.NET Framework 4

**Security Changes in the .NET Framework 4**

There have been two major changes to security in the .NET Framework version 4. Machine-wide security policy has been eliminated, although the permissions system remains in place, and [security transparency](http://msdn.microsoft.com/en-us/library/ee191569.aspx) has become the default enforcement mechanism. (For more information, see [Security-Transparent Code, Level 2](http://msdn.microsoft.com/en-us/library/dd233102.aspx).) In addition, some permission operations that presented the potential for security vulnerabilities have been made obsolete.

|  |
| --- |
| **Description: Important noteImportant** |
| Code access security (CAS) has not been eliminated, Security policy has been eliminated from CAS, but evidence and permissions are still in effect. A few permissions have been eliminated, and transparency has simplified the enforcement of security. For a brief overview of the changes, see [Summary of Changes in Code Access Security](http://msdn.microsoft.com/en-us/library/ff527276.aspx). |

You should be aware of the following key points:

* Transparency separates code that runs as part of the application from code that runs as part of the infrastructure. It was introduced in .NET Framework version 2.0, and has been enhanced to become the code access security enforcement mechanism. Unlike security policy, [level 2 transparency](http://msdn.microsoft.com/en-us/library/dd233103.aspx#transparency2) rules are enforced at run time, not at assembly load time. These rules are always in effect, even for assemblies that run as fully trusted by default. However, level 2 transparency does not affect fully trusted code that is not annotated, such as desktop applications. Assemblies (including desktop assemblies) that are marked with the [SecurityTransparentAttribute](http://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute.aspx) and that call methods marked with the [SecurityCriticalAttribute](http://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute.aspx) receive a [MethodAccessException](http://msdn.microsoft.com/en-us/library/system.methodaccessexception.aspx). You can change this behavior by applying the [SecurityRulesAttribute](http://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute.aspx) and setting the [SecurityRulesAttribute..::.RuleSet](http://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute.ruleset.aspx) property to [Level1](http://msdn.microsoft.com/en-us/library/system.security.securityruleset.aspx); however, you should do this only for backwards compatibility. You must explicitly mark a desktop application as security-transparent to apply transparency restrictions to it.
* Code that calls security policy APIs receives a [NotSupportedException](http://msdn.microsoft.com/en-us/library/system.notsupportedexception.aspx) in addition to compiler warnings at run time. Policy may be re-enabled by using the [<NetFx40\_LegacySecurityPolicy> configuration element](http://msdn.microsoft.com/en-us/library/dd409253.aspx). When policy is enabled, security transparency is still in effect. Security policy is applied at assembly load time and has no effect on transparency, which is enforced by the runtime.
* The obsolete request permissions (RequestMinimum, RequestOptional, and RequestRefuse) receive compiler warnings and do not work in the .NET Framework 4, but they do not cause an exception to be thrown. Deny requests cause a [NotSupportedException](http://msdn.microsoft.com/en-us/library/system.notsupportedexception.aspx) to be thrown at run time.
* The LinkDemand security action is not obsolete, but it should not be used for verifying permissions. Instead, use the [SecurityCriticalAttribute](http://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute.aspx) for types and methods that require full trust, or use the [Demand](http://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.demand.aspx) method for types and methods that require individual permissions.
* If your application is built with Visual Studio 2010, you can run it without these changes by specifying a target .NET Framework version that is earlier than the .NET Framework 4 in the Visual Studio project settings. However, you will not be able to use new .NET Framework 4 types and members. You can also specify an earlier version of the .NET Framework by using the [<supportedRuntime> element](http://msdn.microsoft.com/en-us/library/w4atty68.aspx) in the startup settings schema in your [application configuration file](http://msdn.microsoft.com/en-us/library/1xtk877y.aspx).

The following sections discuss these and other changes in the .NET Framework 4:

* [Security Policy Simplification](http://msdn.microsoft.com/en-us/library/dd233103.aspx#simplification)
* [Security Transparency Level 2](http://msdn.microsoft.com/en-us/library/dd233103.aspx#transparency2)
* [Obsolete Permission Requests](http://msdn.microsoft.com/en-us/library/dd233103.aspx#obsolete)
* [Conditional APTCA](http://msdn.microsoft.com/en-us/library/dd233103.aspx#aptca)
* [Evidence Objects](http://msdn.microsoft.com/en-us/library/dd233103.aspx#evidence_object)
* [Evidence Collections](http://msdn.microsoft.com/en-us/library/dd233103.aspx#evidence_collection)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSecurity Policy Simplification

Starting with the .NET Framework 4, the common language runtime (CLR) is moving away from providing security policy for computers. Historically, the .NET Framework has provided code access security (CAS) policy as a mechanism to tightly control and configure the capabilities of managed code. Although CAS policy is powerful, it can be complicated and restrictive. Furthermore, CAS policy does not apply to native applications, so its security guarantees are limited. System administrators should look to operating system-level solutions such as [Windows Software Restriction Policies](http://go.microsoft.com/fwlink/?LinkId=178101) (SRP) or [AppLocker](http://go.microsoft.com/fwlink/?LinkId=178102) on Windows 7 and Windows Server 2008 R2 as a replacement for CAS policy. SRP and AppLocker policies provide simple trust mechanisms that apply to both managed and native code. As a security policy solution, SRP and AppLocker are simpler and provide better security guarantees than CAS.

