



INTERNET
SECURITY
SYSTEMS™

Authentication and Security Mechanisms in ASP.NET Web Applications

Introduction

Microsoft's .NET framework provides developers with the ability to build and deploy applications and services via the Web. These services can facilitate communication between clients and .NET application servers (such as database servers and so forth) through the use of XML queries issued by the client. This environment is attractive to developers because it is a language-neutral environment that can deliver content to end-users regardless of the platform in use.

ASP.NET is a part of the framework for developing .NET applications. .NET itself is a model of software development and delivery envisioned by Microsoft. Its goal is to provide the software, development platform, and backend infrastructure for applications that can be distributed across the Internet or other networks to a variety of devices such as Personal Computers (PCs) and Personal Digital Assistants (PDAs).

Although this technology provides powerful functionality to developers, these services can represent a potential threat to the security of a Web server. .NET applications and services can provide potential intruders with a new vector of attack, since many firewalls do not process HTTP traffic at a sufficient level to recognize malicious activity. Furthermore, these applications can possibly be used as a gateway for attackers to communicate with the .NET application servers.

This document discusses some of the security mechanisms and configuration options available to administrators to help secure these applications and reinforce the integrity of the Web server.

Basic Definitions

These basic terms are elements of .NET that are referred to throughout this document:

Common Language Runtime (CLR) – The runtime environment for .NET applications. CLR acts as a virtual machine and is capable of providing an additional layer of security on top of the core OS Security mechanisms.

Assemblies – Units of code that combine to form a .NET application. Each assembly can reside in different places on the computer (or network) and can have different security permissions applied to them.

The .NET Framework – The framework upon which .NET applications run. The .NET framework provides the ability for the administrator to configure security policies that can be used by the CLR when assemblies are being loaded.

Roles – Sets of permissions that may be granted to users by the administrator of a given computer. Roles provide a convenient method for administrators to specify what access different classes of users may be given for system resources.

Principals – Represent the identity of the user or agent in which the current thread will execute on behalf of. Principals are objects that contain an identity of a user and the roles that that user is assigned to. There are three types of principals — Windows Principals (for Windows users and their associated roles), Generic Principals (for users and roles that exist independently of the operating system's user database), and Custom Principals (for application developers to define for use with applications).

Mobile Code – Code to be invoked that may reside anywhere on the network.

Code Groups – A set of rules used by the .NET framework to categorize different assemblies. Each category has a set of rules defined that dictate what restrictions are applied to its constituent assemblies.

Evidence – The identification of a given assembly that is used to sort assemblies into code groups. Evidence is information pertaining to a particular assembly including where it originated from, the author, and if the assembly has been cryptographically signed.

Managed Code – .NET assemblies that require the CLR in order to be executed. Code that is unmanaged is native to the operating system and doesn't make use of the CLR.

The .NET Security Model

The .NET Security model is a complex, multi-layered, and highly configurable system. Since part of the purpose of the .NET framework is to allow for “mobile code” to be distributed to users on multiple platforms, security has become a major concern. Mobile code is an application or piece of software that is transmitted from a server to a local system (or other device) to be executed locally. This section provides an overview of several aspects of the .NET security model. The focus for this section is on the configuration of .NET, specifically the configuration options for securing deployment of Web applications. Configurations of the framework itself, such as defining policies and code groups, as well as security features inherent in the CLR are addressed later in this paper.

The Common Language Runtime

Web applications are executed within the CLR environment. This environment incorporates a rich security infrastructure that is highly configurable and can be tailored by administrators for various applications.

The CLR security model is completely abstracted from the operating systems' security mechanisms, which allows a developer or administrator to define resources that can be accessed regardless of a system's user accounts and permissions. This approach is a deliberate design consideration that allows the .NET runtime to authenticate users using a variety of user-defined methods. This allows authenticated transactions in a safe environment for a particular Web application without having specific user accounts defined on the system.

The CLR incorporates the notion of a principal, which is a user or identity that defines a set of roles and permissions enforced upon the running process. When a Microsoft Internet Information Server (IIS) receives a request for a Web application, IIS passes control to an ISAPI filter (ASPNET_ISAPI.DLL), which spawns a worker process (ASPNET_WP.EXE) to execute the application within the CLR. The worker process thread is assigned a system principal that will determine what external resources the application may access, such as files on the system. The CLR also has its own principal that is independent of the OS. The application will use this principal to determine whether certain resources within the application itself may be accessed or executed.

As an example, an application needs to access a data file. This file is only readable by the user 'BOB' that exists on the system. When this application executes, the OS assigns the worker process thread the principal BOB (or permissions from an administrative account, but using this account is bad practice). The application authenticates remote requests against a set of users that are particular to the application — the user information does not physically exist on the system. The application can set the principal that the CLR recognizes to the authenticated user name so that only certain authenticated users may access the data file.

Later, this paper discusses several configuration parameters that affect operating system permissions (specifically, the principal permissions that the application will run with when the application is executed), whereas others affect the CLR principals.

Web Application Security

There are several important features available for configuration for ASP.NET applications. Permissions for Web applications are regulated by the .NET Framework. Each Web application is comprised of a number of assemblies, all of which may have different security permissions granted to them by the CLR. The way these permissions are established is determined by the configuration of the .NET Framework on the server that is executing these applications.

When building Web applications, configuration options may provide additional security beyond default settings. These options provide authentication mechanisms and authorization control to allow for granular support over who may use these applications and what features of the application are available to that user. System-wide policies may also be put in place to enforce rules that will apply to all installed Web applications.

These options are configured by changing the configuration parameters in the Web applications' configuration file, which is located in the directory where the application is installed. This file, written in XML and named "web.config," defines a set of policies that will be applied to the application. The worker process will also recursively examine parent directories for the existence of web.config files. Lists of permissions will be built from policies supplied in each web.config file encountered in the order that they are read (from the root directory of the application up to the root directory of the Web server).

