| | |
|---|---|
| Issue | VSM November 2002 |
| Section | Black Belt column |
| Main file name | VS0211BBt2.rtf |
| Listing file name | -- |
| Sidebar file name | -- |
| Table file name | VS0211BBtb1.rtf |
| Screen capture file names | VS0211BBfX.bmp |
| Infographic/illustration file names | VS0211BBf1,2.bmp |
| Photos or book scans | ISBN 0596003471 |
| Special instructions for Art dept. | |
| Editor | LT |
| Status | TE'd3 |
| Spellchecked (set Language to English U.S.) | * |
| PM review | |
| Character count | 15,093 + 1,162 online table |
| Package length | 3.5 (I think, due to no inline code/listings) |
| ToC blurb | Learn how to safely grant assemblies permissions to perform operations with external entities such as the file system, registry, UIs, and more. |

| **ONLINE SLUGS** | |
|---|---|
| Name of Magazine | VSM November 2002 |
| Name of feature/column/department | Black Belt column |
| 180-character blurb | Learn how to safely grant assemblies permissions to perform operations with external entities such as the file system, registry, UIs, and more. |
| 90-character blurb | Learn how to safely grant assemblies permissions to perform operations with external entities. |
| 90-character blurb describing download | NA |
| Locator+ code for article | VS0211BB_T |
| Photo (for columnists) location | On file |

**TITLE TAG & METATAGS**

<title>**Visual Studio Magazine – Black Belt -** Secure Access to Your .NET Code

# Configure .NET Code-Access Security

</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#, Visual Basic .NET**">
<meta name="Keywords" content="**.NET, C#, Visual Basic .NET, security permission, security evidence, security policy, permission sets, evidence, security, security permission stack walk, custom permission set, code group**">
[[**Please check these and add/subtract as you see fit**.]]
<meta name="DESCRIPTION" content=" Learn how to grant assemblies permissions to perform operations with external entities such as the file system, registry, UIs, and more.**">**

<meta name="Author" content=" Juval Löwy">
<meta name="Issue" content="**November 2002**">
<!-- End META Tags -->

**RESOURCES TAG & METATAGS**

<title>**Visual Studio Magazine – Black Belt -** Secure Access By .NET Code-**Resources**</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#, Visual Basic .NET**">
<meta name="Keywords" content="**.NET, C#, Visual Basic .NET, security permission, security evidence, security policy, permission sets, evidence, security, security permission stack walk, custom permission set, code group**">
<meta name="DESCRIPTION" content=" Additional information about .NET security provisions.">
<meta name="Author" content=" Juval Löwy">
<meta name="Issue" content="**November 2002**">
<!-- End META Tags -->

Overline:

B l a c k  B e l t

Byline:

By Juval Löwy

Technology Toolbox:
VB.NET
C#
SQL Server 2000
ASP.NET
XML
VB6

# Configure .NET Code-Access Security

Deck:

***Learn how to grant assemblies permissions to perform operations with external entities,***

# *and protect your code from abuse.*

.NET component-oriented security derives from an elegant concept: using an administration tool, you grant assemblies certain permissions to perform operations with external entities such as the file system, the registry, the user interfaces, and so on. Using the .NET security infrastructure, you can provide component-oriented security, something that was difficult in the unmanaged world. This article describes the core concepts and default policies, compatible with version 1.1 of .NET.

.NET provides multiple ways to identify which assembly to grant what permission, and what evidence the assembly should provide to establish its identity. At runtime, whenever an assembly tries to perform any protected operation or access a resource, .NET verifies that the assembly and its calling assemblies have the appropriate security permissions.

A security permission is an individual grant to perform a specific operation. Permissions have both type and scope. A file I/O permission's type differs from that of a user interface permission, because they control access to different types of resources.

A permission's scope can be narrow, wide, or unrestricted. For example, a file I/O permission can allow reading from a particular file. You can represent writing to the same file with different file I/O permissions. .NET defines 19 types of permission. These govern ~~all~~ the operations and recourses an application is likely to use, such as file I/O, UI, Web access, database access, network, reflection, and other types, including special security permissions.

