



How To: Create a Service Account for an ASP.NET 2.0 Application



patterns & practices
proven practices for predictable results

[<http://msdn.microsoft.com/practices>]

[patterns & practices Developer Center](#) [<http://msdn.microsoft.com/practices>]

J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Andy Wigley, Kishore Gopalan

Microsoft Corporation

August 2005

Applies To

- | ASP.NET version 2.0
- | Internet Information Services (IIS) version 6.0
- | Microsoft® Windows Server™ 2003 operating system

Summary

This How To shows you how to create and configure a custom least-privileged service account to run an ASP.NET Web application. By default, an ASP.NET application on Microsoft Windows Server 2003 and IIS 6.0 runs using the built-in Network Service account. In production environments, you usually run your application using a custom service account. By using a custom service account, you can audit and authorize your application separately from others, and your application is protected from any changes made to the privileges or permissions associated with the Network Service account. To use a custom service account, you must configure the account by running the `Aspnet_regiis.exe` utility with the `-ga` switch, and then configure your application to run in a custom application pool that uses the custom account's identity.

Contents

- [Objectives](#)
- [Overview](#)
- [Guidelines](#)
- [Summary of Steps](#)
- [Step 1. Create a New User Account](#)
- [Step 2. Assign ASP.NET Permissions to the New Account](#)
- [Step 3. Create a Test ASP.NET Application](#)
- [Step 4. Create an Application Pool with a Custom Identity](#)
- [Step 5. Configure Your Application to Run in the New Application Pool](#)
- [Step 6. Test the Custom Service Account](#)
- [Custom Account vs. Network Service](#)
- [Additional Considerations](#)
- [Additional Resources](#)

Objectives

- | Create a least privileged custom account.
- | Assign the necessary privileges and permissions to the custom account.
- | Configure an application to run using a custom service account.
- | Confirm the identity that your application uses to run.

Overview

By default, an ASP.NET application on Windows Server 2003 and IIS 6.0 runs in the application pool called ASP.NET V2.0. This application pool uses the built-in Network Service account. This account is least privileged, although it does have network credentials which means that you can use it to authenticate against network servers.

The following scenarios may prevent you from using a network service account or a custom domain-level service account:

- | Your Web server is not in a domain.
- | Your Web server and downstream remote server are in separate domains with no trust relationship.
- | Your Web server and downstream remote server are separated by a firewall and you cannot open the ports required for NTLM or Kerberos authentication.

In the above cases you can use mirrored local accounts. With this approach, you use two local accounts with the same user name and password on both servers. Alternatively, you can use SQL authentication, although this is not recommended because it offers weaker security than Windows authentication offers.

By using a custom service account and a dedicated application pool, you gain a number of advantages:

- | You help to isolate applications from one another.
- | You can establish different access controls for each application on local and remote resources. For example, other applications cannot access your application's databases if access is restricted to your application's account.
- | You can use Windows auditing to track the activity of the application separately from other applications.
- | You ensure that any accidental or deliberate changes to the access controls or permissions associated with the general purpose Network Service account do not affect your application.

Guidelines

When you create a custom service account to run your application:

- | Adhere to the principle of least privilege, and grant the account the minimum set of privileges and permissions required.
- | Avoid running ASP.NET using the SYSTEM account.
- | Avoid granting your application's account the Act as part of the operating system user right.

Note An ASP.NET application does not impersonate by default. Therefore, access controls on resources must be configured for your application's identity, and not for end user accounts or user groups.

Summary of Steps

Follow these steps to create and test a dedicated application pool that uses a custom service account identity:

- | Step 1. Create a new user account.
- | Step 2. Assign ASP.NET permissions to the new account.
- | Step 3. Assign minimum privileges to the new account.
- | Step 4. Create a test ASP.NET application.
- | Step 5. Create an application pool with a custom identity.
- | Step 6. Configure your application to run in the new application pool.
- | Step 7. Test the custom service account.

Step 1. Create a New User Account

Start by creating a new Windows account.

To create a new account

1. Create a new local or domain user account. Create a local account by using the Computer Management tool in the Control Panel. Create a domain account by using the Active Directory Users and Computers tool in the Control Panel.
2. Give the account an appropriate name, for example, CustomASP. Clear the User must change password at next logon and select Password never expires.

Note Make sure you use a strong password for the account. Strong passwords should include at least seven characters and should be a mixture of uppercase and lowercase letters, numbers, and other characters such as *, ?, or \$.

Step 2. Assign ASP.NET Permissions to the New Account

When you use a custom service account, the account needs appropriate permissions to access the IIS metabase and the file system folders that are used by ASP.NET. ASP.NET 2.0 provides the `Aspnet_regiis.exe` utility, which allows you to grant appropriate permissions.

To assign ASP.NET permissions to the new account

1. Run the following command from a command window.
`aspnet_regiis -ga MachineName\AccountName`

Where *MachineName* is the name of your server or the domain name if you are using a domain account, and *AccountName* is the name of your custom account.

