How To: Use Code Access Security in ASP.NET 2.0

How To: Use Code Access Security in ASP.NET 2.0

[http://msdn.microsoft.com/practices/default.aspx]

patterns & practices Developer Center [http://msdn.microsoft.com/practices/default.aspx]

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

Microsoft Corporation

August 2005

Applies To

- ASP.NET version 2.0

Summary

This How To shows you how to select an appropriate trust level for your application, and where necessary, how to create a custom ASP.NET code access security policy file to define a custom trust level. You can use different code access security trust levels to incrementally limit your exposure to security attacks and provide extra degrees of application isolation. Code access security allows you to constrain your Web application by restricting the types of resources it can access and the types of operations it can perform. Code access security is particularly important when your application does not trust other applications that run in a hosted or shared server environment. For example, you can use it to ensure that applications cannot read each other's private data and cannot access critical shared system resources.

Contents

Objectives

- Learn how to use code access security in ASP.NET Web applications.
- Learn what has changed with partial trust Web applications in ASP.NET version 2.0.
- Choose the most appropriate trust level for your application.
- Use different code access security trust levels to incrementally limit your exposure to security attacks and provide extra degrees of application isolation.

Overview

By default, ASP.NET version 1.1 and version 2.0 Web applications and Web services run with Full trust. When an application runs with Full trust, code access security places no restrictions on the resources and operations it can access, and resource access is based solely on operating system security and Windows access control lists (ACLs).

To protect your ASP.NET application, you can use code access security to restrict the resources the application can access and the privileged operations it can perform. You do this by configuring the <**trust**> element in either the machine-level Web.config file or the application's Web.config file and setting it to one of the predefined trust levels, as shown here.

```
<trust level="Full|High|Medium|Low|Minimal" />
```

The <**trust**> element supports a number of predefined trust levels. Each level in succession provides a more restrictive environment (with fewer code access security permissions) in which to run your application.

For situations where your application requires a set of code access security permissions that do not exactly match one of the predefined trust levels, you need to evaluate whether it makes sense to create a custom trust level or whether you should target the higher trust level. The best choice depends on how close the existing trust level is to meeting the permission requirements of your application.

What's New in 2.0

The main differences between ASP.NET version 1.1 and ASP.NET version 2.0 for the trust levels are the following:

- In ADO.NET 1.1, you could only use the SQL Server .NET data provider from partial trust applications because this provider was the only one that did not demand full trust. In ADO.NET 2.0, the Oracle .NET data provider, the OLE DB .NET data provider, and the ODBC .NET data provider no longer demand full trust. This allows you to access SQL Server and other databases from partial trust applications. To use the other providers from a partial trust Web application, you need to customize policy and grant the appropriate permission: for example, **OleDbPermission**, **OraclePermission**, or **OdbcPermission**.

  Note Medium trust ASP.NET policy only grants the **SqlClientPermission** required by the SQL Server .NET data provider. This means you can use the SQL Server data provider from medium trust applications without policy modification.

- In ASP.NET version 2.0, **SmtpPermission** is available at Full, High, and Medium trust levels. This allows applications to send e-mail.

Summary of Steps

To use code access security in your ASP.NET applications:

- **Step 1. Identify the permissions your application needs**.
- **Step 2. Choose an appropriate trust level**.
- **Step 3. Configure your ASP.NET application**.
- **Step 4. Optionally create a custom trust level**.

Step 1. Identify the Permissions Your Application Needs

Identify the precise set of code access security permissions that your application requires. You can do this by manually reviewing your code or by using the Permissions Calculator tool (Permcalc.exe).

Manually Reviewing Code to Identity Permission Requirements

To manually identify permission requirements, you need to analyze your code and determine the types of resources it accesses, the kind of resource access it requires (such as read/write), and the privileged operations it performs.

The main difficulty with trying to manually identify permission requirements occurs if your code calls other assemblies, such as third-party assemblies or system assemblies. Identifying their permission requirements can be very difficult. This is where the PermCalc tool can help.