You often need a set of permissions of various scopes and types for an assembly to function properly. .NET facilitates this by letting you create collections of individual permissions—called permission sets. You can also use existing ones, called named permission sets. .NET provides six of these named permission sets: Nothing, Execution, Internet, LocalIntranet, Everything, FullTrust, and SkipVerification (see Table 1 online; see Go Online box for details).

.NET grants permissions to assemblies based on their identities. A security evidence is some form of proof an assembly can provide to validate its identity. Assemblies substantiate their identities with origin- and assembly-based security evidence.

Origin-based evidence simply examines where the assembly is coming from, independent of content. Standard origin-based evidence consists of code coming from the application directory, a particular Web site, a URL, or a zone. A zone is the local machine, an intranet, the Internet, or an explicitly trusted or untrusted Internet site.

Assembly-based evidence *does* examine assembly content. It looks for a specific match with specified criteria. The standard assembly-based evidences are a specific strong name, publisher certificate, assembly hash, and All Code—a wild card satisfied by all code.

You use code groups to classify assemblies so .NET can determine which security permissions to grant. A code group binds a single permission set to a single evidence (see Figure 1). An assembly must satisfy a code group's evidence to win the permissions in the permission set associated with the code group.

Code groups aggregate into .NET security policies. The permissions granted by a policy to a given assembly is the union of all the individual permissions granted by the code groups in that policy whose evidence the assembly satisfies. For example, if an assembly satisfies the evidences of code groups A, B, and C, but not the evidences required by code groups D and E, it is granted only the union of permissions A, B, and C (see Figure 2).

**Compose Security Policies**

.NET lets you provide multiple security policies with different scopes. You can apply restrictive policies to specific cases, such as individual users or machines with limited privileges. Then apply more permissive policies to other machines and users in an organization.

One policy may grant an assembly permission denied by another policy. But because all policies must concur on the allowed permissions, the actual permissions .NET grants to an assembly is the intersection of all permissions granted by all the security policies.

There are four types (or levels) of security policies. An Enterprise policy affects all machines in the Enterprise. Each machine has a Machine policy specific to that machine. The User policy affects one user. And an Application Domain policy applies only to code running in a specific application domain.

You configure Application Domain policies only programmatically. Typically, you place more restrictive policies downstream, more liberal policies upstream. This allows overall flexibility with granular security policy, tight in some places and looser in others.

When .NET loads an assembly, it computes the permissions to grant that assembly. For each security policy, .NET aggregates the permissions from the code groups satisfied in that policy. Then .NET intersects the policies to find the combined overall collection of permission to grant the assembly. It calculates that set of permissions only once (per app domain). The calculated permission persists in memory as long as the assembly remains loaded.

Whenever an assembly calls a .NET framework class, that class may demand assurance from .NET that the assembly calling it has the security permission required to access it. For example, file I/O classes demand appropriate file I/O permission for that access. If the assembly lacks the appropriate security permission, .NET throws a security exception.

So far so good. But suppose a malicious assembly, which lacks permissions to access a class such as FileStream, calls a benign trusted assembly to do its dirty work. Well, the .NET architecture can thwart such attacks. Whenever a class demands security permission checks, .NET traverses the entire call stack, making sure every assembly up the call chain has the required permissions. This is called the *security permission stack walk*. And if .NET finds an assembly without permissions during the stack walk, .NET immediately aborts the stack walk and throws an exception. In future articles I'll show you how to demand security permission stack walks in your own classes.

.NET gives you two ways to configure a code access security policy: caspol.exe, a command-line utility, or the .NET Configuration tool, both with comparable capabilities. Use the .NET Configuration tool to configure security and export the security policies to other machines. Use caspol.exe to make dynamic changes during installation.