Finally, a global configuration file, machine.config, located in the %SYSTEMROOT%\Microsoft.NET\version)\CONFIG directory, defines policies that will be applied to all installed Web applications. Since this policy is read last, all previously read web.config files will install policies with a higher precedence than those contained in the global configuration file.

Authentication

Authentication refers to the method used by the server to verify the clients' identity. This feature provides methods to authenticate clients via a set of standardized and reusable methods that require little or no modification. The methods available to developers are:

None – This method does nothing.

Windows Authentication – Attempts to verify users by validating supplied credentials using authentication methods used by the Windows operating system. This includes NTLM (NT LanMan) and Kerberos (for systems running versions more current than Windows NT 4.0). Windows authentication is used by default if no method is explicitly selected.

IIS Authentication – Uses authentication methods provided by the IIS Web server.

Passport Authentication – Verifies users through the use of the Microsoft Passport authentication server.

Forms Authentication – Allows for application developers to provide a form for authenticating users in a standardized way. User accounts can be made specific to the application and stored in the web.config file.

Authorization

Authorization refers to *who* is allowed to access specific application resources, as opposed to authentication, which is concerned with verifying the user using one of the aforementioned mechanisms. The purpose of authorization is to provide an easy method to apply access controls to Web applications. Authorization can be performed in two different ways:

Users – A user or list of users can be specified that are explicitly allowed or denied from using the application.

Roles – A role or list of roles can be specified that are explicitly allowed or denied from using the application. Any user associated with the role will have the appropriate policy applied to them. Policies written with roles rather than users are preferable, as administration is simplified and doesn't need constant updating.

When a user makes a request to a .NET Web application, .NET examines the web.config file in the application's directory for authorization policies. .NET examines each preceding directory up to the virtual root of the Web directory, and appends any additional authorization rules to the list. Finally, rules residing in the machine.config file are appended to the list. .NET determines access to the specified resource by iterating through the rules in the order in which they were read and allows or denies the request based on the first rule that matches the authenticated user.

When using Windows authentication, authorization can be performed using file permissions. If an application attempts to access a file resource and does not have sufficient permissions, IIS attempts to authenticate the user to allow access to the resource.

Configuration Syntax

The discussion of some of the security features for ASP.NET applications also requires an examination of how these mechanisms are used. The following section presents the syntax for supplying the necessary configuration parameters to Web applications.

Operating System Related Parameters

The operating system controls what external resources the Web application may access by assigning a principal to the worker process thread. The access associated with the worker process is derived in one of two ways.

Impersonating accounts – IIS assigns a token to the process with credentials of the requestor (either IUSR_machine for anonymous requests or some other principal for authenticated requests).

Default privileges – Another way to assign access is to start a worker process with the default permissions specified by the operating system (usually the 'ASPNET' user).

Impersonating accounts

This tag causes the IIS Web server to hand off a token to the worker process defining the access privileges of the application. It is usually placed in the machine.config file or in the root web.config file for an application in the system.web section.

Syntax: `<identity impersonate="true|false" />`

Default value: false

Example:

```
<configuration>
  <system.web>
    <identity impersonate="true" />
  </system.web>
</configuration>
```

Default privileges

This tag sets the default identity in which the worker process will run as. This parameter exists in the machine.config file in the system.web section. Two accounts (SYSTEM and MACHINE), which may be supplied as username values, do not require a valid password. In these instances, the password field is set to "Autogenerate".

Syntax: `<processModel enabled="true|false" username="username" password="password"/>`

Default value: MACHINE

Example:

```
<configuration>
  <system.web>
    <processModel enabled="true" username="BOB"
password="bobspassword"/>
  </system.web>
</configuration>
```

CLR Related Parameters

The following tags are configuration parameters that are processed by the HttpModules (primarily UrlAuthorizationModule) and affect what the CLR permits to be executed or accessed.

Authentication tag

The authentication tag sets the authentication method for a given Web application. This tag is usually placed in the web.config file in the system.web section. Some of the authentication modes require nested tags that define their behavior. For example, the 'Forms' authentication mode has a nested forms tag shown in the following example.

Syntax: `<authentication mode="None|Forms|Windows|Passport">`

Default value: Windows

Example:

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx" />
    </authentication>
  </system.web>
</configuration>
```

Credentials

Sometimes Web applications authenticate using a custom mechanism or via the forms mechanism shown in the previous example. When this occurs, user credentials can be stored in the web.config file to authenticate users. The credentials tag specifies in what manner these credentials will be stored. Examine the user tag later in this section to see how it is used in conjunction with the credentials tag.

Syntax: `<credentials passwordFormat="Clear|SHA1|MD5"></credentials>`

Default value: none

User

The user tag is used in conjunction with the credentials tag described in the previous section. This tag describes login credentials in the format specified.

Syntax: `<user name="name" password="password" />`

Default value: none

Example:

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx">
        <credentials passwordFormat="Clear">
          <user name="bob" password="bobspassword" />
        </credentials>
      </forms>
    </authentication>
  </system.web>
</configuration>
```

Authorization tags

Three tags define authorization parameters related to Web applications. These tags are placed in either a web.config file or the machine.config file to apply default policies to all applications.

The first tag is the authorization tag, which defines the section where policies can be defined to allow or deny users or roles. These rules are checked in the order in which they are read.

Syntax: `<authorization> </authorization>`

Default value: none

The second and third tags are the 'allow' and 'deny' tags, which have the same syntax.

Syntax: `<allow|deny users|roles="(userlist|roleslist)" />`

These two tags can be used to apply an explicit allow or deny policy to either users or roles. The user or roles list can be in the form of a comma-delimited list of users or roles for which this policy will apply. The "?" character means "unauthenticated user" and the "*" wildcard means "anyone."

