **Properties**. To change settings for all website, right-click the **Websites** node instead.
2. On the **Directory Security** or **File Security** tab, click **Edit**.
3. To disable anonymous access for the resource, disable the **Enable Anonymous Access** option. To change the user account that is impersonated when anonymous access is enabled, enter the **User name** of the user account, or click the **Browse** button to search for and select the account. Enter the **Password** for the user account and click **OK**.

IIS 6.0 Running in IIS5 Isolation Mode

When running in IIS 5.0 isolation mode, web application pools are not used to host websites or web applications. Instead, each website or web application can be set to one of three isolation levels:

- **Low Isolation** When set to low isolation, the web application runs inside the InetInfo.exe process. This process runs as the built-in LocalSystem.
- **Medium Isolation** When set to medium isolation, the web application runs inside dllhost.exe. A single dllhost.exe process hosts all web applications set to medium isolation. By default, the process identity of dllhost.exe is IWAM_<computername>.
- **High Isolation** When set to high isolation, the web application runs inside a dedicated dllhost.exe. There will be one dllhost.exe process for each web application configured to use high isolation. As with medium isolation, the process identity for the dllhost.exe process is IWAM_<computername>.

Note that these descriptions do not apply to ASP.NET applications. All ASP.NET applications run inside a single, separate process called aspnet_wp.exe. This uses the local ASPNET account as its process identity. Table 5.2 lists the user rights held by these accounts. A user right held by an account is indincated with an "x." If a particular user is explicitly denied a right, it is indicated with a "denied."

Table 5.2 User Rights Held by Common IIS 5.0 Isolation Mode User Accounts

User Right	IUSR_ <machinename>	IWAM_ <machinename>	ASPNET
Replace a Process Level Token (SeAssignPrimary-TokenPrivilege)		x	
Adjust Memory Quotas for a process (SeIncreaseQuota-Privilege)		x	
Bypass Traverse Checking (SeChangeNotify-Privilege)	x	x	

Continued

Table 5.2 User Rights Held by Common IIS 5.0 Isolation Mode User Accounts

User Right	IUSR_<machinename>	IWAM_<machinename>	ASPNET
Access this computer from a network (SeNetworkLogon-Right)	x	x	x
Logon as a Batch Job (SeBatchLogon-Right)	x	x	x
Logon as a Service (SeInteractiveLogon-Right)			x
Allow Logon Locally (SeInteractiveLogon-Right)	x		Denied
Logon through Terminal Services (SeDenyRemoteInteractiveLogonRight)			Denied

IWAM_<computername> Account

IWAM_<computername> is the default process identity for *out of process* web applications. Out of process refers to being outside the code InetInfo.exe process, and thus refers to medium and high isolation applications.

Use the following steps to change the user account used for an out of process application:

1. Open the **Component Services** MMC snap-in, located in the **Administrative Tools** folder.
2. Expand the **Computer** node, then expand the **COM+ Applications** node.
3. Right-click **IIS Out-Of-Process Pooled Applications** and select **Properties**.

4. On the **Identity** tab, select one of the preconfigured accounts or enter your own custom account and corresponding password.
5. Click **OK**.

ASPNET Account

The ASPNET account is used as the process identity for the aspnet_wp.exe process. This process is used to host all ASP.NET applications running on the server.

To change this process identity you need to edit the **<ProcessModel>** section of the machine.config file located in %windir%\Microsoft.Net\Framework\<framework_version>\config\. The machine.config file is an XML file, and can be edited in any text editor.

IUSR_<machinename>

The IUSR_<machinename> account is used for the same purposes in IIS5 isolation mode as in IIS 6.0 worker process isolation mode.

REALITY CHECK...

IIS 6.0 ships in a locked-down configuration, and this extends to the user rights granted to the accounts that are used in a default IIS configuration. Unless you have a good reason to do so, it is generally unwise to change the default configuration.

You may wish to change the anonymous user account to a domain account if your web application requires the privileges that a domain user has. Or, if you are a hosting company that needs to strictly isolate each client's website, you will also need to have a custom configuration. In other circumstances however, the default configuration is a good compromise between safety and flexibility.

Configuring URLScan

Microsoft provides an Internet Server Application Programming Interface (ISAPI) filter called URLScan, which is designed to examine incoming requests very early in the processing cycle, and to reject requests that are not deemed to be acceptable. URLScan was initially released with the

IISLockDown tool. The IISLockDown tool, when run on Windows 2000 machines, disables a number of IIS features that were enabled by default, thus reducing the attack surface of IIS 5.0. There is no IISLockDown tool for IIS 6.0, as IIS 6.0 ships in a locked-down state.

BY THE BOOK...

URLScan is a security tool that restricts the types of HTTP requests that IIS will process. By blocking specific HTTP requests, the URLScan security tool helps prevent potentially harmful requests from reaching the server. URLScan v2.5 has been updated to work with IIS 6.0, and installs on servers running IIS 4.0 and later.

Many of the features of URLScan were absorbed into IIS 6.0. However, URLScan does offer a number of features that are not available with IIS 6.0, and also offers additional flexibility that is not available with IIS 6.0.

Microsoft provides information about URLScan capabilities at www.microsoft.com/technet/security/tools/urlscan.mspx. Included is a comparison between URLScan's capabilities and IIS 6.0 native capabilities to help evaluate whether URLScan is appropriate for your server.

URLScan can be downloaded from www.microsoft.com/technet/security/tools/urlscan.mspx. To install URLScan, run the setup.exe file. To uninstall it at any time, use the **Add/Remove Programs Control Panel**. Once URLScan is installed, you can configure its settings by navigating to %windir%\system32\inetsrv\urlscan, which contains the URLScan.ini file. Open this file in Notepad.exe (or a similar text editor) to edit the settings. URLScan.ini settings are read in by the URLScan filter when IIS is started. For changes to the settings to take effect, you will have to restart the IISAdmin service. You can do that within the IIS Manager by right-clicking on your server and selecting **All Tasks** | **Restart IIS**. You can also restart IIS from the command line by typing **iisreset.exe**.

Configuring URLScan.ini

The URLScan.ini file is divided into sections. The first section, **[Options],** contains most of the major settings (listed in Table 5.3). Other sections contain supplemental information pertinent to the selections made in the **[Options]** section. To comment out any particular setting, begin the line with a semicolon.

Table 5.3 The [Options] section of URLScan.ini

Parameter	Explanation
UseAllowVerbs = 0 | 1	If UseAllowVerbs is set to 1, the HTTP verbs (for example, GET and POST) listed in the [AllowVerbs] section will be used. Requests using other verbs will be rejected. If set to 0, requests using the HTTP verbs listed in the [DenyVerbs] section will be denied, and all other requests allowed. The default is 1.
UseAllowExtensions = 0 | 1	If UseAllowExtensions is set to 1, requests for files ending in the extensions listed in the [AllowExtensions] section will be allowed, and all others denied. If set to 0, requests for files ending in the extensions listed in the [DenyExtensions] section will be denied, and all others allowed.
NormalizeUrlBeforeScan = 0 | 1	Requests can be encoded. Here a value in the URL is replaced with a % sign followed by the numerical ASCII value. For example, the "." character can be encoded as %2E, the letter "a" as %61, and so forth. Setting NormalizeUrlBeforeScan to 1 un-encodes the URL before attempting to match any of the rules specified in URLScan.ini. This prevents attackers from attempting to bypass URL restrictions (for example, by encoding extension). The NIMDA worm was able to spread from IIS server to IIS server by exploiting an un-encoding bug in IIS. The default is 1.

Continued

Table 5.3 The [Options] section of URLScan.ini

Parameter	Explanation
VerifyNormalization = 0 | 1	An encoded request can be encoded again. For example the character "." can be encoded as %2E. The % can be encoded as %25, resulting in %252E. Previous versions of IIS were found to be vulnerable to attacks involving multiple levels of encoding. By setting VerifyNomalization to 1 in conjunction with NormalizeUrlBeforeScan), URLScan will canonicalize the URL, then repeat the process on the un-encoded URL and compare the results. If they are different, the URL has been encoded more than once, and the request will be rejected. The default is 1.
AllowHighBitCharacters = 0 | 1	AllowHighBitCharacters = 1 allows requests to contain UTF8 characters in the URL. High bit characters may be required for languages that contain extended character sets. If your files are named using only ASCII characters, this should be set to 0. The default is 0.
AllowDotInPath = 0 | 1	AllowDotInPath determines whether URLs that contain a "." character that is not part of the file extension should be allowed. A setting of 0 denies requests with the "." character if it's not part of the file extension. The presence of a "." character may indicate a *directory traversal* attack, where the attacker attempts to navigate outside the web root using a URL that contains "../" to move up a directory from the current directory. It may also indicate an attack that attempts to call a denied file, but attempts to hide the attack by including the name of safe file in the URL, for example, `/someExecutable.exe?someSafeFile.html`

Continued

Table 5.3 The [Options] section of URLScan.ini

Parameter	Explanation
	However, some web applications (such as Outlook Web Access) may require you to set this to 1, because filenames for individual messages are based on the subject of the message, and the subject may contain the "." character.
RemoveServerHeader = 0 | 1	As part of an HTTP response, the server normally returns an HTTP server header indicating the type of server responding. IIS 6.0 returns "Server: Microsoft-IIS/6.0". By setting this to 1, this behavior is suppressed. Some corporate policies require this setting to obscure the brand and version of the server. However, this does not provide protection against automated attacks that systematically attempt to exploit vulnerabilities from a wide variety of platforms, nor will it prevent OS fingerprinting via other means. By default, this is set to 0.
AlternateServerName = <name>	If RemoveServerHeader is set to 1, you can supply an alternate HTTP Server: header by supplying a value for AlternateServerName.
EnableLogging = 0 | 1	If set to 1, URLScan will log rejected requests to a URLScan logfile. If set to 0, logging is not enabled.
PerProcessLogging = 0 | 1	If set to 1, URLScan will create separate log files for each w3wp.exe worker process. The log file name includes the process ID (PID) of each worker process. If set to 0, all rejected requests are logged to the same file.
PerDayLogging = 0 | 1	If set to 1, URLScan will create separate log files each day. The log file name will contain (in MMDDYY format) the day pertaining to the log file. If this setting is used in conjunction with PerProcessLogging, the file name will contain both the date and the PID in the format, for example, Urlscan.DDMMYY.<processID>.log.