For more information about the permissions you need to be able to access specific resources types, see the "Resource Access Permissions Summary" section later in this document.

For more information about privileged operations and the permissions you need to be able to perform them, see the "Privileged Operation Permissions Summary" section later in this document.

Using the Permissions Calculator Tool

If you use separate assemblies for your application's business and data access logic and place them either in your application's \bin directory or the global assembly cache, you can run PermCalc on those assemblies. You cannot run PermCalc directly on .aspx Web pages.

To see the permission requirements of an assembly

1. Run the following command from a Microsoft Visual Studio .NET 2005 command window.

   permcalc -Show <***assemblyName***>

   The following is sample output from this command.

```
<?xml version="1.0" ?>
<Assembly>
  <Namespace Name="ClassLibrary1">
      <Type Name="Class1">
       <Method Sig="instance void test()" />
       <Method Sig="instance void .ctor()">
         <Demand>
          <PermissionSet version="1"
class="System.Security.PermissionSet">
           <IPermission version="1"
class="System.Security.Permissions.RegistryPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Read="true" />
           <IPermission version="1" class="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Unrestricted="true" />
          </PermissionSet>
       </Demand>
       <Sandbox>
        <PermissionSet version="1" class="System.Security.PermissionSet">
         <IPermission version="1" class="System.Security.Permissions.RegistryPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Read="true" />
         <IPermission version="1" class="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Unrestricted="true" />
        </PermissionSet>
      </Sandbox>
    </Method>
    </Type>
  </Namespace>
</Assembly>
```

2. Examine the permissions listed in the <**Demand**> element. These represent the permissions that the assembly needs. In this case, the assembly needs **RegistryPermission** and **FileIOPermission**.

```
<IPermission version="1" class="System.Security.Permissions.RegistryPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561
<IPermission version="1"
class="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Unrestricted="true" />
```

   You can pass a number of assemblies to the PermCalc tool at the same time, as shown in the following command example.

   permcalc -Show <assembly1> <assembly2> <assembly3>

   Note If any resource is accessed directly from an .aspx page, you will need to manually calculate the permission requirements for that resource. PermCalc works only with assemblies.

Step 2. Choose an Appropriate Trust Level

Try to evaluate whether the permissions required for your application match those provided by any of the standard trust levels.

Trust Levels Summary

The capabilities available to applications running at the various trust levels are summarized in Table 1.

Table 1. Trust Levels and Their Key Capabilities and Restrictions

Trust Level	Key Capabilities and Restrictions
Full	No restrictions imposed by code access security.
High	No unmanaged code. No enterprise services. Can access Microsoft SQL Server and other OLE DB data sources. Can send e-mail by using SMTP servers. Very limited reflection permissions. No ability to invoke code by using reflection. A broad set of other framework features are available. Applications have full access to the file system and to sockets.
Medium	Permissions are limited to what the application can access within the directory structure of the application. No file access is permitted outside of the application's virtual directory hierarchy. Can access SQL Server. Can send e-mail by using SMTP servers. Limited rights to certain common environment variables. No reflection permissions whatsoever. No sockets permission. To access Web resources, you must explicitly add endpoint URLs□either in the **originUrl** attribute of the <**trust**> element or inside the policy file.
Low	Intended to model the concept of a read-only application with no network connectivity. Read only access for file I/O within the application's virtual directory structure.

Minimal	Execute only. No ability to change the **IPrincipal** on a thread or on the **HttpContext**.

If your application calls unmanaged code, it must run with Full trust. Even the least restrictive partial trust level, High, does not permit calls to unmanaged code.

To choose an appropriate trust level

1. Examine each trust level, beginning with High trust.
2. Look inside the High trust policy file, web_HighTrust.config.
3. If your application requires fewer code access security permissions than those provided by the High trust level, move on to consider Medium trust.
4. Repeat the process, moving from Medium to Low to Minimal, and keep evaluating the partial trust levels until you reach an exact match to your application's requirements or until your application's required permissions slightly exceed a partial trust level.