Example:

```
<configuration>
  <system.web>
    <authorization>
      <allow users="bob, john, paul" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

```

machine - WordPad
File Edit View Insert Format Help
<customErrors mode="RemoteOnly" />
<!--
  authentication Attributes:
  mode="[Windows|Forms|Passport|None]"
-->
<authentication mode="Windows">
  <!--
    forms Attributes:
    name="[cookie name]" - Name of the cookie used for Forms Authentication
    loginUrl="[url]" - Url to redirect client to for Authentication
    protection="[All|None|Encryption|Validation]" - Protection mode for data in cookie
    timeout="[seconds]" - Duration of time for cookie to be valid (reset on each request)
    path="/" - Sets the path for the cookie
  -->
  <forms name=".ASPXAUTH" loginUrl="login.aspx" protection="All" timeout="30" path="/">
    <!--
      credentials Attributes:
      passwordFormat="[Clear|SHA1|MD5]" - format of user password value stored in <user>
    -->
    <credentials passwordFormat="SHA1">
      <!-- <user name="UserName" password="password"/> -->
    </credentials>
  </forms>
  <!--
    passport Attributes:
    redirectUrl="[url]" - Specifies the page to redirect to, if the page requires authentication, and the user he
  -->
  <passport redirectUrl="internal" />
</authentication>
<!--
  identity Attributes:
  impersonate="[true|false]" - Impersonate Windows User
  userName="Windows user account to impersonate" | empty string implies impersonate the LOGON user specified by II
  password="password of above specified account" | empty string
-->
<identity impersonate="false" userName="" password="" />
<authorization>
  <!--
    allow/deny Attributes:
  -->
  </allow/deny>
-->
For Help, press F1

```

Figure 1. The machine.config file.

Recommended Configurations

This section lists some example configuration files for both Personal and Intranet .NET applications. These examples combine a number of features to serve several different types of applications.

Public .NET Application

This application is a typical public application in which anonymous users are permitted to access most resources, but are also able to create private accounts with personalized data. Applications that might be configured in this manner are Web mail applications and shopping carts.

Configuration

This application is stored in c:\application. Private data is stored in files in the C:\application\data directory. The forms authentication mechanism has been chosen since the application maintains its own user database. Additionally, administrator accounts are able to configure the application using scripts in C:\application\config.web.

C:\application\web.config

```

<configuration>
  <system.web>

    <!-- authentication section, use a forms based-login -->
    <authentication mode="Forms">
      <forms loginUrl="login.aspx">
        <credentials passwordFormat="SHA1">
          <user name="administrator" password=" ..." />
        </credentials>
      </forms>
    </authentication>
  </system.web>
</configuration>

```

```
        <user name="bob" password="..." />
        <user name="john" password="..." />
        <user name="paul" password="..." />
    </credentials>
</forms>
</authentication>

<!--we just want to deny anonymous users -->
<authorization>
    <deny user="?*" />
</authorization>

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

C:\application\data\web.config

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="users" />
      <deny roles="*" />
    </authorization>
  </system.web>
</configuration>
```

C:\application\config\web.config

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="admins" />
      <deny roles="*" />
    </authorization>
  </system.web>
</configuration>
```

Intranet .NET Application

An intranet application usually only grants access to employees. The most appropriate form of authentication is likely to be the Windows authentication method, where users are mapped to valid Windows accounts. The application is installed in C:\intranetapp.

Configuration

C:\intranetapp\web.config

```
<configuration>
  <system.web>
    <authentication mode="Windows" />

    <authorization>
      <allow roles="employees, managers">
      <deny users="*">
    </authorization>
  </system.web>
</configuration>
```

.NET Framework Security Mechanisms

Role-Based Security

Role-Based security refers to the security context under which an assembly is run. Assemblies have associated privilege levels that define what resources may be accessed at execution time. The framework establishes the privileges granted to a particular assembly when *authentication* takes place.

The application programmer defines rules related to roles in an XML configuration file, such as specifying policies which govern privileges the application has when it is invoked, as well as who is permitted to use it. This is achieved by specifying rules for both *authentication* and *authorization*. Once authenticated and authorized, an Identity is established and associated with the running assemblies principal. Identities associated with a principal may either be accounts and roles on the system, or custom identities that are created by the application programmer.