Continued

Table 5.3 The [Options] section of URLScan.ini

Parameter	Explanation
LoggingDirectory = <path>	A full path that indicates where URLScan logs should be stored. By default, this is %windir%\system32\inetsrv\urlscan\logs\.
AllowLateScanning = 0 | 1	This setting determines whether the URLScan filter is a high priority filter (it applies before other ISAPI filters) or a low priority filter (it applies after high priority filters). AllowLateScanning = 0 loads URLScan as a high priority filter, and is the default. If you wish to use Frontpage Server Extensions, you will need to set this to 1.
UseFastPathReject = 0 | 1	This setting is used to determine the user experience and IIS logging of rejected requests. Setting UseFastPathReject = 1 will cause URLScan to send a plain "404 File Not Found" error message to the client, and URLScan will not log the rejection in the IIS logs.
RejectResponseURL = <URL>	If UseFastPathReject is set to 0, you can deliver a customized "404 File Not Found" page by supplying a valid virtual path for this parameter. For example, /someDirectory/someErrorPage.htm. This means that you can deliver the same rich user experience as with legitimate requests (that is, non-blocked) requests for non-existent resources. Additionally, the following variables are available as part of the request context, which can be accessed from an ASP page or ASP.NET page (in the Request.ServerVariables() collection): ■ HTTP_URLScan_Status_Header = why the request had been blocked. ■ HTTP_URLScan_Original_Verb = the request's HTTP verb.

Continued

Table 5.3 The [Options] section of URLScan.ini

Parameter	Explanation
	■ HTTP_URLScan_ Original_URL = the original URL requested. If RejectResponseURL is set to /~*, URLScan enters a special logging mode where requests are not rejected, but requests that would be rejected are still logged in the URLScan log. This is useful for testing your URLScan.ini settings.
LogLongURLs = 0 | 1	Setting LogLongURLs allows URLScan to log rejected URLs up to 128KB. If set to 0, only the first 1KB of a rejected URL will be logged.

The URLScan.ini file contains a number of additional sections, which we'll examine briefly here.

Other Sections

The [AllowVerbs] and [DenyVerbs] sections define the HTTP verbs (also known as methods) that URLScan permits. URLScan decides which section to use based on the value of the UseAllowVerbs parameter examined in the [Options] section. Common HTTP verbs include GET, POST and HEAD. Other verbs are used by applications, such as FPSE and Web Distributed Authoring and Versioning (WebDAV).

Both the [AllowVerbs] and the [DenyVerbs] sections have the same syntax. They are made up of a list of HTTP verbs, and each verb appears on its own line. URLScan.ini comes with some predefined default lists.

The [DenyHeaders] section allows you to deny requests that contain any of the specified HTTP headers in the request. When a client makes a request to the server, it sends a set of HTTP headers. These commonly include the *User-Agent* (a string that describes the browser), *Referer* (the page the browser came from) and *Accept* (which types of files the browser can accept). To block a request based on the presence of a HTTP header, add the header name followed by a colon. URLScan.ini contains a default list of HTTP headers that block WebDAV requests.

The [AllowExtensions] and [DenyExtensions] sections permit you to define requests for files with extensions that URLScan will block. For example, you can configure URLScan to reject requests for .exe files to prevent Web users from executing applications on your system. URLScan

decides which section to use based on the value of UseAllowExtensions discussed in the [Options] section.

Both the [AllowExtensions] and the [DenyExtensions] sections have the same syntax. They are made up of a list of filename extensions, and each extension appears on its own line. The extension starts with a period (.) (for example, **.ext**). You can configure URLScan to block requests that contain certain sequences of characters in the URL using the [DenyUrlSequences] section. For example, you can block requests that contain two consecutive periods (..), which are frequently used with exploits that take advantage of directory traversal vulnerabilities. To specify a character sequence to block, put the sequence on a line by itself in the [DenyUrlSequences] section.

Note that adding character sequences may adversely affect Outlook Web Access (OWA) for Microsoft Exchange. When you open a message from OWA, the subject line of the message is contained in the URL that is requested from the server. Subject lines that contain characters or sequences listed in the [DenyURLSequences] cannot be previewed, opened, or moved by OWA users.

The [RequestLimits] section allows you to limit the size of any part of the incoming request, including limits on the length of individual HTTP headers. To limit the length of any HTTP header, propend Max- followed by the HTTP header name, for example:

```
Max-User-Agent: 1000      ; limit user-agent header to 1000 bytes
```

URLScan.ini comes with default settings for overall content-length (30,000,000 bytes), maximum URL length (260 bytes) and maximum querystring length (2048 bytes).

REALITY CHECK...

Most of the functionality of URLScan is already included in IIS 6.0. Additionally, the built-in security features in IIS 6.0 provide a simpler way of maintaining your security policy. For example, disabling the ASP web services extension in the IIS Manager automatically disables all extensions that are mapped to asp.dll. To do the same thing in URLScan would require manually adding each extension (including any custom file extensions you have mapped to asp.dll).

However, URLScan does provide some advantages. It offers greater granularity that IIS 6.0 in rejecting requests. If you require the granularity provided by URLScan, it cannot easily be replicated in IIS 6.0's native features. URLScan also intercepts requests very

early in the request processing cycle, leading to faster rejection of disallowed requests. By contrast, rejection of a request for a disabled web service extension occurs very late in the request processing cycle. Finally, as a defense-in-depth measure, running both IIS 6.0 and URLScan diversifies risk. A bug in either product that may make your server vulnerable may be prevented by the other, helping to keep your server uncompromised.

Configuring Your Server to Use SSL

Secure Sockets Layer is an industry standard method of encrypting traffic. While it is typically used for securing HTTP traffic, the technology can also be used for securing other types of traffic such as Simple Mail Transfer Protocol (SMTP). SSL should be used whenever you need to send sensitive information between client and server (for example, authentication credentials or user-supplied information such as credit card numbers). This is particularly important when using basic authentication, as user credentials are passed in an unencrypted format (see Configuring Authentication in this chapter for more information on basic authentication). The technologies that SSL uses can also be used to certify the identity of a server (or client), so you should use SSL whenever you need to certify the identity of your server or clients.

BY THE BOOK...

Secure Sockets Layer is a public key-based security protocol that is used by Internet services and clients to authenticate each other and to ensure message integrity and confidentiality. Certificates are used to authenticate the server (and optionally the client), and cryptography is used to ensure message confidentiality and prevent tampering.

SSL should be used to secure the transmission of any sensitive data, including user credentials and user supplied data (such as credit card numbers). Use of SSL however, does place an additional resource burden on the server, as there is an overhead involved in encrypting and decrypting packets.

SSL uses certificates and public key cryptography to establish the identity of the server or client, and to create secure, encrypted traffic between server and client.

First, the identity of the server or client and the validity of its SSL certificate are checked. When a client requests a resource using the https:// protocol (and the server is configured to allow https:// traffic), the server will return its SSL certificate. The client will perform a number of checks on this certificate:

- It will check to ensure that the certificate hasn't expired.
- It will ensure that the name of the server that it is connecting to is the same as the name of the server in the certificate (for example, this can stop a malicious attacker from setting up a site pretending to be syngress.com since only syngress.com has a certificate containing "www.syngress.com" as the site to be secured. This works in conjunction with the next check).
- It will ensure that the certificate was originally issued by a trusted Certificate Authority (CA). Browsers have built-in trust for a number of major commercial CAs such as Verisign or Thawte. You can check the trusted CAs in Internet Explorer by accessing **Tools** | **Internet Options** | **Content** | **Certificates** | **Trusted Root Certificate Authorities** in Internet Explorer. Because the browser trusts those CAs, it trusts certificates issued by those CAs. These CAs are expected to perform due diligence on applicants for certificates to ensure that only legitimate applicants are issued with certificates. This prevents a malicious user from setting up their own CA, and issuing themselves a certificate for syngress.com. The browser will not trust a certificate issued by a non-trusted CA.

Your organization (or partner organizations) can setup a CA and configure browsers within the organization to trust certificates issued by it. This is useful if you have non-public websites that have to be secured with SSL, as it avoids the expense of purchasing a certificate from a commercial CA. Optionally, a browser may also check the CA's certificate revocation list (CRL) to ensure that a legitimately issued certificate (one that meets the checks) has not been subsequently revoked by the issuing CA.

If the certificate meets all these requirements, then the client and server can proceed to the next step: configuring a secure channel to encrypt traffic. If the certificate does not meet these requirements, the user will be warned about potential problems with the certificate and

must manually decide whether to proceed or not (as shown in Figure 5.8 and Figure 5.9).

Figure 5.8 The Hostname Requested does not Match the Hostname in the Certificate

Figure 5.9 The Certificate is Issued by an Untrusted Certificate Authority

To set up the encryption to be used for traffic, the browser will extract the server's public key from the server's SSL certificate. The client will generate a random session key and encrypt this with the server's public key and return it to the server. The server will use its private key to decrypt the transmission and extract the session key. Future communication between the browser and server will be based on this session key using symmetric encryption (which is faster than public/private key encryption). More information on public key encryption, certificates, and certificate trust hierarchies is available in Chapter 11, which covers

Microsoft Certificate Services. Microsoft Certificate Services can acts as a CA for your organization.

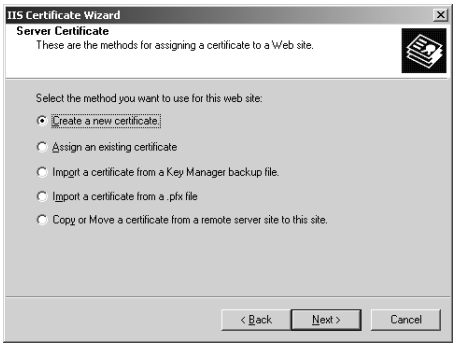
In the next section we will look at the options available in IIS 6.0 has for configuring SSL. When securing SSL websites, be aware that you cannot use host headers to run multiple SSL-secured websites on a single IP address. For more information on host headers, see "Configuring IP Address, TCP Port and Host-Header Combinations" in this chapter.

Generating a Certificate Request

The first step in configuring IIS 6.0 to allow https:// requests is to generate a certificate request. This request for an SSL certificate will be sent to a CA for processing. This can be a commercial CA (in the event that your Web server will be available to the general public), or an internal, organizational CA (in the event that the site will be accessed by internal users only). To generate a CA request:

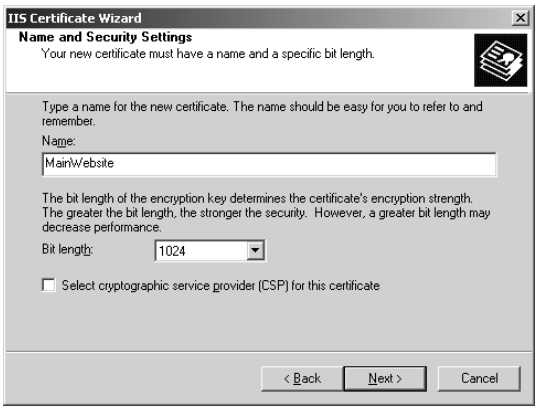
1. Open the IIS Manager. Right-click the website for which you will generate a certificate request, then click **Properties**.
2. Select the **Directory Security** tab. At this stage, your Web server does not have an SSL certificate, so the **View Certificate** option should be unavailable.
3. Click the **Server Certificate** button to begin the Web Server Certificate wizard.
4. Click **Next** on the initial Welcome screen.
5. The Web Server Certificate wizard allows you to generate a new certificate request, or to manage existing certificates. For example, if you have moved an existing website to this server, you could import the existing certificate for use with this website. In this case, we do not already have a certificate, so choose **Create a new certificate** and click **Next** (shown in Figure 5.10).