This process will help you to identify a trust level that matches your application's code access security permission requirements as closely as possible but does not grant permissions that your application does not need.

Step 3. Configure Your ASP.NET Application

You can configure your ASP.NET application to use a standard trust level either in your application's Web.config file or in the machine-level Web.config file. Machine-level configuration affects all the ASP.NET Web applications and Web services hosted on that server. Application-level configuration affects only a specific application.

For machine-level configuration, you make changes to the Web.config file located in the following directory: %windir%\Microsoft.NET\Framework\{version}\CONFIG\.

For application-level configuration, you make changes to the Web.config file located in your application's virtual directory root.

The following example shows how to configure your application for Medium trust.

```
...
<system.web>
  ...
  <trust level="Medium" originUrl="" />
  ...
</system.web>
...
```

Step 4. Optionally Create a Custom Trust Level

You may require a custom trust level if none of the predefined trust levels provide the exact set of permissions that your application needs to run. For example, you may find that your application needs more permissions than Medium trust provides but fewer permissions than High trust provides.

Customizing Trust Level Policy

With this approach you do the following:

1. Copy one of the existing trust level policy files to create a custom policy file.
2. Add the required permissions to the custom policy file.
3. Configure your application to run using the custom policy.

> **Note** You are unlikely to be able to use this approach to customize the trust level in hosted environments, where policy restrictions are likely to be rigid.

If you can customize policy, this is the preferred option because it does not require any development effort.

In the following example, the application requires all the permissions provided by Medium trust. In addition, it needs unrestricted access to the Windows registry. While High trust provides unrestricted registry access, it also provides permissions that you may not want to grant to your application, such as unrestricted file I/O access. This is the type of situation where you should consider creating a custom trust level.

To create the custom trust level configuration file

1. Copy the Medium trust policy file, web_MediumTrust.config, which is located in the %windir%\Microsoft.NET\Framework\{version}\CONFIG\ directory to a file named Web_CustomTrust.config in the same directory.
2. Add the **RegistryPermission** security class definition to the <**SecurityClass**> section in the Web_CustomTrust.Config file, as shown in the following example.
```
<SecurityClass Name="RegistryPermission"
 Description="System.Security.Permissions.RegistryPermission,
mscorlib, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"/>
```
3. Add the unrestricted **RegistryPermission** to the "ASP.Net" named permission set.
```
<PermissionSet
    class="NamedPermissionSet"
    version="1"
    Name="ASP.Net">
  ...
     <IPermission
         class="RegistryPermission"
         version="1"
         Unrestricted="true" />
     ...
  </PermissionSet>
```
4. Modify the default Web.config file in the %windir%\Microsoft.NET\Framework\{version}\CONFIG\ directory to add the custom trust level that references the custom trust configuration file you have created.
5. Add a new <**trustLevel**> element to the <**securityPolicy**> section of the Web.config file to define a new level called **"Custom"** and to associate it with the custom policy file.
```
<location allowOverride="true">
    <system.web>
      <securityPolicy>
       <trustLevel name="Full" policyFile="internal" />
       <trustLevel name="High" policyFile="web_hightrust.config" />
       <trustLevel name="Medium" policyFile="web_mediumtrust.config" />
       <trustLevel name="Low" policyFile="web_lowtrust.config" />
       <trustLevel name="Minimal" policyFile="web_minimaltrust.config" />
       <trustLevel name="Custom" policyFile="web_CustomTrust.config" />
      </securityPolicy>
      <trust level="Full" originUrl="" />
     </system.web>
  </location>
```

The preceding changes apply to the root Web.config file in the framework CONFIG folder. This applies defaults to all Web applications on the current server.