**Administer Your Security Policies**

The .NET admin tool has a Runtime Security Policy folder. Once expanded, the folder contains an item for each policy: Enterprise, Machine and User. Each policy item has subfolders, containing its code groups and permission sets (see Figure 3). Each security policy is stored in a dedicated XML-formatted security configuration file. The .NET Configuration tool merely provides a visual editor for those files.

The Permission Sets folders in all three policies contain the same set of named permission sets. Both the Enterprise and the User policies contain a single code group, All_Code, by default. This group uses the All Code evidence with the FullTrust permission set, so all assemblies are unrestricted.

The Machine policy has a single root code group, also called All_Code, that uses the All Code evidence with the Nothing permission set. By itself it grants nothing, and instead relies on the nested code groups to grant permissions.

The My_Computer_Zone code group uses the Zone evidence, with the zone set to My Computer, granting the FullTrust permission set. Consequently, all code coming from the local computer gets full trust. The My_Computer_Zone code group has two nested child code groups, Microsoft_Strong_Name and ECMA_Strong_Name (see Figure 3). These nested code groups use the Strong Name evidence (with the value set to the Microsoft public key) and ECMA public key respectively. Both these nested code groups use the FullTrust permission set, granting unrestricted access to any assembly originating from Microsoft or ECMA, regardless of its zone.

The LocalIntranet_Zone code group uses the zone evidence with a value set to the Local Intranet zone. The permission set is LocalIntranet. The problem is that the LocalIntranet permission set doesn't grant Web access or File I/O or Directory Services permission. So an assembly originating in the Intranet may not function properly if it requires access to its original site or its install directory.

To compensate, the LocalIntranet_Zone dode group contains two nested code groups (see Figure 3). The Intranet_Same_Site_Access code group lets code access its site of origin, and the Intranet_Same_Directory_Access code group lets code access its original install directory.

The Internet_Zone code group uses the zone evidence, with a zone set to Internet. The permission set used is Internet. The Internet_Zone code group has a child code group called Internet_Same_Site_Access (see Figure 3), allowing code coming from a site to connect to its site of origin.

The Restricted_Zone code group uses the zone evidence, with a zone set to Untrusted Sites and the permission set Nothing. The Trusted_Zone code group uses the zone evidence, with a zone set to Trusted Sites. The permission set used is the Internet permission set. The Trusted_Zone code group has a child code group called Trusted_Same_Site_Access granting code coming from a trusted site permission to connect to its site of origin.

**Build Custom Permission Sets**

You can define custom permission sets and compose granular permissions suitable for a particular need. Create a new permission set by right-clicking on the Permissions Sets folder and selecting New… to bring up the Create Permission set wizard. The first screen lets you name the new permission set and provide a description. Click Next >, to assign individual permissions to the new permission set (see Figure 4).

When you add a permission type from the left pane, a dedicated dialog for that type appears, allowing you to add individual permissions of that type. Suppose you need the new permission set to grant file I/O permissions to read the C drive, and full access to C:\temp. Select File I/O on the left pane of the dialog; click Add >> to bring the file I/O permission setting dialog.

The dialog has a grid where each line corresponds to a single file I/O permission. You can also grant unrestricted access to the file system. Configure the required setting and click OK to return to the previous dialog and click Finish. You can now use this permission set with any code group in the policy.

You can modify existing code groups or create new ones. Maybe you want to create a new code group that grants all assemblies signed with your organization's public key the Everything permission set. This can be handy when different teams use each other's assemblies across an intranet. The logical place for the new code group is in the Machine policy, under the My_Computer_Zone code group. Highlight the My_Computer_Zone code group, click on it and select New… from the context menu. In the Create Code Group dialog, name the new code group and click Next >.

In the "Choose a condition" type dialog go to the condition type textbox. Select Strong Name from the drop-down combo box (see Figure 5). This changes the lower part of the dialog to reflect the value of the requested strong key. Provide the public key value by importing it from an already-signed assembly. Click the Import… button to -browse to a signed assembly (either an EXE or a DLL) and select it. The wizard reads the public key from the assembly's manifest and populates the Public Key textbox.