Figure 5.10 Create a New Certificate Request



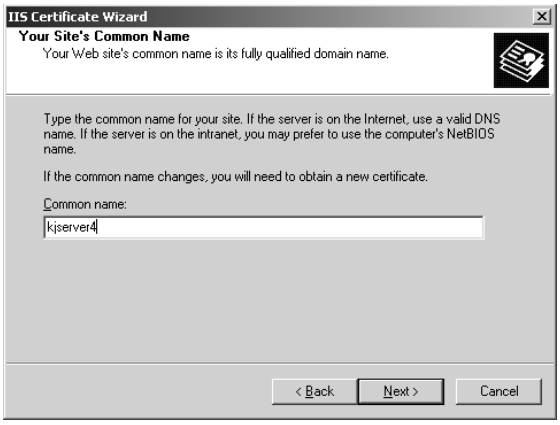
6. You can now choose to either create a request and submit it manually to a CA, or submit the request automatically to an online CA. The latter option is available if you have an Active Directory Integrated Enterprise Root CA (see Chapter 11 for more information). In this case, we will create the request, and submit it manually. Choose **Prepare the Request now, but send it later** and click **Next**.
7. On the next screen, enter a "friendly" **Name** for the certificate (Figure 5.11). Additionally, choose a **Bit length** to be used for the public key encryption. **1024** bits is the standard length. Higher values are stronger, but place an increased computational burden on your server.

Figure 5.11 Entering a Friendly Name and Configuring Key Bit Length



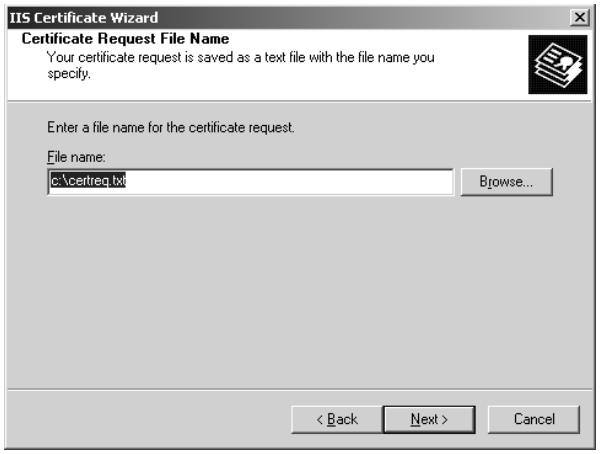
8. Click **Next**.
9. Next, enter your **Organization**, and **Organizational unit**. These do not affect the security of your certificate, but are visible to end users if they examine the details of your certificate. After entering these details, click **Next**.
10. The next screen asks for your server's common name (Figure 5.12). It is critical that you enter the correct name at this step. The name should be the same as what users will enter in their browsers to access your site. If this is a public site, you should use a Fully Qualified Domain Name (FQDN) such as **www.myCompany.com**. If this is an internal site that will be accessed by its NetBIOS name, you can enter the NetBIOS name instead, for example, **ITSupportIntranet**. Enter your server's **Common name** and click **Next**.

Figure 5.12 Enter Your Site's Common Name



11. In the next step, you are required to select your **Country** and enter your **State/Province** and **City/Locality**. These do not affect the security of your certificate, but are visible to end users who choose to view the details of your certificate. After entering the proper information, click **Next**.
12. In the next step, you save the request to a file. This file will be submitted to your CA as a request for an SSL certificate. Choose a location and click **Next** (Figure 5.13).

Figure 5.13 Choosing a Filename for the Certificate Request



13. You will be asked to confirm all the details you have entered. If you need to change any details, click the **Back** button to

return to the appropriate previous screen. When the details are all correct, press **Next** to create your certificate request, then click **Finish** to close the wizard.

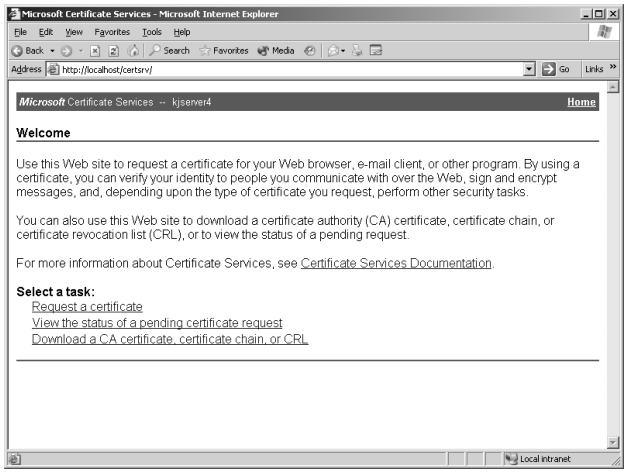
IIS 6.0 remembers that you have generated a certificate request for the website in question. The next time you start the Web Server Certificate wizard, you will have the option to process the pending request (that is, install a certificate) or delete the pending request. Choose the second option if you want to remove the existing pending request and generate a new certificate request.

Submitting a Certificate Request

The certificate request must now be submitted to a Certificate Authority. You can submit the request to a commercial CA (such as Versign, Thawte or GeoTrust), or to an internal CA. In this case we will submit the request to a Microsoft Certificate Services server. Use the following steps to submit your request:

1. Obtain the name of your Microsoft Certificate Service server.
2. Browse to **http://<certificate server name>/certsrv/** (shown in Figure 5.14).

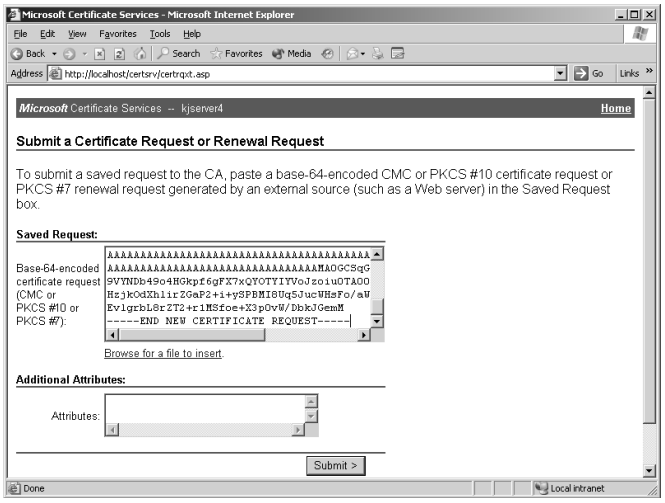
Figure 5.14 Certificate Services Website



3. Select **Request a Certificate** and then choose **Submit an Advanced Certificate Request**.

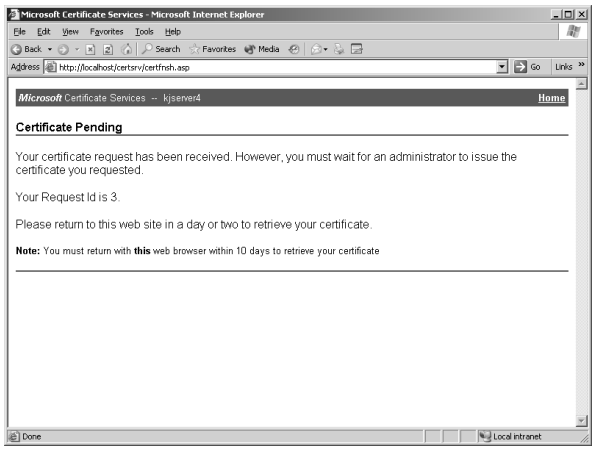
4. On the Advanced Certificate Request page, select **Submit a certificate request by using a base-64-encoded CMC or PKCS #10 file**.
5. Using Notepad, open the certificate request file you saved in the previous section (located in c:\certreq.txt by default), and paste the entire contents into the textbox (shown in Figure 5.15). By default, the **Browse for a File** option will not work if you a browsing from a Windows Server 2003 machine due to default IE security restrictions, though it will work if you are browsing from a different OS.

Figure 5.15 Entering the Certificate Request



6. Click the **Submit** button to submit your request. If the data in the certificate request file is valid, the Certificate Service website will present an acknowledgement page (Figure 5.16).

Figure 5.16 Certificate Request Submitted Successfully



Your Certificate Services administrator will now either issue a certificate or reject the request. This is done through the Certificate Services MMC snap-in.

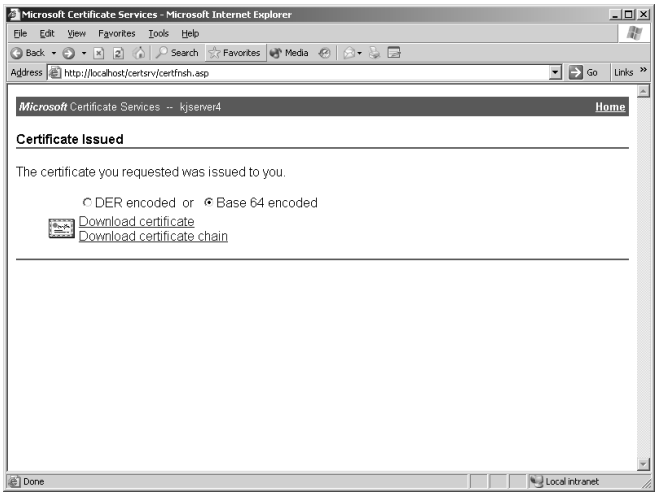
Installing an Issued Certificate

Once your certificate has been issued, you can install it on your Web server. In this section, we will download and install the SSL certificate generated by Microsoft Certificate Services. If you have submitted your request to a commercial CA, they will have procedures for you to follow to obtain your certificate.

1. Browse to **http://<certificate server name>/certsrv**, and select **View the status of a pending certificate request**".
2. Select the certificate request you submitted earlier.
3. Download the certificate using either of the encoding methods (shown in Figure 5.17) and save the file onto your hard disk.