To configure the trust level for your specific application

1. Copy the web_CustomTrust.config file to your application's virtual directory.
2. Instead of referring to the Web_CustomTrust.config from the machine-level Web.config file, refer to it directly from your application's Web.config, as shown in the following example.
```
...
<location allowOverride="true">
   <system.web>
      <securityPolicy>
        <trustLevel name="Custom" policyFile="web_CustomTrust.config" />
      </securityPolicy>
    <trust level="Custom" originUrl="" />
  </system.web>
```

```
    </location>
     ...
```

Locking the Trust Level

If a Web server administrator wants to use code access security to ensure application isolation and restrict access to system-level resources, the administrator must be able to define security policy at the machine level and prevent individual applications from overriding it.

Application service providers or anyone responsible for running multiple Web applications on the same server should lock the trust level for all Web applications.

To do this, enclose the <**trust**> element in the machine-level Web.config file in a <**location**> tag, and set the **allowOverride** attribute to **false**, as shown in the following example.

```
...
<location allowOverride="false">
   <system.web>
     ...
    <trust level="Medium" originUrl=""/>
    ...
   </system.web>
...
</location>
...
```

Wrapping Privileged Code

To avoid running with unnecessary permissions, you can either create a custom trust level that provides your exact set of permission requirements or you can wrap your privileged resource access code and grant Full trust to the wrapper assembly. The first approach is easier and involves no development effort. However, if you do not want all the pages in your ASP.NET application to have a particular extended permission granted to them, you can use a wrapper assembly to control access to your resources. With this approach, you separate your privileged code into a separate wrapper assembly.

To wrap privileged code

1. Identify the precise set of extra permissions beyond those provided by Medium trust that your application needs.
2. Place your application's resource access code in a wrapper assembly.
3. Grant the wrapper assembly (not the Web application) Full trust by strong naming it and installing it into the global assembly cache.
4. Add the **AllowPartiallyTrustedCallersAttribute** to the wrapper assembly to allow it to be called by your partial-trust Web application.
5. In the wrapper assembly, assert the permissions that the resource access code needs by calling the **Assert** method of the **CodeAccessPermission** class, perform the privileged operation (or access the privileged resource), and then call the **RevertAssert** method.

> **Note** If you want to prevent any other partially trusted code from accessing the resources and operations exposed by the wrapper assembly, before calling the **Assert** method, demand a custom permission that is only granted to your Web application.

Resource Access Permissions Summary

Table 2 shows the code access security permissions that are required to access various resource types.

Table 2: Resources and Associated Code Access Security Permission Requirements

Resources Accessed	Required Permissions
DPAPI encryption	DataProtectionPermission
DNS directory	DnsPermission
Environment variables	EnvironmentPermission
Event log	EventLogPermission
File dialog	FileDialogPermission
File system	FileIOPermission
Isolated file storage	IsolatedStoragePermission
Key containers	KeyContainerPermission
Message queues	MessageQueuePermission
Network information and traffic statistics	NetworkInformationPermission
OLE DB data sources	OleDbPermission
Performance counters	PerformanceCounterPermission
Printers	PrintingPermission
Reflection	ReflectionPermission
Registry	RegistryPermission
Security	SecurityPermission
SMTP servers	SmtpPermission
Sockets	SocketsPermission
SQL Server notifications	SqlNotificationPermission
SQL Server	SqlClientPermission
Stores containing X.509 certificates	StorePermission
User interfaces and clipboard	UIPermission
Web services (and other HTTP Internet resources)	WebPermission

Privileged Operation Permissions Summary

Table 3 shows the code access security permissions that are required to perform privileged operations.