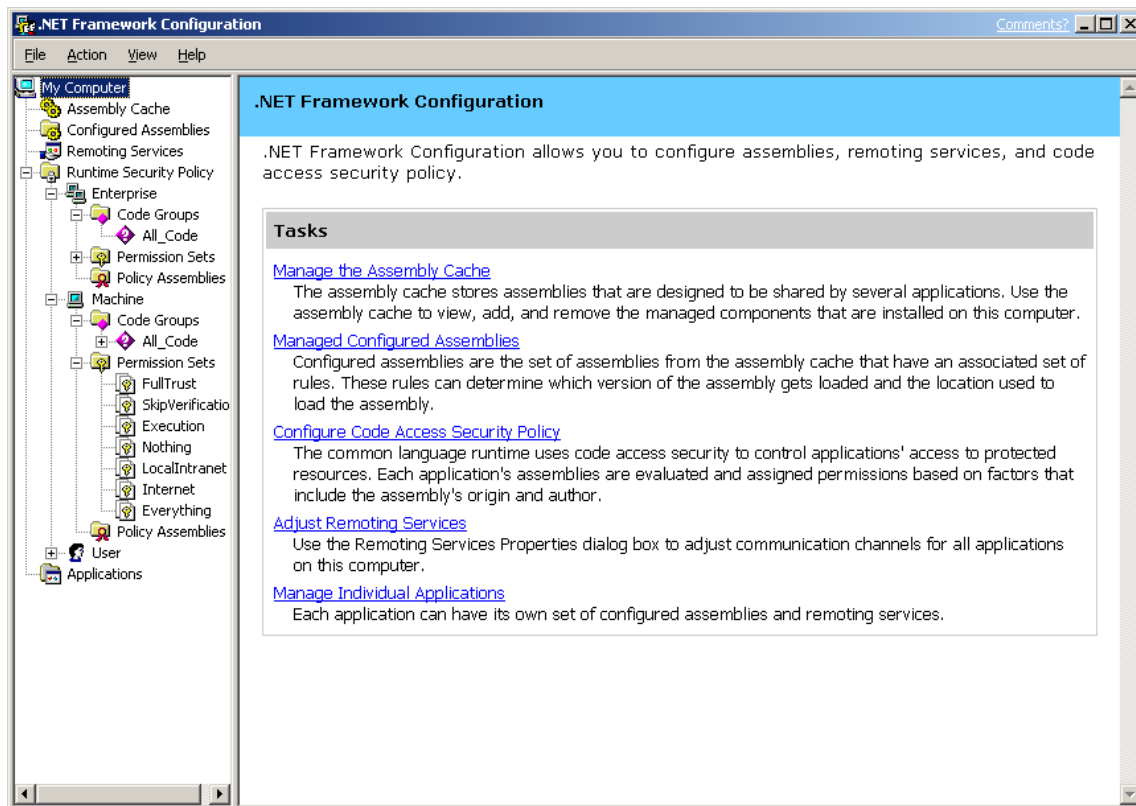


Figure 2. .NET Framework Configuration.

Evidence-Based Security

When an assembly is being prepared for execution, the .NET framework must decide what resources the application may access. This is accomplished by evaluating the *evidence* associated with the assembly and allocating rights based on this evidence.

Evidence refers to the collection of information describing various aspects of the origin of the assembly. The purpose of this information is to establish the level of trust that should be granted to the assembly for execution on the local machine. The information present in assemblies depends on what is known about the assembly in question. Information supplied could include:

Valid Digital Signatures – If an assembly is signed and the signature can be verified, digital signatures may be used to ensure that the code is trustworthy if it comes from a reputable source.

URL or Site – The origin of assemblies may also help establish some level of trust if the source is reputable.

Zone – The zone from where the assembly originated can be used to determine whether privileges should be granted to it. Untrusted zones, such as the Internet in a default setup, have a low amount of privileges granted to it.

Path – The path where the assembly is physically located on disk is accessible. This may be useful to lower privileges of assemblies created by particular users or Web applications that will be serving anonymous requests.

Evidence is used to categorize assemblies into *code groups*, which refer to the different sets of evidence given and associated privileges. Each assembly that is going to be executed is categorized in a code group so that privileges may be determined and applied. After privileges for an assembly have been discovered, the framework checks what permissions the assembly requests to function. If assigned less privileges than the assembly requires, it is not run.

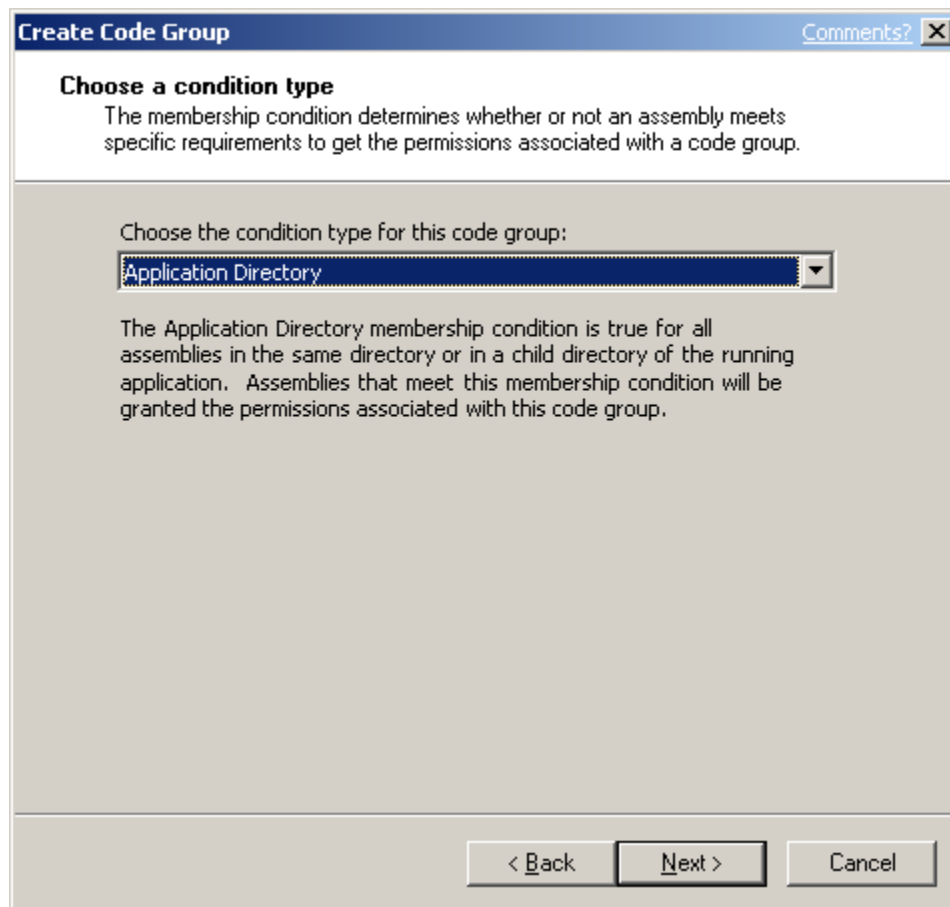


Figure 3. Creating a code group.