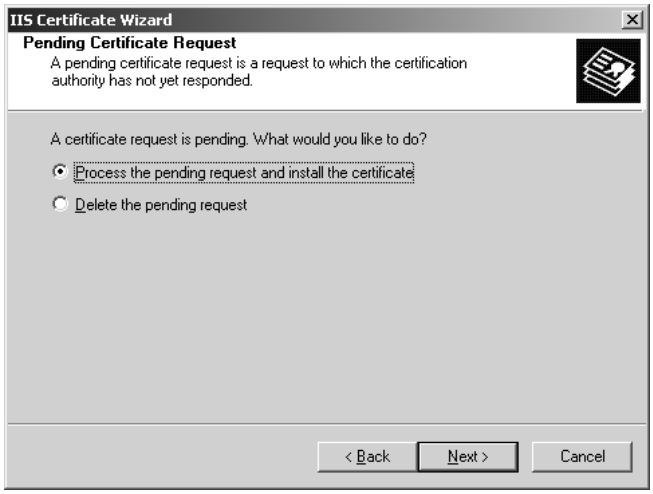
   Note that the server running Microsoft Certificate Services has URLScan installed, .cer files are blocked in the default configuration. You will need to edit the URLScan.ini file to allow requests for files with the .cer extension. This applies only to the Web server running on the Microsoft Certificate Services server, not the Web server you are installing the certificate onto.

Figure 5.17 Download the Issued Certificate



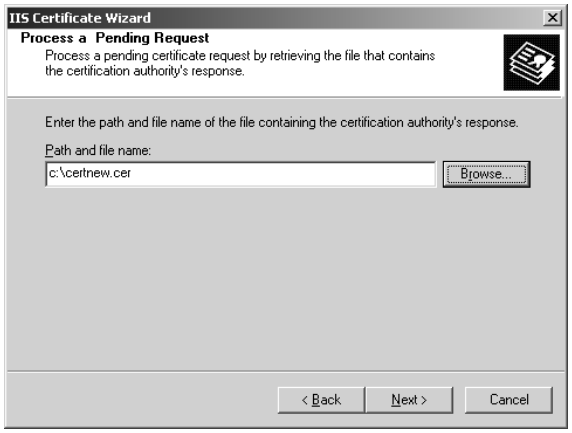
4. Open IIS Manager, right-click the website on which you will be installing the certificate and click **Properties**.
5. On the **Directory Security** tab, click the **Server Certificate** button to start the Web Server Certificate wizard.
6. Click **Next** on the initial Welcome screen.
7. Select the **Process the pending request and install the certificate** option (Figure 5.18) and click **Next**.

Figure 5.18 Installing the Issued Certificate



8. Enter the path of the certificate you downloaded and saved earlier in this section and click **Next** (Figure 5.19).

Figure 5.19 Enter the Path to the Certificate



9. Enter the SSL port that this website should use. By default, this is port **443**. Note that only one website per IP address can listen on port 443 (you cannot use host headers with SSL-secured sites). Click **Next**.
10. A screen detailing your choices will be presented. Confirm that the information is correct. If anything needs to be altered, click the **Back** button to return to the appropriate screen. If everything is correct, click **Next** to install the certificate.
11. Click **Finish** to close the wizard.
12. To verify that your SSL certificate is installed correctly, open your web browser and navigate to **https://<your server name>/**.

If you are having problems with your SSL-secured site after you finish installing the certificate, Microsoft has an SSL diagnostics tool you can run that checks for common issues. You can download SSLDiag from www.microsoft.com/downloads/details.aspx?FamilyID=cabea1d0-5a10-41bc-83d4-06c814265282&.

Managing your Website Certificates

To manage certificates issues to your websites, use the "Web Server Certificate" wizard. This wizard allows you to export certificates (for

example, if you wish to move the website to a new server), create new requests, and remove the currently installed certificate.

Configuring IIS SSL Options

To configure the website's SSL settings:

1. Open the IIS Manager, right-click the desired website, folder, or file, and select **Properties**.
2. On the **Directory Security** or **File Security** tab, select **Edit**.
3. To require an SSL connection to access the resource, enable the **Require secure channel (SSL)** option (shown in Figure 5.20).

Figure 5.20 Require a Secure Connection for the Resource

4. To require 128-bit encryption, enable the **Require 128-bit encryption** option. Older browsers, and browsers distributed in countries where US export restrictions still apply, may not support 128-bit encryption, and will not be able to negotiate a connection if this option is enabled. However, if this option is not enabled, these browsers will fall back to lower levels (for example, 40-bit encryption). These levels are no longer deemed secure because modern computers can break the encryption relatively quickly.
5. Client certificates can be used to identify clients in the same way that a server's SSL certificate identifies the server. By default, client certificates are ignored, but you can choose to **Accept client certificates** or **Require client certificates** (the former

allows client certificates as an option, and the latter requires the client to present a certificate to allow the connection).

6. To map client certificates to Windows user accounts, enable the **Enable client certificate mapping** option and click **Edit** to map certificates to user accounts. See Chapter 11 for more information on issuing certificates to clients.

7. If you are using client certificates to identify users and wish to restrict the CAs whose certificates you will accept, enable the **Enable certificate trust list** option and click **Add** to add the desired CAs. Certificates from CAs not defined here will not be accepted. This may be useful in an intranet scenario where you want only certificates issued by your own internal CAs to be used for identifying clients.
8. Click **OK**.

REALITY CHECK...

SSL provides a secure and trusted method of verifying the identity of servers and clients, and for encrypting traffic between server and client. It is designed to prevent identity hijacking threats (where a server or client is not who it claims to be), man-in-the-middle attacks (where an attacker attempts to intercept and modify traffic in transit), and snooping attacks (where an attacker tries to intercept and read traffic in transit).

Despite these benefits, there are some limitations to SSL. First, any information transmitted as part of the requested URL (for example, as part of the query string) is not encrypted. If you pass sensitive information in the URL request, it can be read by anyone intercepting traffic. The following URLs, for example, would be vulnerable:

- https://user:password@www.myCompany.com and
- www.myCompany.com?user=user&password=password

Second, information that is transmitted in an encrypted format is decrypted at both the client and server sides. An attacker who was able to compromise a client or server would be able to read the decrypted information in clear text. This could be done on-the-fly, or it could be done by retrieving information stored in databases or in IIS log files.

Another limitation of SSL is the inability to use host headers to run multiple SSL-secured websites on a single IP address. A normal, unsecured HTTP v1.1 request includes a host: HTTP header:

```
Get /default.aspx HTTP/1.1
Host: www.myCompany.com
Accept: */*
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.1; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
```

When an SSL-secured connection is used, everything except the first line (the GET request) is encrypted. Because of this, the Web server does not know to which website the request should be routed. As a result, the Host: HTTP header cannot be used to identify which server the browser is seeking the resource from. Each SSL-secured website must by run on its own IP address or TCP port (if using a single IP address and multiple websites).

For more information on host headers see "Configuring IP Address, TCP Port and Host-Header combinations" in this chapter.

Configuring URL Authorization with the Authorization Manager

Windows Server 2003 introduces a new *role-based* authorization manager. While traditional authorization has revolved around creating Access Control Entries (ACEs) on predefined resources such as files or registry keys, the Authorization Manager is designed to provide access control to tasks that comprise an application.

The Authorization Manager can be managed using an MMC snap-in. To access the Authorization Manager, select **Start** | **Run** and enter **azman.msc**. An authorization application programming interface (API) is also exposed that applications (including ASP and ASP.NET web-based applications) can utilize to access the services provided by the Authorization Manager.

BY THE BOOK...

In the Windows Server 2003 family, Authorization Manager introduces a new role-based authorization mechanism. Rather than

base access on static Access Control Entries (ACEs), access can be granted or denied based on the type of work the user is performing.

Authorization Manager allows you to define tasks and roles. Only those users who are in configured roles are allowed to execute the defined tasks. The rules governing role membership can be programmed using a scripting language, offering the ability to dynamically decide what tasks can be executed.

For example, a role called "Expense Authorizers" may allow users in the role to authorize expenses via a web-based application, but only if the expense amount is less than a specified level (which in turn may be dynamically determined, by being retrieved from a database). Users who are not in the role cannot authorize expenses at all.

A detailed analysis of the full scope of Authorization Manager is beyond the scope of this book. Authorization Manager provides a set of APIs that can be programmed against in ASP or ASP.NET applications. An example is the "Authorization and Profile Application Block" produced by the Microsoft Patterns and Practices group, which can use an Authorization Manager datastore. The Authorization and Profile Application Block can be downloaded from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/authpro.asp?frame=true.

In addition to the APIs, a URL Authorization feature is available, which allows administrators to determine who can access a given URL without setting ACEs on the physical file. Instead, rules determining who can access the URL can be managed via Authorization Manager. In this section we will examine the concepts used in Authorization Manager by building and configuring a simple, dynamic, authorization rule.

After you create the authorization store itself, you will be required to configure access to it, and to create applications, operations, scopes, and roles. Finally, you will associate resources with the applications.

Creating the Authorization Store

To begin, use the following steps to create an authorization store that will hold the application's rules:

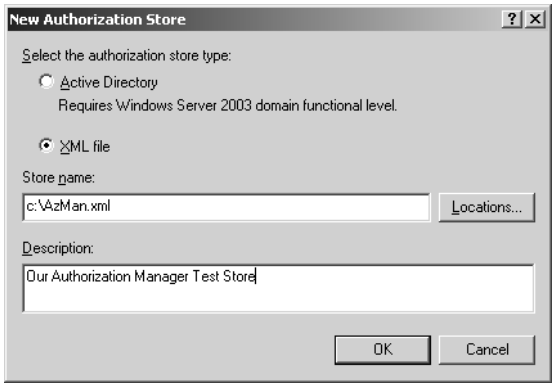
1. To open the Authorization Manager MMC snap-in, select **Start** | **Run** and enter **azman.msc**.

2. By default, the application will open in admin mode. This mode does not allow the creation of new authorization stores, only the administration of previously created stores. To create a new store, you must first switch to developer mode. To do so, right-click **Authorization Manager** and select **Options**. Select **Developer Mode** (Figure 5.21) and click **OK**.

Figure 5.21 Switching from Administrator to Developer Mode

3. Right-click **Authorization Manager**, and select **New Authorization Store**.
4. Select to either store the authorization store in **Active Directory** or in a local **XML file**. For this example, we will use an XML file. Enter the name and location in the **Store name** field or click the **Locations** button to search for and select the store (Figure 5.22). Enter a **Description** and click **OK**.

Figure 5.22 Create a new XML Authorization Store



Configuring Access to the Authorization Store

Now that the authorization store has been created, you must give the IIS worker process access to read it. The default IIS worker process identity is **Network Service**. However, if you have a web application that you'd like to secure running under a different process identity, you give that account permission to read the store instead.

1. Right-click on the authorization store you created in the previous step and select **Properties**.
2. On the **Security** tab, select **Reader** from the **User Roles** drop-down list. Click **Add** and select **Network Service** (Figure 5.23).