Click Next > to proceed to the next dialog, where you need to assign a permission set to the new code group. You can use any existing permission set in the policy by selecting the set from the drop-down combo box. Select Everything, click Next >, and Finish in the next dialog. The new code group is now part of the policy.

**About the Author:**

Juval Löwy is a software architect and principal of IDesign, a .NET consulting and training company. He's also a Microsoft Regional Director, the .NET California Bay Area User Group Program committee chairman, and a conference speaker. This article derives from his upcoming book on .NET components (O'Reilly). Contact Juval at www.idesign.net

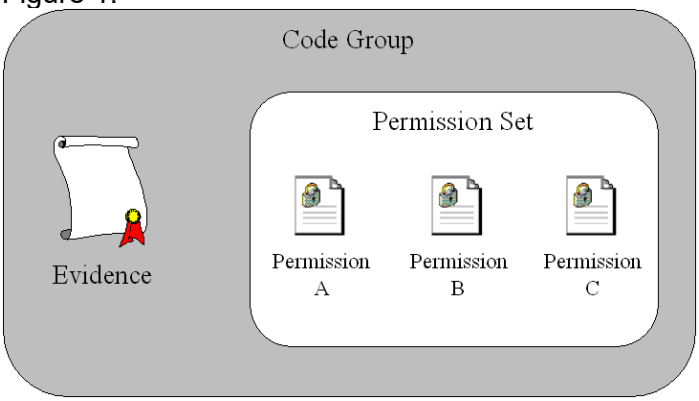
Pullquotes:

The permissions .NET grants to an assembly is the intersection of all permissions granted by all the security policies.

If .NET finds an assembly without permissions during the security permission stack walk, .NET immediately aborts the stack walk and throws an exception.

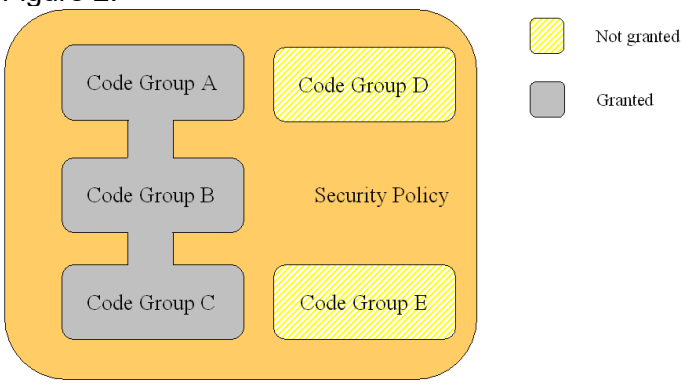
Figure Captions:

Figure 1:




**Bind Evidence to a Permission Set.**
.NET security is based on granting assemblies permissions to perform operations on the file system, the registry, the user interfaces, and other external entities. You use code groups to classify assemblies so .NET can determine which security permissions to grant. A code group binds a single permission set to a single evidence. When an assembly identity matched the evidence, the assembly is granted the permission in the permission set.