The permissions refer to what objects may be accessed on the target system. There are a large number of objects that an assembly may require to perform its function. These objects include DataAccess, FileIO, DNS, EventLog, and Environment. By default, very low privilege sets are granted to assemblies originating from unknown authors on the Internet, so malicious applications cannot make use of the file system or tamper with the Registry.

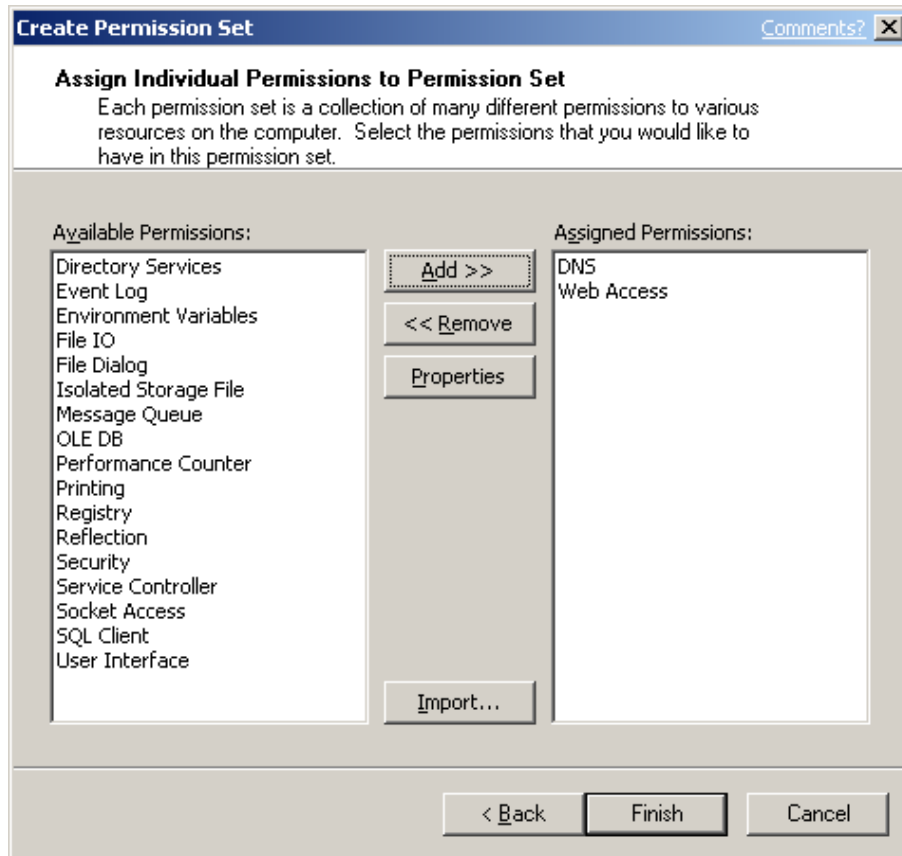


Figure 4. Creating a permission set by assigning individual permissions.

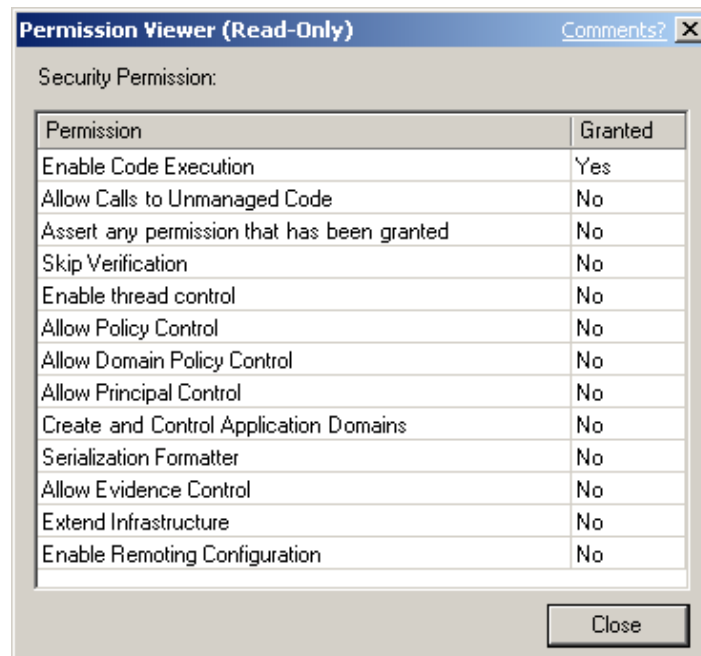


Figure 5. Permission levels for a specific permission set.

Cryptography

Cryptography is an important feature incorporated into the .NET framework because it provides programmers with the ability to securely transmit sensitive data. The .NET framework implements a streams-based encryption layer, which allows data streams to be routed through encryption objects to produce encrypted output streams. The .NET framework handles automatic generation of asymmetric key pairs and symmetric shared secrets applied to CryptoStream objects.

.NET supports the following symmetric algorithms, which use the same key for encryption and decryption:

- DES
- 3DES
- RC2
- Rijndael/AES

.NET supports the following asymmetric algorithms, which use one key for encryption and another for decryption, called public and private keys:

- RSA
- DSA

.NET supports the following hashing algorithms:

- MD5
- SHA1
- SHA256
- SHA384
- SHA512

.NET also includes some support for X.509 certificates provided to application programmers.

Code Access Security

Code Access is a security mechanism that prevents assemblies from acquiring privileges during execution. This mechanism addresses the concern that an assembly might invoke a more privileged assembly and perform a restricted action, allowing malicious programmers to gain access to unauthorized resources and subvert the policies put in place when evidence was first evaluated. The Code Access Security (CAS) mechanism prevents such an attack by initiating a *stack walk* when an object is demanded.