1. Review the permissions required by your custom account. When you run `Aspnet_regiis.exe` with the `-ga` switch, the command grants the following rights to the account:
 - I Access to the IIS Metabase
 - I Permission to write to the `%Windir%\Microsoft.NET\Framework\version\Temporary ASP.NET Files` folder

The account is also a member of the local Users group; therefore, it has read access to the `\inetpub` directory tree (these directories have an ACL that grants read access to the Users group).

Note The `-ga` switch makes a number of global changes. If you want to restrict access to specific folders, you need to manually adjust the ACLs on those folders.

Step 3. Create a Test ASP.NET Application

In this step, you create a test ASP.NET application with a single page that displays the Windows identity used to run the application.

To create the test application

1. Create a new Web application in Visual Studio .NET called TestCustomPool.
2. Add the following code in the Default.aspx page load event handler.

 [Copy Code](#)

```
using System.Security.Principal;
...
WindowsIdentity id = WindowsIdentity.GetCurrent();
Response.Write("<b>Windows Identity Check</b><br>");
Response.Write("Name: " + id.Name + "<br>");
```

3. Compile the test application and run it. Note the output, which shows that the application currently runs under the default Network Service account.
The browser should display the following text:

 [Copy Code](#)

```
Windows Identity Check
```

Name: NT AUTHORITY\NETWORK SERVICE

Step 4. Create an Application Pool with a Custom Identity

In this step, you create a new application pool to run your ASP.NET application, and you configure it to use the custom service that you created earlier.

To create an application pool that runs using a custom service account

1. Start Internet Information Services (IIS) Manager.
2. In the left pane, expand the local computer and then expand Application Pools.
3. Right-click the Application Pools node, click New, and then click Application Pool.
4. In the Add New Application Pool dialog box, type TestPool in the Application Pool ID text box. Leave the Use default settings for new application pool option selected, and click OK. This creates a new application pool called TestPool.
5. Right-click the new application pool. and click Properties.
6. Click the Identity tab.
7. In the Application pool identity section, click Configurable.
8. Type CustomASP in the User name text box.
9. Type the password for the CustomASP account in the Password text box, and click Apply.
10. The Confirm Password dialog box appears. Type the password again, click OK, and then click OK again.

Step 5. Configure Your Application to Run in the New Application Pool

In this step, you configure your test ASP.NET application to run in the new application pool. This ensures that it runs using the custom service account identity.

1. Return to the Internet Information Services (IIS) Manager.
2. Locate your test application, TestCustomPool, in the left pane of the IIS Manager console.
3. Right-click TestCustomPool and click Properties.
4. On the Directory tab in the Properties dialog box, in the Application Settings section, select TestPool from the Application pool list, and then click OK.

Step 6. Test the Custom Service Account

Browse to the Default.aspx page of your test application. It should display the name of your custom service account, confirming that your application is running using this identity. The browser should display the following:

 [Copy Code](#)

```
Windows Identity Check  
Name: <ServerName>\CustomASP
```

Where *<ServerName>* is the name of your server.

Custom Account vs. Network Service

If you need your application to use a specific identity to access resources, you have two main choices. You can use a custom application pool identity, or run using the default Network Service identity and then call the LogonUser API to create a Windows identity that you can then impersonate in the specific methods that require an alternate identity. There are advantages and disadvantages to both approaches.

Using a Custom Account

With this approach, you run your application in an application pool configured to run using a specific Windows identity.

Advantages

Using a custom account has the following advantages:

- | The account credentials are stored in the IIS metabase, which is readable only by the SYSTEM account and members of the Administrators group.
- | Your application does not need to manage thread security context, which can be difficult to do correctly. Mistakes can lead to handle leaks and lost impersonation tokens because of asynchronous thread switches.

Disadvantages

Using a custom account has the following disadvantages:

- | When you change the identity of your application pool to use a domain account instead of a machine account, you lose the ability to perform Kerberos authentication until a domain administrator runs the Setspn utility to create a service principal name (SPN) for the domain account. If you have multiple applications on the same server that use separate domain identities and they need to use Kerberos authentication, then you need to use separate Domain Name System (DNS) names for each application.
- | In addition to adding the custom account to the IIS_WPG group, it is likely that you will need to configure additional file system ACLs for the custom account.
- | You have to manage account lifetimes and credentials on each Web server in a farm.
- | If the operation that your Web application needs to perform using a custom identity requires extended privileges, you need to run the entire Web application with these privileges, and that means that you have a great deal of code to run.

Using Network Service

With this approach, you run your application using the least privileged Network Service account. However, when your application needs to access resources or perform operations using a specific identity, you call the LogonUser API to create a new Windows identity. You then impersonate that identity only in those methods that require it.

Advantages

Using the Network Service account has the following advantages:

- | You can discretely limit elevation of privilege to those parts of the application that need it. If an attacker compromises the application, the attacker needs to do more work to exploit the extended privileges.
- | No reconfiguration of file system ACLs are required.
- | Account credential storage could be centrally located; for example, in a database with restricted access to the machine identity or similar, or in a Web.config file protected using shared RSA keys. This provides an easier Web farm rollout if the farm belongs to a domain (ideally, this should be a restricted domain in the perimeter network), and domain accounts are used when you call the LogonUser API.