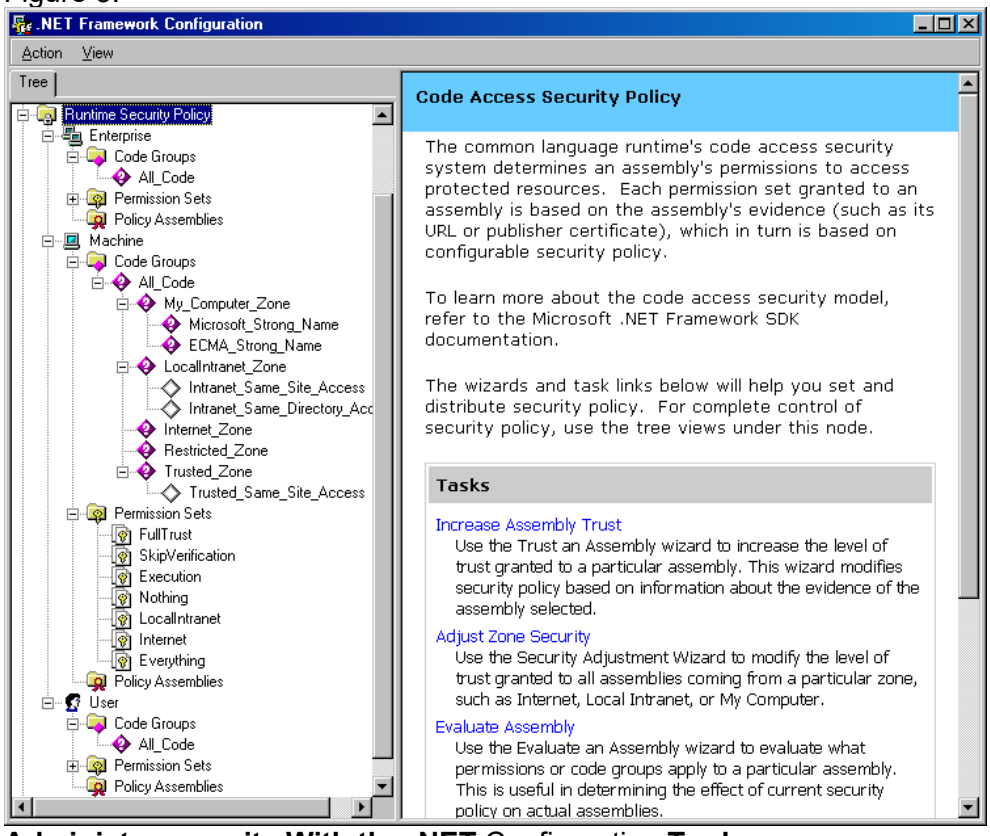
Figure 2:




**Implement Security With Policies.**

Code groups aggregate into .NET security policies. The permissions granted by a policy to a given assembly is the union of all the individual permissions granted by the code groups in that policy whose evidence the assembly satisfies. For example, if an assembly satisfies the evidences of code groups A, B, and C, but not the evidences required by code groups D and E, it is granted only the union of permissions A, B, and C

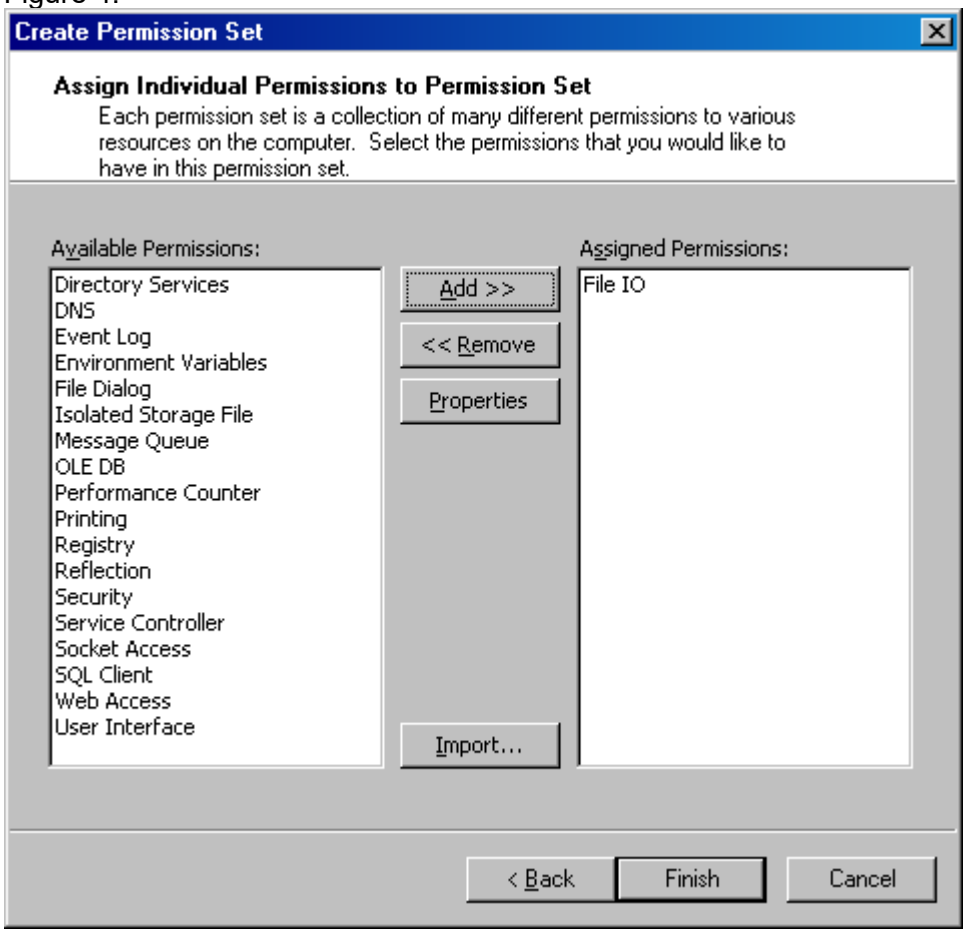
Figure 3:




**Administer security With the .NET** Configuration **Tool.**

The common language runtime's code access security system determines an assembly's permissions to access protected resources. Each permission set granted to an assembly is based on the assembly's evidence (such as its URL or strong name). . Each policy has a folder for permission sets and code groups, as you can see in the Runtime Security Policy folder.

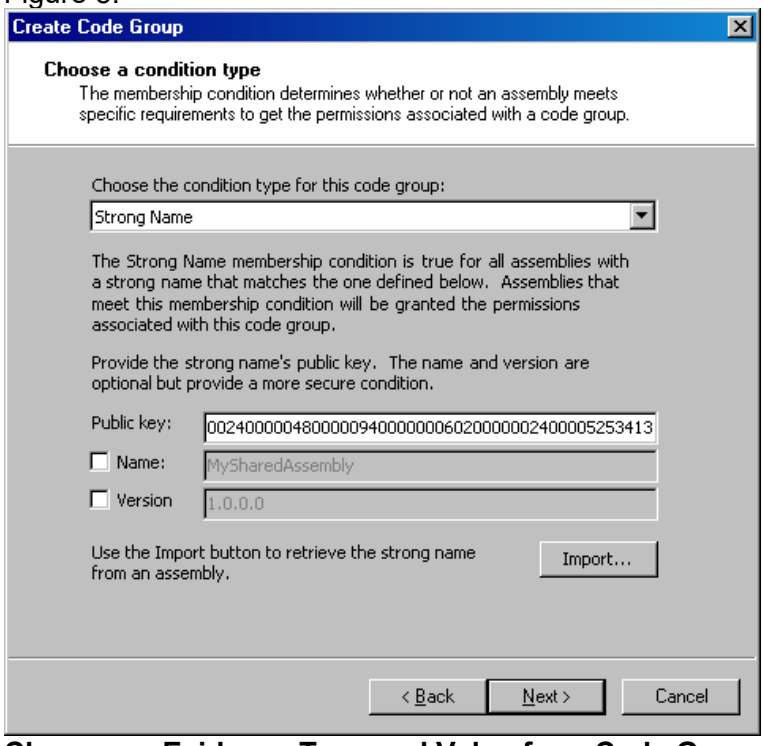
Figure 4:




**Create Custom Permission Sets.**

Each permission set is a collection of many different permissions to various resources on the computer. Select the permissions you'd like to have in this permission set. Each permission type has a dedicated permission configuration dialog, letting you set the permission value.

Figure 5:




**Choose an Evidence Type and Value for a Code Group.**

The membership condition determines whether or not an assembly meets specific requirements to get the permissions associated with a code group. Each evidence type has a dedicated dialog letting you specify a value, such as public key for a strong name evidence.