A stack walk is the process of verifying that each assembly in the call-chain has permissions to access the object in question. This process prohibits programmers from performing an attack like the one described above. When a resource is accessed, the application programmer can trigger this stack walk by using the Demand() method of a Permission object. Stack walks can be modified in certain ways to allow certain assemblies to bypass stack checking.

Code Verification

Many security vulnerabilities in software are due to malicious users having the ability to influence the state of the running program and even alter execution flow. The security mechanisms incorporated in the .NET framework require that the CLR's state cannot be altered in such a way. If a malicious assembly were able, for example, to jump to arbitrary locations in memory, it would have complete control, or have the ability to modify objects in memory.

To combat these problems, assemblies are *verified* before they are executed. This process involves ensuring that the assembly is type safe. Type safety verifies that assemblies do not assign variables of different data types to each other or perform evaluations of different types of

data where one must be converted to the type of the other. Additionally, the verification process restricts a programmer's ability to use pointer data types to point to arbitrary locations in memory. This restriction prevents corruption of the state of the machine, because using pointer data types and modifying arbitrary locations in memory could allow state data or memory management data to become corrupted. By default, code that is not verifiably safe is not run on a machine, unless it is local.

Application Domains

In modern execution environments, applications are isolated from each other and cannot directly manipulate another processes' memory or resources. This is a good design principle because applications cannot have an accidental adverse effect on other running applications, and applications cannot maliciously use other applications to leverage access.

The .NET framework extends this principle by introducing *application domains*. Application Domains enforce resource or privilege isolation within a process and can be used for high-end server applications to run multiple domains in one process. This ability can minimize the cost of a context-switch operation, while still providing the security of applications not affecting each other. This feature requires that the verification engine guarantee the inability of applications to jump to or modify arbitrary locations in memory.

Configuration

Now that this paper has discussed security mechanisms, this section lists some of the parameters that can be used to help deploy a secure .NET server. This section describes useful configuration tips for achieving a higher level of security than the default setup.

Managing Code Groups

Evidence and code group configuration can be set with one of two tools: the code access security policy tool (caspol.exe) or the .NET framework configuration tool (mscorcfg.msc). In this section the use of caspol.exe will be demonstrated, but similar operations can be performed with mscorcfg.msc.

Viewing Security Policy

To view the current policies in action (the code group hierarchy):

```
caspol [-<level>] -list
```

The optional level argument defines what policy level you want to list code groups for. It may be either:

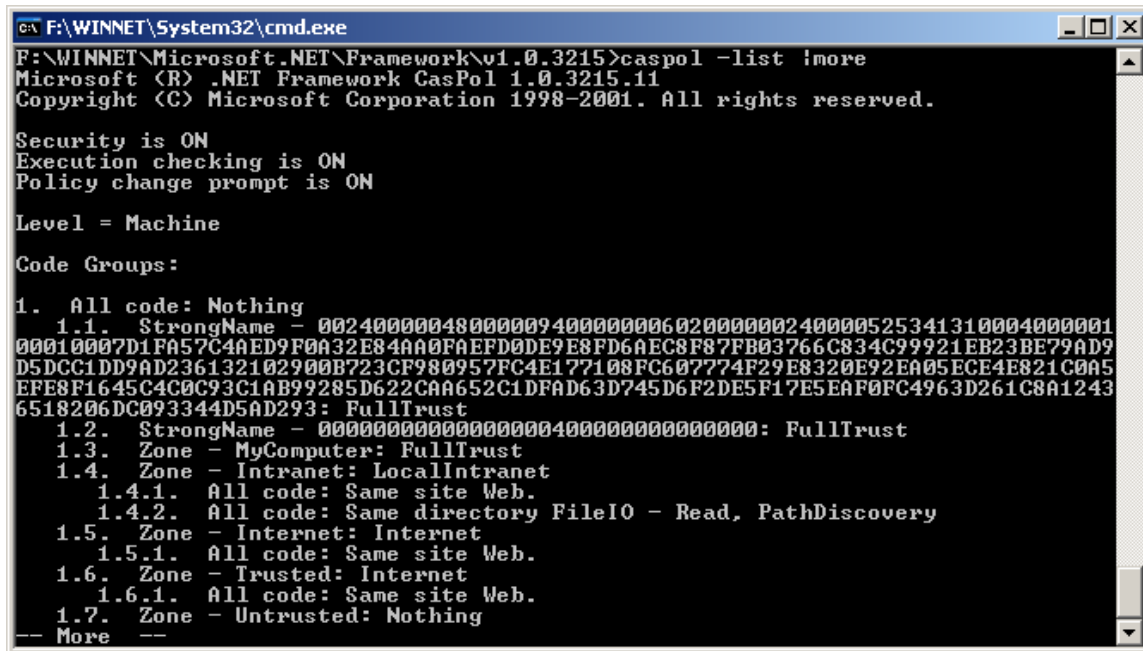
1. enterprise
2. machine
3. user
4. all

This argument may be omitted, in which case the "all" option will be assumed.

To view a code group list, or details:

```
Caspol [-<level>] [-listgroups|-listdetails]
```

The levels are identical to the options described in the previous command.



```

F:\WINNET\System32\cmd.exe
F:\WINNET\Microsoft.NET\Framework\v1.0.3215>caspol -list :more
Microsoft (R) .NET Framework CasPol 1.0.3215.11
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Security is ON
Execution checking is ON
Policy change prompt is ON

Level = Machine

Code Groups:

1. All code: Nothing
  1.1. StrongName - 0024000004800000940000000602000000240000525341310004000001
00010007D1FA57C4AED9F0A32E84AA0FAEFD0DE9E8FD6AEC8F87FB03766C834C99921EB23BE79AD9
D5DCC1DD9AD236132102900B723CF980957FC4E177108FC607774F29E8320E92EA05ECE4E821C0A5
EFE8F1645C4C0C93C1AB99285D622CAA652C1DFAD63D745D6F2DE5F17E5EAF0FC4963D261C8A1243
6518206DC093344D5AD293: FullTrust
  1.2. StrongName - 00000000000000004000000000000000: FullTrust
  1.3. Zone - MyComputer: FullTrust
  1.4. Zone - Intranet: LocalIntranet
    1.4.1. All code: Same site Web.
    1.4.2. All code: Same directory FileIO - Read, PathDiscovery
  1.5. Zone - Internet: Internet
    1.5.1. All code: Same site Web.
  1.6. Zone - Trusted: Internet
    1.6.1. All code: Same site Web.
  1.7. Zone - Untrusted: Nothing
-- More --

```

Figure 6. Using caspol to view current policies.