Disadvantages

Using the Network Service account has the following disadvantages:

- | This approach leads to additional complexity in managing thread security context. This is especially difficult if your application uses asynchronous causalities or has many calls from different places.
- | Credentials must be stored and read by your application. Therefore, an attacker who compromises your application can eventually obtain those credentials.
- | LogonUser should be avoided on Windows 2000, because calls to the LogonUser API require

that you grant your process identity the Act as part of the operating system powerful user right. This user right is not required to call LogonUser on Windows Server 2003.

For more information about impersonating in ASP.NET Web applications, see [How To: Use Impersonation and Delegation in ASP.NET 2.0](http://msdn2.microsoft.com/en-us/library/ms998351.aspx) [<http://msdn2.microsoft.com/en-us/library/ms998351.aspx>] .

Additional Considerations

Other issues to consider when creating a service account to run an ASP.NET application include:

- | Creating service principal names (SPNs) for domain accounts
- | Using IIS 5.0 isolation mode

Creating Service Principal Names (SPNs) for Domain Accounts

When you switch from using a machine account, such as Network Service, to a domain account and if your application uses Kerberos authentication to authenticate its clients, Kerberos authentication will stop working unless you have a service principal name for the domain account registered in Microsoft® Active Directory® directory service.

To create an SPN for a domain account

1. Install the Windows Server 2003 tools from the Windows Server 2003 CD.
2. From a command prompt, run the Setspn tool as follows:
setspn -A HTTP/webservername domain\customAccountName
setspn -A HTTP/webservername.fullyqualifieddomainname
domain\customAccountName

The tool creates an SPN for the custom domain account (domain\customAccountName) and associates the account with the HTTP service on the specified Web server. By running the command twice as shown above you can associate the account with the NetBIOS server name and the fully qualified domain name of the server. This ensures that the SPN is established correctly even if your environment does not consistently use fully qualified domain names.

Note You cannot have multiple Web applications with the same host name if you want them to have multiple identities. This is an HTTP limitation, not a Kerberos limitation. The workaround is to have multiple DNS names for the same host, and start the URLs for each Web application with a different DNS name. For example, you would use <http://app1> and <http://app2> rather than <http://site/app1> and <http://site/app2>.

Using IIS 5.0 Isolation Mode

If you configure your IIS 6.0 server to run in IIS 5.0 isolation mode, then ASP.NET applications run using account credentials defined on the <processModel> element in the Machine.config file. With this configuration, ASP.NET applications run in a shared worker process called Aspnet_wp.exe, and they do not use IIS 6.0 application pools. If you change the account credentials on the <processModel> element, you cause all ASP.NET applications on the server to run under the specified account.

For more information about how to modify the account credentials using the <processModel> element, see [How To: Create a Custom Account to Run ASP.NET](#).

Additional Resources

- | [How To: Use Impersonation and Delegation in ASP.NET 2.0](http://msdn2.microsoft.com/en-us/library/ms998351.aspx) [<http://msdn2.microsoft.com/en-us/library/ms998351.aspx>]

Feedback

Provide feedback by using either a Wiki or e-mail:

- | Wiki. Security Guidance Feedback page:
http://channel9.msdn.com/wiki/default.aspx/Channel9_SecurityGuidanceFeedback
[<http://channel9.msdn.com/wiki/default.aspx/channel9.securityguidancefeedback>]
- | E-mail. Send e-mail to secguide@microsoft.com.

We are particularly interested in feedback regarding the following:

- | Technical issues specific to recommendations
- | Usefulness and usability issues

Technical Support

Technical support for the Microsoft products and technologies referenced in this guidance is provided by Microsoft Support Services. For support information, please visit the Microsoft Support Web site at <http://support.microsoft.com> [<http://support.microsoft.com/>] .

Community and Newsgroups

Community support is provided in the forums and newsgroups:

- | MSDN Newsgroups: <http://msdn.microsoft.com/newsgroups/default.asp>
- | ASP.NET Forums: <http://forums.asp.net> [<http://forums.asp.net/>]

To get the most benefit, find the newsgroup that corresponds to your technology or problem. For example, if you have a problem with ASP.NET security features, you should use the ASP.NET Security forum.

Contributors and Reviewers

- | External Contributors and Reviewers: Jason Taylor, Security Innovation; Rudolph Araujo, Foundstone Professional Services; Manoranjan M Paul
- | Microsoft Services and PSS Contributors and Reviewers: Adam Semel, Nobuyuki Akama, Tom Christian, Wade Mascia
- | Microsoft Product Group Contributors and Reviewers: Erik Olson, Stefan Schackow
- | Test team: Larry Brader, Microsoft Corporation; Nadupalli Venkata Surya Sateesh, Sivanthapatham Shanmugasundaram, Infosys Technologies Ltd.
- | Edit team: Nelly Delgado, Microsoft Corporation; Tina Burden McGrayne, Linda Werner & Associates, Inc.
- | Release Management: Sanjeev Garg, Microsoft Corporation



patterns & practices
proven practices for predictable results

[<http://msdn.microsoft.com/practices>]