Figure 5.23 Allow the Network Service user to access the Authorization Store

3. Click **OK**.

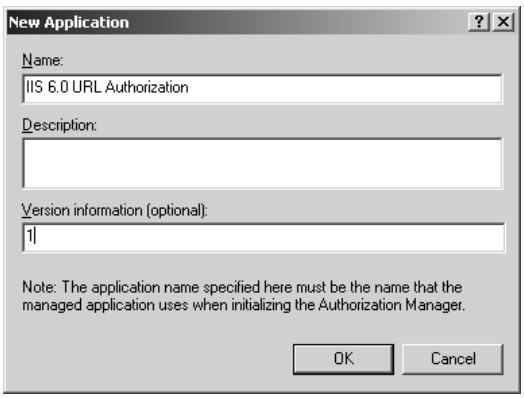
Creating a New Application

Next, you must create a new application. Each store can host multiple applications, each containing their own roles, tasks, and rules. For URL authorization, the application must be called **IIS 6.0 URL Authorization**.

1. Right-click on the authorization store you created previously and select **New Application**.

2. Enter **IIS 6.0 URL Authorization** as the application name, and enter a **Description** and **Version information** (Figure 5.24).

Figure 5.24 Creating the IIS 6.0 URL Authorization Application



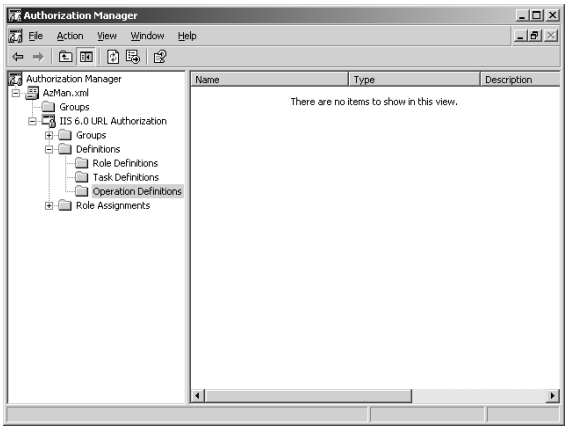
3. Click **OK**.

Creating an Operation

Operations are used to determine whether access should be granted to a specified URL. In the following steps, you will create an operation called **AccessURL**.

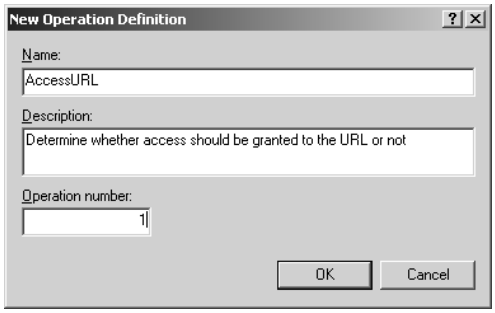
1. Expand the **IIS 6.0 URL Authorization** application that you created earlier, then expand the **Definitions** node (Figure 5.25).

Figure 5.25 Operation Definitions Node



2. Right-click **Operations Definitions** and select **New Operation**.
3. Enter **AccessURL** as the operation **Name**, and enter **1** as the **Operation Number** (Figure 5.26).

Figure 5.26 Creating the AccessURL Operation



4. Click **OK**.

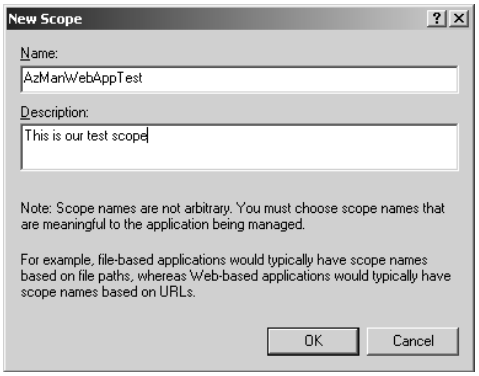
Note that in Authorization Manager, tasks are the smallest work unit. They can be combined (or grouped) into operations. Typical tasks might be "read a file", or "open a database connection". Operations combine various tasks into a logical work unit (for example, "authorize a payment").

Creating a Scope

Each web application that uses URL authentication requires a scope. Multiple web applications can share a scope, or they can each have their own scope.

1. Right-click **IIS 6.0 URL Authorization** and select **New Scope**.
2. Enter a meaningful **Name** and **Description** for the scope, then click **OK** (Figure 5.27).

Figure 5.27 Creating a Web Application Scope

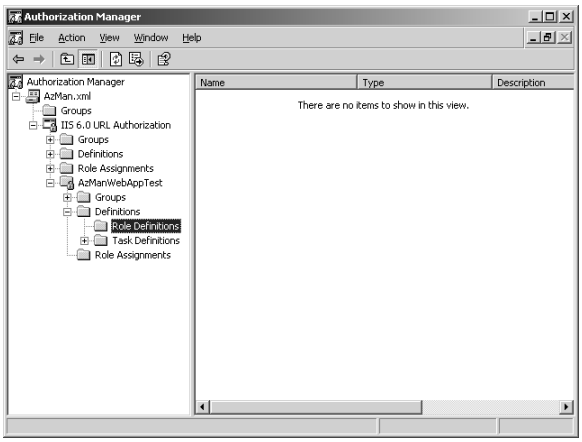


Creating a Role

Next, configure a *viewer* role for the scope. Users in this role will be able to execute the AccessURL operation that we defined earlier.

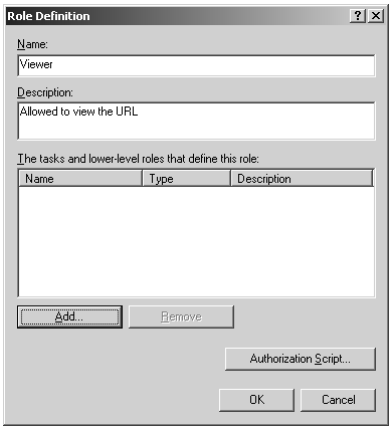
1. Expand the scope you created earlier, then expand the **Definitions** node (Figure 5.28).

Figure 5.28 Creating the Viewer Role



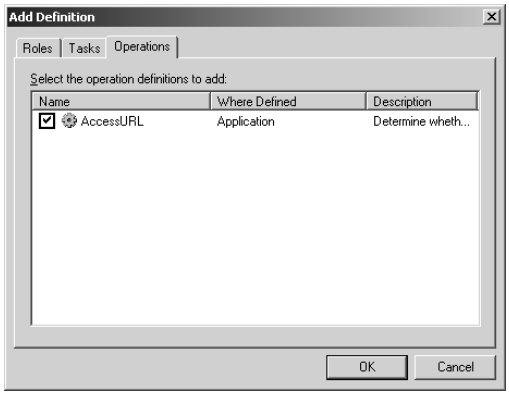
2. Right-click **Role Definitions** and select **New Role Definition**.
3. Set the **Name** to **Viewer** and, if desired, enter a **Description** (Figure 5.29).

Figure 5.29 New Role Definition



4. Click **Add** and select the **Operations** tab. Enable the **AccessURL** checkbox (shown in Figure 5.30) and click **OK**.

Figure 5.30 Adding the "AccessURL" Operation



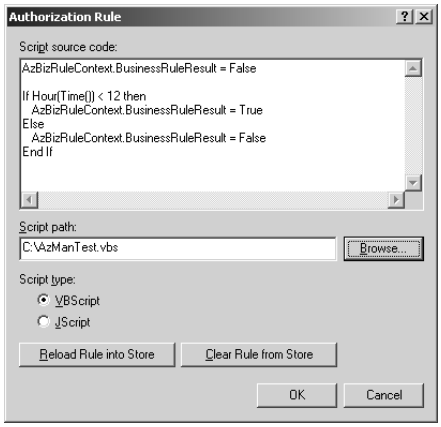
5. Next, add a business rule to dynamically determine whether the AccessURL option can be executed. Enter the following code in Notepad, and save it as **AzManTest.vbs** on your hard disk:

```
AzBizRuleContext.BusinessRuleResult = False
If Hour(Time()) < 12 then
   AzBizRuleContext.BusinessRuleResult = True
Else
   AzBizRuleContext.BusinessRuleResult = False
End If
```

6. In Authorization Manager, click the **Authorization Script** button and use the **Browse** button to load the script into the window (Figure 5.31).

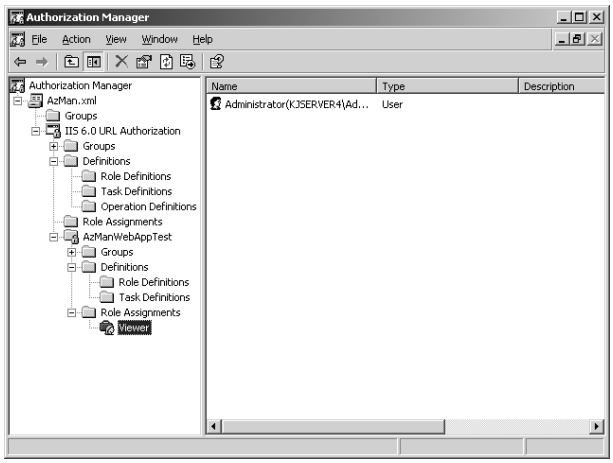
   The code allows access to the URL if the current system time is before midday, and denies access if the time is after midday. Click **OK** twice to exit and return to Authorization Manager. If your current system time is greater than 12, swap the **<** sign for a **>** sign.

Figure 5.31 Adding a Business Rule



7. Next, assign Windows users to the application role. Right-click the **Role Assignments** node and select **Assign Roles**. Enable the **Viewer** checkbox and click **OK**. Viewer should now be added as an icon under Role Assignments.
8. Right-click **Viewer** and select **Assign Windows Users and Groups**. Add the desired user account and click **OK**. The Authorization Manager should look similar to Figure 5.32.

Figure 5.32 Assigning Windows Users and Groups to a Role



Configuring IIS 6.0

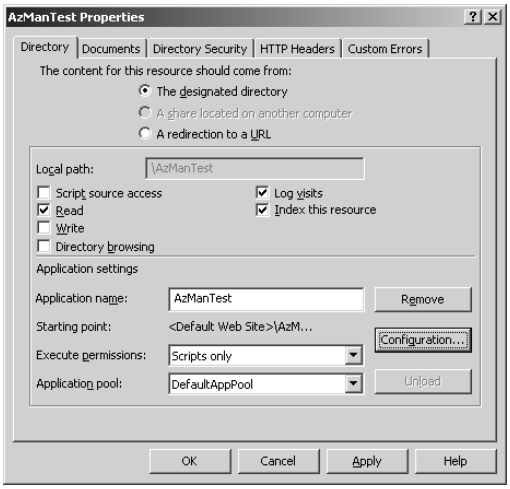
Now that we have completing defining the application in Authorization manager, we need to associate an resource (e.g. a webpage) with this application. We also need to make some configuration changes to IIS, to link IIS to Authorization Manager. To associate an IIS resource with this application:

1. Create a new folder under the default website root, and name the folder **AzManTest**. Place a simple HTML page within that folder, containing the desired text. Your entry should look like the following:

```
<html>
<body>Hello World</body>
</html>
```

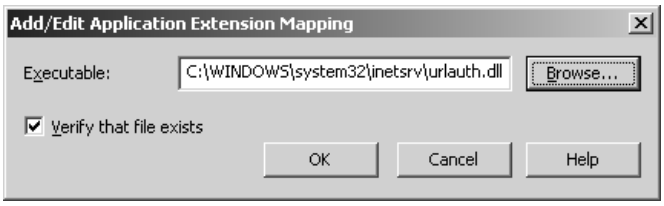
2. Open the IIS Manager, right-click the **AzManTest** folder and select **Properties**. On the **Directory** tab, click **Create** to create a new web application.
3. Click the **Configuration** button (Figure 5.33).