Adding a Code Group

To add a group:

```
caspol [-<level>] -addgroup <parentname|parentlabel> <membership>
<permission set> [-name name] [-description description] [-exclusive
on|off]
```

The arguments are:

Parentname – The parent code group name that the new code group will be added to under the code group hierarchy.

Parentlabel – The label of the parent code group.

Membership – Membership condition for this code group. The membership arguments are listed in the “Membership flags” section later in this paper.

Permission Set – Name of the permission set to apply to the new code group. These are discussed in section 4.3.

Name – Name of the new code group.

Description – Description of the new code group.

Exclusive – Indicates whether the new groups permissions will override that of other code groups in the same policy level.

Removing Code Groups

To remove a code group:

```
caspol [-<level>] -remgroup <grouplabel|groupname>
```

Where:

1. *grouplabel* — label of the group to remove
2. *groupname* — name of the group to remove

The level is the same as those described in the “Viewing Security Policy” section earlier in this paper, except that you cannot use the “all” qualifier.

Changing Code Groups

Code Groups can be modified and can be assigned new permission sets, membership requirements, name, or description. To change a group:

```
caspol [-<level>] -chggroup <grouplabel|groupname> <attribute changes>
```

Where:

grouplabel – label of the group to change
groupname – name of the group to change

The attribute changes supplied can be any of the parameters for adding a code group, as listed in the “Adding a Code Group” section earlier in this paper.

Membership flags

The following list of membership flags and their descriptions are used when adding or modifying code groups. These flags compare evidence and classify assemblies into code groups.

all – All code.

appidir <directory> – Specifies an application directory in which an assembly resides.

hash <hashalg> <-hex value|-file assembly> – Specifies a hex value or assembly file with a hash.

pub <-cert cert_name|-file signed_file|-hex hex_string> – Specifies assemblies with a certificate, a signature, or a hexadecimal representation of an X.509 certificate.

site <site> – Specifies a site from which assemblies originate.

URL <url> – Specifies a URL from which assemblies originate.

zone <zone name> – Specifies the name of the zone from which assemblies originate. The zone name can be MyComputer, Intranet, Trusted, Untrusted, and Internet.

It is also possible to specify custom membership conditions using XML files.

Configuring Permission Sets

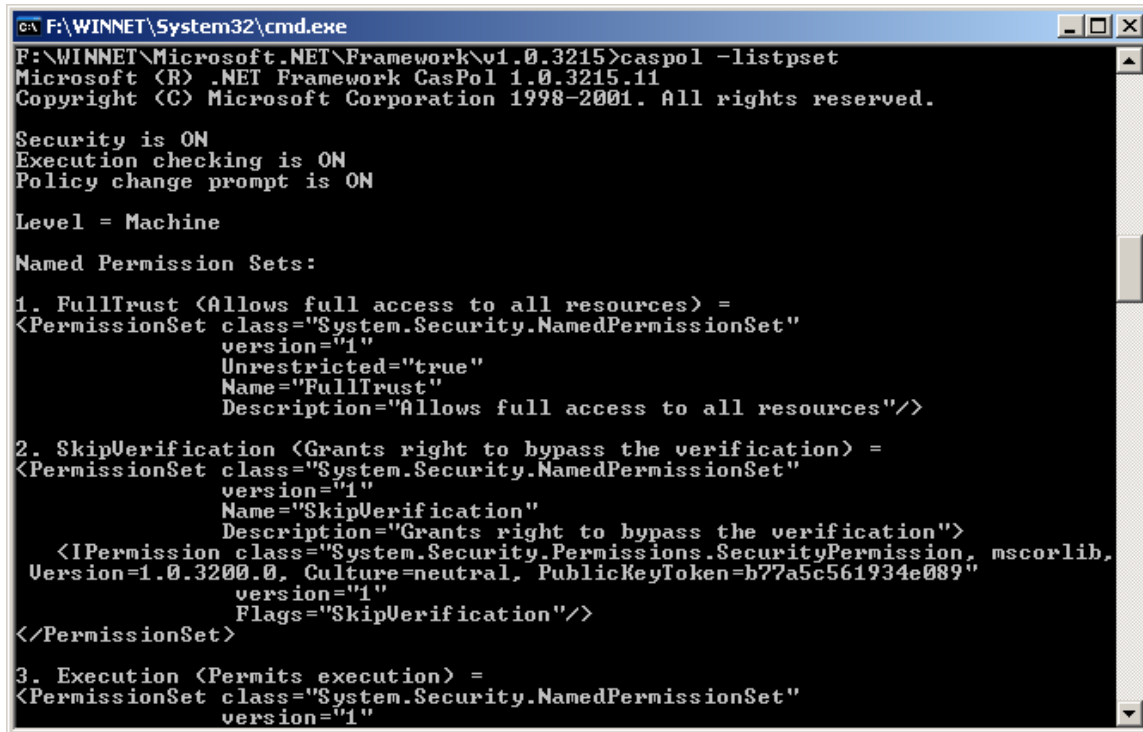
Permission sets are applied to code groups that were discussed in the previous section. This section examines how to set up these permission sets. Permission sets are also administered with the capsol tool.