Table 3: Privileged Operations and Associated Code Access Security Permission Requirements

Privileged Operation	Required Permissions
Creating and controlling application domains	SecurityPermission with SecurityPermissionFlag.ControlAppDomain
Specifying policy application domains	SecurityPermission with SecurityPermissionFlag.ControlDomainPolicy
Asserting security permissions	SecurityPermission with SecurityPermissionFlag.Assertion
Creating and manipulating evidence	SecurityPermission with SecurityPermissionFlag.ControlEvidence
Creating and manipulating principal objects	SecurityPermission with SecurityPermissionFlag.ControlPrincipal
Configuring types and channels remoting	SecurityPermission with SecurityPermissionFlag.RemotingConfiguration
Manipulating security policy	SecurityPermission with SecurityPermissionFlag.ControlPolicy

Serialization	SecurityPermission with SecurityPermissionFlag.SerializationFormatter
Threading operations	SecurityPermission with SecurityPermissionFlag.ControlThread
Reflection	ReflectionPermission
Calling unmanaged code	UnmanagedCodePermission
Call DPAPI to encrypt and decrypt data	DataProtectionPermission

ASP.NET Trust Level Summary

Table 4 shows the code access security permissions that are provided by each level. Permissions in ASP.NET 2.0 are highlighted with an asterisk. Note that the main difference between ASP.NET 1.1 and ASP.NET 2.0 permissions is that Medium trust ASP.NET 2.0 applications can access SQL Server databases because the **SqlClientPermission** is available at Medium trust and the ADO.NET 2.0 .NET Framework Data Provider for SQL Server no longer demands Full trust.

Table 4: Default ASP.NET Policy Permissions and Trust Levels

Permission and State	High	Medium	Low	Minimal
AspNetHostingPermission Level	High	Medium	Low	Minimal
DnsPermission Unrestricted	√	√		
EnvironmentPermission Unrestricted Read Write	√	TEMP; TMP; USERNAME; OS; COMPUTERNAME		
EventLogPermission				
FileIOPermission Unrestricted Read Write Append PathDiscovery	√	 $AppDir$ $AppDir$ $AppDir$ $AppDir$	 $AppDir$ $AppDir$	
IsolatedStorageFilePermission Unrestricted AssemblyIsolationByUser Unrestricted UserQuota	√	 √ √	 √ 1 MB (can vary with site)	
OleDbClientPermission Unrestricted				
PrintingPermission Unrestricted DefaultPrinting	√	√		
ReflectionPermission Unrestricted ReflectionEmit	√			
RegistryPermission Unrestricted	√			
SecurityPermission Unrestricted Assertion Execution ControlThread ControlPrinicipal RemotingConfiguration	 √ √ √ √ √	 √ √ √ √ √	 √	 √
SocketPermission Unrestricted	√			
SmtpPermission* Access		Connect		
SqlClientPermission Unrestricted	√	√		
SqlNotificationPermission* Unrestricted		√		
WebPermission Unrestructed ConnectAccess	√	 $OriginHost$		

Additional Resources

- How To: Use Medium Trust in ASP.NET 2.0 [http://msdn.microsoft.com/en-us/library/ms998341.aspx]

Feedback

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

- **Wiki.** Security Guidance Feedback page: http://channel9.msdn.com/wiki/securityguidancefeedback/ [http://channel9.msdn.com/wiki/securityguidancefeedback/]
- **E-mail**. Send e-mail to mailto: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 product support information, please visit the Microsoft Product Support [http://support.microsoft.com/default.aspx] Web site.

Community and Newsgroups

Community support is provided in the forums and newsgroups:

- MSDN Newsgroups [http://www.microsoft.com/communities/newsgroups/default.mspx]
- ASP.NET Forums [http://forums.asp.net/default.aspx]

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 would use the ASP.NET Security forum.

Contributors and Reviewers

- **External Contributors and Reviewers:** Jason Taylor, Security Innovation; Rudolph Araujo, Foundstone Professional Services
- **Microsoft Consulting Services and PSS Contributors and Reviewers:** Adam Semel, Tom Christian, Wade Mascia
- **Microsoft Product Group Contributors and Reviewers:** 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, TinaTech Inc.
- **Release Management:** Sanjeev Garg, Microsoft Corporation

[http://msdn.microsoft.com/practices/default.aspx]