Figure 5.33 Configuring the IIS Web Application



4. On the **Mappings** tab, click the **Insert** button.
5. In the **Add/Edit Application Extension Mapping** window (Figure 5.34), click **Browse** and navigate to **%windir%\system32\inetsrv**. Select **urlauth.dll** and click **Open**. Click **OK** twice.

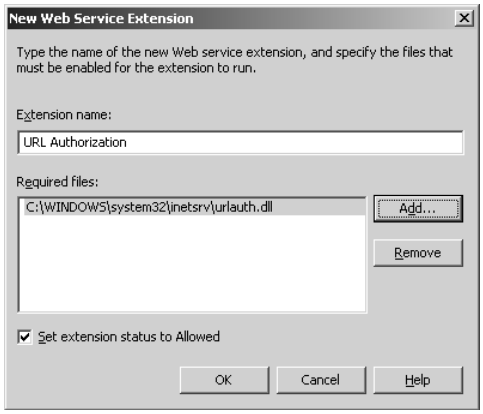
Figure 5.34 Add the URLAuth ISAPI Interceptor



6. Click the **Directory Security** tab and click the **Edit** button.
7. Disable the **Allow Anonymous Access** option. Ensure that at least one of the other options is checked.
8. In the IIS Manager, access the **Web Service Extension** node and click **Add a New web Service Extension**.
9. Click the **Add** button. Click **Browse** and navigate to **%windir%\system32\inetsrv**, then select **urlauth.dll**. Click **OK**.

10. Enter **URL Authorization** in the **Extension Name** field and enable the **Enable Extension** option (Figure 5.35).

Figure 5.35 Adding the URL Authorization Web Service Extension



11. Click **OK** and close the IIS Manager.

We will now associate the AzManTest folder with the scope we defined in Authorization Manager.

1. Use Notepad to enter the following code into a text file, and save it as **SetURLAuth.js**:

```
var objVDir =
GetObject("IIS://localhost/w3svc/1/root/AzManTest");
objVDir.AzEnable = true;
objVDir.AZStoreName = "MSXML://c:\AZMan.xml";
objVDir.AzScopeName = "AzManWebAppTest";
objVDir.AZImpersonationLevel = 1;
objVDir.SetInfo();
```

2. Replace **c:\AzMan.xml** with the path to the XML authorization store you created, and replace **AzManWebAppTest** with the name of the scope you defined under IIS 6.0 URL Authorization in Authorization Manager.
3. Double-click the JS file to run it.

Testing the Authorization Store

The final step is to test the URL authorization. To begin, open a web-browser and navigate to **http://<yourserver>/AzManTest/default.htm**. Note that if you are using Internet Explorer and **Integrated Windows Authentication** is enabled, IE will automatically log you on. Since we explicitly added our account to the **Viewer** role, we will be permitted to view the page. To prevent an auto-logon, use the IP address or the FQDN of the server instead.

According to the business rules we set up in the above exercise, if you open a new browser window and enter **http://<IPAddress>/AzManTest/default.htm**, and supply a username/password that is not associated with our usual account, we should be denied access. This is because we did not add any other users to the Viewer role in our authorization store.

To test the business rule, edit the AzManTest.vbs file you created earlier, and swap the < for a > symbol (or visa versa), so that the code should deny access based on your current time. In the Authorization Manager, navigate to the **Role Definitions** node, and double-click the **Viewer** icon. On the **Definition** tab, click **Authorization Script**, and choose **Reload Rule Into Store**. The Script Source window should reflect your change. If you now attempt to reload the page you successfully loaded before, you will be denied access with a HTTP 401.1 "Unauthorized: Access is denied due to invalid credentials" error.

Configuring Custom Error Messages

IIS provides you with the ability to return a customized URL to users when a HTTP error is generated. These are commonly used to produce a nicer user experience, especially in the case of "404 File Not Found" situations. However, using custom error messages can also provide a security benefit. In the event of an application error, a custom error message can prevent information disclosure (by preventing the user from seeing the error's source and stack trace), and by allowing the server to log the error or alert the administrator.

BY THE BOOK...

IIS 6.0 provides two methods for configuring custom error messages for ASP based applications. Either of these methods can be used when an *unhandled exception* is raised. An unhandled exception is an error that is not taken care of (handled) within the code itself. A simple generic error page can be sent back for any unhandled ASP error, or a custom page can be sent back. IIS does not handle ASP.NET errors natively in IIS 6.0. Instead, to configure a custom page for unhandled ASP.NET exceptions you must edit the ASP.NET web.config file.

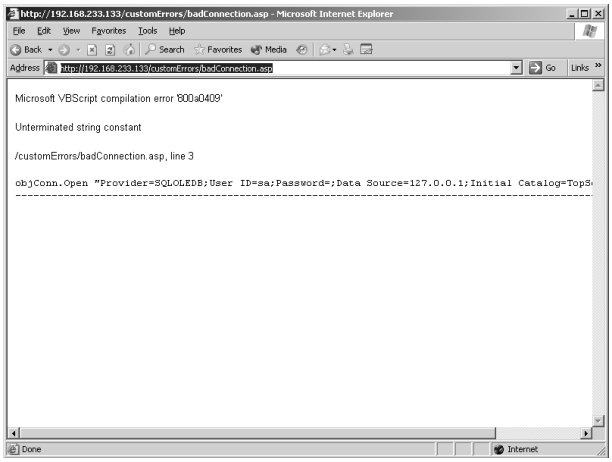
By using custom error pages for unhandled application errors, you can reduce the risk of disclosing sensitive information about the structure of the application that can aid an attacker. Some examples of information disclosure are provided later in this section.

Additionally, you can provide a better monitoring environment for administrators by having a page generate an alert (for example, via e-mail), or log an event (for example, to a database). The alert can include information about the requested URL, querystring, remote IP address, and the error type.

The Default ASP Error Message

Before customizing the error message, we will look at what the default ASP error message looks like (Figure 5.36).

Figure 5.36 Default ASP Error Message Disclosing a Connection String



This piece of programming contains a simple VBScript error on the same line that holds the connection string to an SQL server. Also visible in the connection string is the User ID being used to login to the server, the Password (a blank one), and the IP address of the SQL server (the same as the Web server). While not all coding errors might so easily result in information that could be of use to attackers, the possibility of disclosing some information that would be useful to attackers is real.

Notes from the Underground…

The Risks of Information Disclosure

Many poorly programmed web applications share a common characteristic – they fail to rigorously filter input supplied by the user. This allows a malicious attacker to mount SQL injection attacks, cross-site-scripting attacks, or replay attacks.

SQL injection vulnerabilities occur when a web application takes user input, concatenates it with an existing, predefined SQL string, and submits it to the database for processing. If the user input is not adequately screened, then it may be possible to "inject" malicious SQL code.

For example, a typical login form on a webpage might solicit a username and password from a visiting user, and concatenate that with the following string:

```
SELECT * FROM Users WHERE Username = '<supplied
username>' AND UserPassword = '<supplied password>'
```

If the attacker could enter the following as a username: ' OR 1=1 —, then the SQL statement becomes:

```
SELECT * FROM Users WHERE Username = '' OR 1=1 -- AND
UserPassword = ''
```

For an SQL server database, the double dashes (—) indicate that the rest of the line is a comment, and since 1=1 is always true, all user records will be returned from the database, allowing the user to log in. In a more malicious example, assume a form that allows a user to submit an OrderID, and the form returns all items that were purchased as part of that order:

```
SELECT ItemName, ItemDescription, Quantity FROM Items
INNER JOIN OrderItems WHERE OrderID = <user supplied
OrderID>
```

Continued

If a malicious user could enter the following as a OrderID: 1 UNION SELECT Username, UserPassword, 1 FROM Users, the final SQL statement would be:

```
SELECT ItemName, ItemDescription, Quantity FROM Items
INNER JOIN OrderItems WHERE OrderID = 1 UNION SELECT
UserName, UserPassword, 1 FROM Users
```

This would allow the attacker to see all users and their passwords in the database.

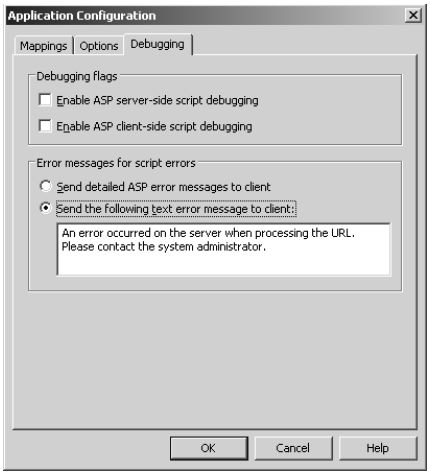
SQL injection attacks usually require the attacker to know something about the schema of the database so that they can appropriately name the tables and fields in their malicious SQL code. By failing to suppress the default error messages ASP generates, an attacker can easily map the tables and fields that do exist. An example of this is beyond the scope of this book, however the following two papers from security firm NGSSoftware demonstrate how this can be done: www.nextgenss.com/papers/advanced_sql_injection.pdf and www.nextgenss.com/papers/more_advanced_sql_injection.pdf.

Configuring a Basic ASP Error Message

IIS 6.0 provides the ability to replace the default error messages generated by ASP applications with a generic error message that masks the underlying error cause. Configuring this option is straightforward; however it provides no alerting capabilities, and does not provide a rich user experience. This setting can only be configured on a website or web application root, but not on individual folders or files.

1. Open the IIS Manager and navigate to the website or web application that you would like to configure the error for. Web applications are represented with a small cog icon. Right-click and select **Properties**.
2. On the **Home Directory** tab (for websites) or **Virtual Directory** tab (for web applications), click the **Configuration** button.
3. On the **Debugging** tab, select **Send the following text error message to the client** (shown in Figure 5.37) and edit the text if desired.