Viewing Permission Sets

To view permission sets:

```
caspol [-<level>] -listpset
```

The level is the same as described in the “Viewing Security Policy” section earlier in this paper.



```
c:\ F:\WINNET\System32\cmd.exe
F:\WINNET\Microsoft.NET\Framework\v1.0.3215>caspol -listpset
Microsoft (R) .NET Framework CasPol 1.0.3215.11
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Security is ON
Execution checking is ON
Policy change prompt is ON

Level = Machine

Named Permission Sets:

1. FullTrust (Allows full access to all resources) =
<PermissionSet class="System.Security.NamedPermissionSet"
  version="1"
  Unrestricted="true"
  Name="FullTrust"
  Description="Allows full access to all resources"/>

2. SkipVerification (Grants right to bypass the verification) =
<PermissionSet class="System.Security.NamedPermissionSet"
  version="1"
  Name="SkipVerification"
  Description="Grants right to bypass the verification"
  <IPermission class="System.Security.Permissions.SecurityPermission, mscorlib,
  Version=1.0.3200.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
  version="1"
  Flags="SkipVerification"/>
</PermissionSet>

3. Execution (Permits execution) =
<PermissionSet class="System.Security.NamedPermissionSet"
  version="1"
```

Figure 7. Using caspol to view permission sets.

Adding a Permission Set

To add a permission set:

```
caspol [-<level>] -addpset psetfile [psetname]
```

Where:

psetfile – Specifies an XML file that contains the permission set to add. XML Files for permission sets are configured using the <PermissionSet> and <Ipermission> tags.

psetname – Specifies a name for the permission set.

Deleting a Permission Set

To delete a permission set:

```
caspol [-<level>] -rempset psetname
```

Where:

psetname – Specifies a name for the permission set.

Changing a Permission Set

To change a permission set:

```
capsol [-<level>] -chgpset psetfile psetname
```

Where:

psetfile – Specifies an XML file that contains the permission set to add. XML Files for permission sets are configured using the <PermissionSet> and <Ipermission> tags.

psetname – Specifies a name for the permission set.

Secure Programming Tips

Some of the security features described in previous sections require secure programming practices rather than configuration. This section gives a brief overview of some of the programming features present in .NET that can provide the application programmer with a way to ensure secure transmission and execution of assemblies and their associated data.

Cryptography Classes

For secure transmission of sensitive data, .NET provides the System.Security.Cryptography namespace. A large number of classes exist in this namespace, which allow for algorithm service provision as well as classes for encrypting and decrypting data using these service providers. Cryptographic implementation within the .NET framework is streams based. This is done utilizing the CryptoStream class. The CryptoStream class is similar to an IO Stream, with methods allowing for reading, writing, and seeking.

Configuration files

Configuration parameters presented in this document are contained in either the machine.config file or web.config files in the application directories. A malicious user attempting to subvert security mechanisms controlled by the runtime will find these files and their information highly useful.

The machine.config file is by default readable by everyone. This is not a problem unless the processModel tag is used to specify an alternate user to execute applications. The plaintext name and password is contained in this file and would help local attackers gain access to a different account. Additionally, if an attacker gains write access to this file, he or she could enable applications to run with SYSTEM privileges without knowing the password.

Web.config files are also important to protect. These files hold information about who has permissions to use applications and can possibly be read to obtain names and passwords specific to the application. By default, IIS will not allow web.config files to be served to clients, so this information normally cannot be accessed. However, developers should be careful when writing applications so clients do not have the ability to read arbitrary files, since these permissions could be abused to read Web configurations. Also, improper permissions may allow malicious users to write their own web.config files, which would allow them to specify custom authentication mechanisms and potentially execute malicious code.

Code Access Security

Code Access Security is part of the .NET Framework security model. Code originating from different zones has different privileges granted to it that affect what it may or may not be allowed to do. By default, local code is allowed to do a large number of things, which includes gaining higher privileges and accessing local resources.

A compromised Web application may use this feature to its advantage, taking control of the computer and subverting many security checks that are inherent in the .NET framework. An administrator can apply configurations that greatly reduce an application's ability to be subverted. An administrator can specify directories (in this instance, publicly served Web directories) in which assemblies have reduced trust on the computer.

CAS can be done imperatively or declaratively; both achieve the same effect. Programmers can demand that callers have correct permissions using the Demand() method of the various permissions classes (FileIOPermission, ZoneIdentityPermission, and so on), bypass stack walk checks with the Assert() method, or Deny access to resources when they're not in use using the Deny() method.

Cookies

The "Forms" authentication mechanism stores authentication credentials in cookies, which are relayed to and from the server. Attackers who are watching a transaction can obtain these authentication credentials. Administrators are strongly recommended to use SSL whenever using the forms authentication mechanism.

Conclusion

ASP.NET applications can be configured to ensure that clients are securely authenticated and provided access in a controlled manner. This paper has presented configurations that allow for the secure deployment of applications in a distributed environment.

About Internet Security Systems (ISS)

Founded in 1994, Internet Security Systems (ISS) (Nasdaq: ISSX) is a pioneer and world leader in software and services that protect corporate and personal information from an ever-changing spectrum of online threats and misuse. Internet Security Systems is headquartered in Atlanta, GA, with additional operations throughout the Americas, Asia, Australia, Europe and the Middle East. For more information, visit the Internet Security Systems Web site at www.iss.net or call 888-901-7477.

Copyright © 2002, Internet Security Systems, Inc. All rights reserved worldwide.

Internet Security Systems, the Internet Security Systems logo and X-Force are trademarks of Internet Security Systems, Inc. Other marks and trade names mentioned are marks and names of their owners as indicated. All marks are the property of their respective owners and used in an editorial context without intent of infringement. Specifications and content are subject to change without notice.