Figure 5.37 Configure a simple ASP Error Message



4. Click **OK** twice.

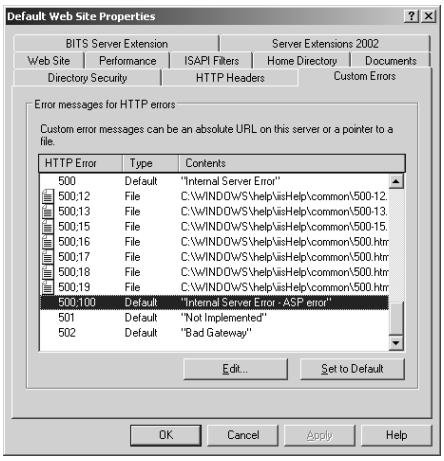
Configuring a Custom ASP Error Message

IIS 6.0 provides a more feature-rich capability that allows you to return a completely customized error page (which could be themed with your corporate style). In addition, since this page can be an ASP page itself, you can use the intrinsic ASPError object to get information about the page that generated the error, the error's source (including line number), and information posted to the server by the user that may have resulted in the error.

This custom error page can be configured for a website, a folder, or any individual ASP page within the site, providing greater flexibility for you as an administrator. To configure this option:

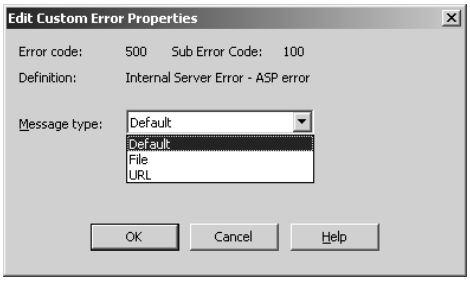
1. Open the IIS Manager, and right-click the desired website, folder, or individual ASP file. Select **Properties**.
2. On the **Custom Errors** tab, scroll down to **500:100 Internal Server Error – ASP Error** (Figure 5.38) and click **Edit**.

Figure 5.38 Locating the ASP 500:100 Error Page



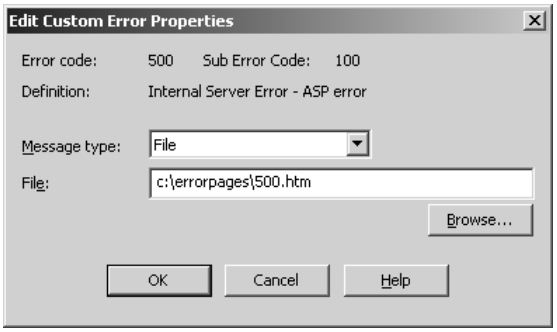
3. Select a **Message type** option (Figure 5.39) and click **OK**.

Figure 5.39 Choices for Custom Error Page



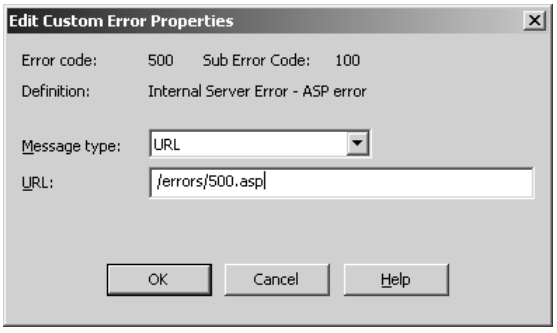
4. The **File** option serves any file off your server's hard disk without any server-side processing (as shown in Figure 5.40). This is useful only if you are serving an HTML page to the client. The HTML page could contain your corporate branding, and a message indicating that an error has occurred. If you choose this option, enter the name of the **File** that contains the error message, or click the **Browse** button to search for and select the desired file.
5. Click **OK**.

Figure 5.40 File Message Type



6. The **URL** option allows you to specify a virtual path that points to another page on the server. This is useful if the custom error page will do some server-side processing itself (for example, generating an e-mail and sending it to the administrator). If you choose this option, enter the **URL** (which can be an ASP page itself) and click **OK** (Figure 5.41).

Figure 5.41 Configuring the URL Option



If the custom 500-100.asp page is configured to be an ASP page, you can use the intrinsic ASPError object to return details of the unhandled exception. The following code demonstrates how to retrieve information that may be useful for debugging errors in your applications:

```
<%
Set objASPError = Server.GetLastError()
strErrorCode = objASPError.ASPCode
strErrorNumber = objASPError.Number
strErrorSource = objASPError.Source
strErrorFile = objASPError.File
```

```
strErrorLine = objASPError.Line
strErrorDescription = objASPError.Description
strASPDescription = objASPError.ASPDescription
strRemoteIP = Request.ServerVariables("Remote_Addr")
strHTTPReferer = Request.ServerVariables("HTTP_Referer")
strHTTPMethod = Request.ServerVariables("Request_Method")

' The following two lines get the information sent
' by the browser as a form POST or via the querystring
' If you are placing this into a database you may
' wish to truncate this in case it is larger than your
' field definition
strPostData = Request.Form
strGetData = Request.QueryString
%>
```

The information can be set up to be e-mailed to you, or logged to a database.

It should be noted that errors in the custom error page itself are not handled by serving another copy of the custom error page (that would lead to an infinite loop). Instead, you will need to use VBScript's (On Error Resume Next) or Jscript's (Try…Catch) error handling options to ensure that your custom error page itself doesn't generate an unhandled exception. For example, if your web application loses connectivity to your database server then your application may start generating exceptions. IIS 6.0 will serve the configured custom 500-100 error page. However if you attempt to connect to the same database in your 500-100 error page (without using On Error Resume Next if you are using VBScript), then the 500-100 error page itself will generate an unhandled exception, and this error will be sent directly to the user browsing your page.

Configuring a Custom ASP.NET Error Message

ASP.NET does not use the settings in IIS 6.0 to determine which error pages to supply when an unhandled exception occurs. Instead, it uses its own configuration files (typically a web.config file) to determine what should happen. Since ASP.NET does not use IIS 6.0 settings, a detailed discussion on how to configure error messages for ASP.NET is beyond the scope of this book. To set custom error pages, edit the <customErrors> node of the web.config file as follows:

```
<customErrors mode="RemoteOnly"
defaultRedirect="GenericErrorPage.htm">
        <error statusCode="403" redirect="NoAccess.htm"/>
        <error statusCode="404" redirect="FileNotFound.htm"/>
        <error statusCode="500"
redirect="InternalServerError.htm"/>
<customErrors>
```

The **mode** attribute is used to determine whether the custom errors should be displayed for all clients, or only for clients that are not browsing from the local machine. The **defaultRedirect** attribute is used for all errors that do not have a specific error message listed. For each error code that you wish to handle, add an <error> node that contains the HTTP status that the page should be served for, and a virtual path provided to, the page in question.

To handle 500 errors resulting from ASP.NET pages, you can use create an Application_Error routine in the global.asax file serving your web application. This routine will handle all exceptions not handled on a page basis.

When creating custom error messages, be aware that the error message itself must be served from the same web application pool as the page where the error was generated. Attempting to serve a custom error page from a folder that is being served by a different web application pool will generate an error.

If your custom error page is too short, Internet Explorer 5 and later will substitute its own, more detailed, error page in place of yours, unless the user chooses to disable this behavior. The user can do this by disabling the **Show Friendly HTTP Errors** option (to access this option, select **Tools** | **Internet Options** | **Advanced.** For more information, see the Microsoft KB article: http://support.microsoft.com/?id=218155.

REALITY CHECK...

Custom error pages allow you to present a rich page to your users when an error condition occurs. Additionally, error pages that handle application errors help to alert you to bugs in your application, while keeping information disclosure to attackers to a minimum (they will be aware that they have located a bug in your application, but may have some difficulty determining the nature and extent of the bug).

Regardless of whether you log the error information to a database or have the details sent via an e-mail alert, consider the use of a governor of some kind. Otherwise you run the risk on a busy site of being flooded with alerts.

Securing Include Files

Include files are a convenient way of storing commonly used HTML or code. The code or HTML is placed into a central file, and then "included" with every file that requires it. This improves the maintainability of your web applications because changes to the contents of the include file are reflected immediately in every page that includes this central file.

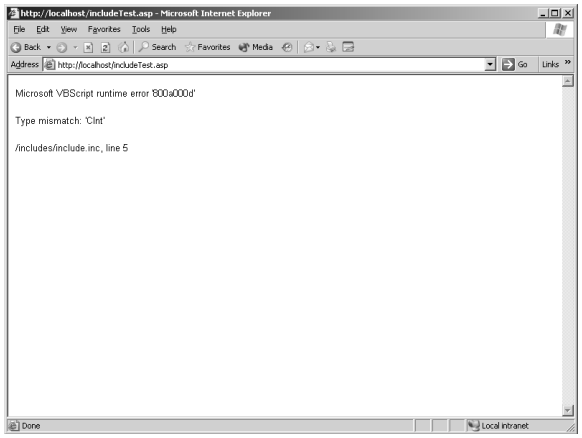
BY THE BOOK...

Include files use *directives* (pieces of code) included in web pages. IIS 6.0 provides three technologies that support server side include (SSI) directives. Files with .stm, .shtm, and .shtml extensions are processed by the SSI web service extension. ASP files (.asp) can also contain include directives. Finally, ASP.NET pages can also contain include directives. However, there are alternate ways to include content in ASP.NET pages (for example, user controls) that are superior to using include directives.

Since include files are now often used to centralize web application data and settings (such as database connection strings), it is important that any include files you do have are secured against malicious attackers.

If an attacker was able to determine the name of your include files, he or she may be able to directly request the include file. A common way of finding the name of an include file is by attempting to generate an unhandled exception (error) in your code. If the error occurs in an include page, then the default ASP error page that is generated includes the name of the include file (as shown in Figure 5.42) If you are using a customized ASP error page, you can prevent this type of information disclosure.

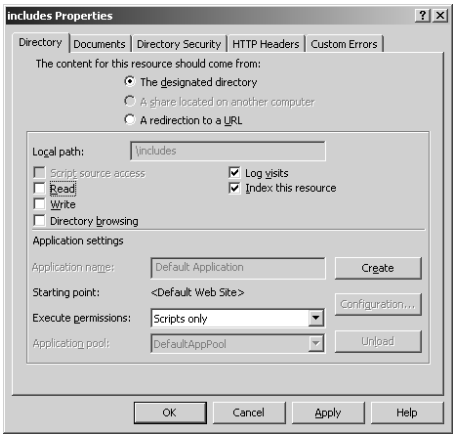
Figure 5.42 Disclosing the Name of an Include File



To prevent attackers from gaining access to possible script source code in your include pages, the following configuration steps are recommended:

- If your include files contain an extension that is not used for any other purpose in your web application (for example, .inc) and you're running URLScan, add that extension to the [DenyExtension] list.
- If your include files are named with a static file extension (that is, a file extension that's not mapped to a web service extension), such .htm or .inc, remove IIS *Read* web permissions for the folder that contains your include files (ensure that there are no static files in that folder that are required to be served directly). To do this, open the IIS Manager, and expand the folder that contains your includes files. Right-click and choose **Properties**. On the **Directory** tab, disable the **Read** option (shown in Figure 5.43). This prevents static files from being served from this directory.

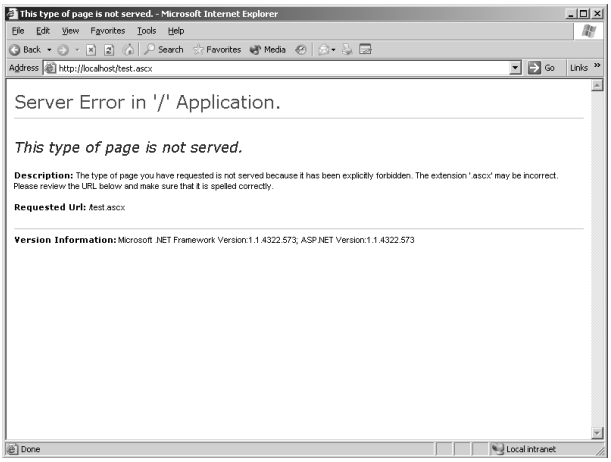
Figure 5.43 Removing IIS Web Read Permissions



- If your include files are named with a dynamic file extension (that is, a file extension that is mapped to a web service extension), such as .asp, remove the IIS script web permissions for the folder that contains your include files (ensure that there are no dynamic files in that folder that are required to be served directly). To do this, open the IIS Manager, and expand the folder that contains your includes files. Right-click and choose **Properties**. On the **Directory** tab, change the **Execute** permissions from **Scripts Only** to **None**. This prevents dynamic files from being served from this directory.

ASP.NET introduces the concept of *user controls* (with an .ascx extension). User controls are a superior way of storing commonly used content to include files. The default ASP.NET configuration prevents ASP.NET user controls from being requested by a user directly. This is set in the machine.Config file located in %windir%\Microsoft.NET\Framework\v1.1.4322\config, where the .ascx extension is mapped to the System.Web.HttpForbiddenHandler, which denies requests (as shown in Figure 5.44).

Figure 5.44 ASP.NET User Control Files are not Served by Default



REALITY CHECK...

Some security guides recommend using the .asp extension for any include file. If the include file is requested directly, it will be processed by the ASP engine, and the source will not be sent to the user (only the results of the processing).

While this may work in some cases (where the source code consists of routines, or classes), it may present problems if the included code opens database connections, generates e-mails, or instantiates other objects, since an attacker who repeatedly calls the page may start to consume an excessive amount of resources on your server (unless the include file also contains code to clean up and dispose of the objects used).

Disabling Parent Paths

Parent paths are paths that access folders located above the current folder. Enabling parent paths allows application code to access folders above the current folder.

BY THE BOOK...

When you enable parent paths, you specify that an ASP page should allows paths relative to the current directory (using the ../ notation). Parent paths are no longer enabled by default. This

affects your application if it has a web page that contains the *#include* server-side include directive and uses ".." notation to refer to a parent directory. Enabling parent paths corresponds to the metabase setting AspEnableParentPaths.

Enabling parent paths becomes a problem if the application navigates so far up the folder hierarchy that it is now outside the web root. For example, assume you have two web applications maintained by two different users, located in c:\inetpub\application1\ and c:\inetpub\application2\.

If parent paths are allowed, then a malicious coder could enter the following:

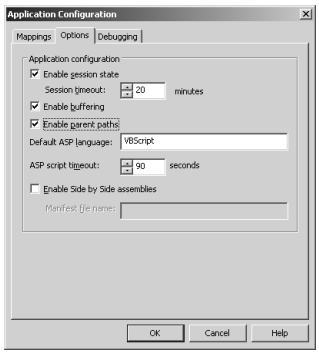
```
Response.Write(Server.MapPath("../application2/default.asp"))
```

in order to get the physical path to the default.asp page located in application2 (the ../ syntax tells the code to move up one folder). This physical path could then be used by the file system object to open the default.asp page and stream the source code back to the malicious user.

By default, parent paths are disabled in IIS 6.0. This prevents ASP code and include directives from using the "../" syntax to move up a folder from the current folder. If an application requires parent paths to be enabled, perform the following steps:

1. Open the IIS Manager and locate the website or web application root where the files that require parent path access are located. Right-click and choose **Properties**.
2. On the **Home Directory** or **Directory** tab click the **Configuration** button.
3. On the **Options** tab, enable the **Enable parent paths** option (shown in Figure 5.45).

Figure 5.45 Enabling Parent Paths



Note that enabling parent paths poses a security risk. Before enabling parent paths, ensure that the application in question does not attempt to access unauthorized resources. Microsoft recommends disabling parent paths on your web servers: http://support.microsoft.com/?kbid=184717.

Configuring IP Address, TCP Port and Host-Header combinations

Introduced as part of HTTP v1.1, the *host* HTTP header allows multiple websites to be run on a single IP address using TCP port 80 only. Prior to this, each website required its own unique IP address, or had to be run on a non-standard (not port 80) TCP port.

BY THE BOOK...

When an HTTP v1.1-compatible client makes a request to a Web server for a resource (for example, a web page, image or document), it includes the DNS or NetBIOS name (the host) of the website that it's requesting the resource from. The Web server (in this case IIS) examines the supplied host header to see if it matches any of those configured on the server. If there's a match, the normal request processing process occurs. If there's no match, IIS returns an "HTTP 400 Bad Request" error to the client browser.

When configuring websites, each website can have one or more combinations of IP address plus TCP port (this is typically port 80) plus host-header name. Each combination of IP address plus TCP port plus host header name is known as a *website identity*. Each website has at least 1 website identity, but can have more.

However, on a given Web server, each such identity must be unique. If they are not unique, when an HTTP request comes in, IIS will not know which website the request should be routed to.

Requests for a website identity that doesn't match any configured on the server will be rejected with an "HTTP 400 Bad Request" error.

When configuring a website identity, the host-header name is optional. Additionally, the IP address can be a specific IP address assigned to the machine, or you can choose **All unassigned**, in which case the website identity will include all IP addresses not already assigned to other websites.

When IIS matches incoming requests to website identities, it assigns the requests in order of specificity.

- If there is a website that has an exact match of IP address plus TCP port plus host header name, the request is routed to that website.
- If there is a website that has an IP address plus TCP port match (and no host-header name), the request is routed that that website.
- If the request does not match any of the these two, but there is a website that has all unassigned addresses for the IP address, plus a TCP port match (and no host-header name), the request is routed to this website.

Otherwise, the request is rejected.

Suppose a corporation has the following four DNS names configured as shown in Table 5.4:

Table 5.4 Sample DNS Name Configuration

DNS Name	IP Address
www.myCompany.com	192.168.0.100
mail.myCompany.com	192.168.0.100
support.myCompany.com	192.168.0.100
intranet.myCompany.com	192.168.0.200

Both 192.168.0.100 and 192.168.0.200 are assigned to the same IIS server. In the IIS MMC snap-in, the website identities shown in Table 5.5 are configured:

Table 5.5 Configured Website Identities

Website Number	IP Address	Port	Host Header Name
1	192.168.0.100	80	www.myCompany.com
	192.168.0.100	80	mail.myCompany.com
2	192.168.0.100	80	<blank>
3	<All Unassigned>	80	<blank>

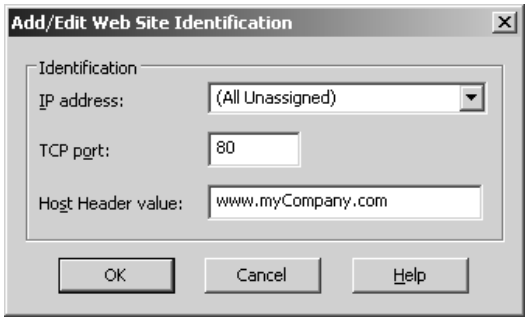
Requests for both www.myCompany.com and http://mail.myCompany.com will be answered by website 1, since there is an exact match. Requests for http://support.myCompany.com and http://192.168.0.100 will be answered by Website 2. Website 1 will not answer requests for http://192.168.0.100 because that doesn't match any of the configured identities for that website. Website 3 will not answer requests for http://192.168.0.100 because that's been allocated to website 2. Website 3 will only answer request on otherwise unallocated IP addresses.

Lastly, requests for http://intranet.myCompany.com and http://192.168.0.200 will be answered by Website 3. Website 3 will answer requests for http://192.168.0.200 because that IP address hasn't been allocated to any other website.

When IIS 6.0 is installed, the default website is configured to listen on "All Unassigned" IP addresses (which equates to all addresses assigned to the machine, since there are no other sites configured). To change this behavior, and have the default site listen on for requests for a specific DNS name perform the following steps:

1. Open the IIS Manager from the Administration Tools Folder.
2. Expand the **Websites** node.
3. Right-click the **default website** and select **Properties**. Note that unlike many properties, website identities can only be configured at the website level (not the folder, or file level).
4. On the **Website** tab, click the **Advanced** button.
5. Select the pre-existing website identity and click the **Edit** button.
6. Enter the DNS name that you wish the site to answer requests for (see Figure 5.46), and click the **OK** button.

Figure 5.46 Adding/Editing a Website Identity



7. If the site should answer requests for more than one DNS name (like website 1 in the example above), click the Add button to add another website identity.
8. Click **OK** twice to return to the IIS Manager.

REALITY CHECK…

Configuring host header names for all your websites isn't strictly a security measure. Host header names were designed to allow multiple websites to be hosted on a single IP address.

However, many automated worms that attack IIS, including Code Red and NIMDA are incapable of interrogating the DNS. Instead, they send HTTP requests to an IP address only, omitting a host name. If no website on your Web server is configured to listen on an IP address only, then the worm will never be able to have its payload examined by IIS. Additionally, your website's log files do not record numerous worm-inspired requests such as the following, which is a typical Code Red request from an infected machine to your server:

GET/default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%
u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190
%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a

Configuring host header names for all your websites will not stop a dedicated attacker, but it may make your server more resistant to automated worm-based attacks.

Your A** is Covered if You…

☑ Familiarize yourself with the available authentication methods, and the benefits and drawbacks of each. For basic authentication, evaluate the need for SSL to secure transmission of user credentials. For digest and IWA, ensure that your client browsers support these authentication mechanisms, and your server and network support the prerequisites for using these authentication mechanisms.

- ☑ Configure user accounts with the minimum privileges required for IIS web functionality.
- ☑ Are aware of which user account settings must to be configured so that you can isolate web applications from each other if required.
- ☑ Are familiar with the URLScan tool from Microsoft, and how it can help secure your Web server by providing an additional defensive layer.
- ☑ Configure appropriate application settings to protect your web applications from information disclosure attacks. You should develop custom application error pages that inform your developers of errors while hiding configuration information from malicious attackers. You should secure include files that may contain sensitive configuration information about your application.
- ☑ Be aware of the new Authorization Manager functionality included with Windows 2003, and how it allows for role-based authorization, as compared with the traditional ACE authorization method traditionally used to secure access to resources.