In the .NET Framework 4, machine-wide security policy is turned off by default. Applications that are not hosted (that is, applications that are executed through Windows Explorer or from a command prompt) now run as full trust. This includes all applications that reside on shares on the local network. Hosted or sandboxed applications continue to run with trust policies that are decided by their hosts (for example, by Internet Explorer, ClickOnce, or ASP.NET). Applications or controls that run in sandboxes are considered partially trusted.

To simplify the security policy, the transparency model has been applied to the .NET Framework. Applications and controls that run in a host or sandbox with the limited permission set granted by the sandbox are considered transparent. Transparency means that you do not have to be concerned about checking CAS policy when you are running partially trusted applications. Transparent applications just run using their grant set. As a programmer, your only concern should be that your applications target the grant set for their sandbox and that they do not call code that requires full trust (security-critical code).

|  |
| --- |
| **Description: Important noteImportant** |
| As a result of these security policy changes, you may encounter compilation warnings and runtime exceptions if you call the obsolete CAS policy types and members either explicitly or implicitly (through other types and members). For a list of obsolete types and members and their replacements, see [Code Access Security Policy Compatibility and Migration](http://msdn.microsoft.com/en-us/library/ee191568.aspx).  You can avoid the warnings and errors by using the [<NetFx40\_LegacySecurityPolicy> configuration element](http://msdn.microsoft.com/en-us/library/dd409253.aspx) in the runtime settings schema to opt into the legacy CAS policy behavior. However, specifying the use of legacy security policy does not include any custom CAS policy for that version unless it is migrated to .NET Framework 4.  You can also enable legacy CAS policy by setting the target .NET Framework version for your Visual Studio project to an earlier version than the .NET Framework 4. This enables legacy CAS policy and includes any custom CAS policies you specified for that version. However, you will not be able to use new .NET Framework 4 types and members. You can also specify an earlier version of the .NET Framework by using the [<supportedRuntime> element](http://msdn.microsoft.com/en-us/library/w4atty68.aspx) in the startup settings schema. |

[Back to top](http://msdn.microsoft.com/en-us/library/dd233103.aspx#top)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSecurity Transparency Level 2

Security transparency was introduced in the .NET Framework version 2.0, but it was very limited and used primarily to improve code validation efficiency. In the .NET Framework 4, transparency is an enforcement mechanism that separates code that runs as part of the application from code that runs as part of the infrastructure. Transparency draws a barrier between code that can do privileged things (critical code), such as calling native code, and code that cannot (transparent code). Transparent code can execute commands within the bounds of the permission set it is operating within, but cannot execute, call, derive from, or contain critical code.

The primary goal of transparency enforcement is to provide a simple, effective mechanism for isolating different groups of code based on privilege. In the sandboxing model, these privilege groups are either fully trusted (that is, not restricted) or partially trusted (that is, restricted to the permission set granted to the sandbox).

Desktop applications run as fully trusted; therefore, they are not affected by the transparency model. For more information about security transparency changes, see [Security-Transparent Code, Level 2](http://msdn.microsoft.com/en-us/library/dd233102.aspx).

[Back to top](http://msdn.microsoft.com/en-us/library/dd233103.aspx#top)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifObsolete Permission Requests

Runtime support has been removed for enforcing the Deny, RequestMinimum, RequestOptional, and RequestRefuse permission requests. In general, these requests were not well understood and presented the potential for security vulnerabilities when they were not used properly:

* A Deny action could be easily overridden by an Assert action. The code in an assembly was able to execute an Assert action for a permission if the permission was in the grant set for the assembly. The Assert prevented the Deny from being seen on the stack, making it ineffective.
* RequestMinimum could not be used effectively outside the application scope. If RequestMinimum appeared on an executable (.exe) file and the grant set was not met, end users of the file received an unhandled [FileLoadException](http://msdn.microsoft.com/en-us/library/system.io.fileloadexception.aspx) exception that contained no information about how to correct the problem. You could not use a single minimum request set for libraries (.dll files), because different types and members in the assembly generally have different permission requirements.
* RequestOptional was confusing and often used incorrectly with unexpected results. Developers could easily omit permissions from the list without realizing that doing so implicitly refused the omitted permissions.
* RequestRefuse did not provide an effective least-privilege model, because it required you to explicitly identify permissions you did not want instead of identifying the permissions you needed. In addition, if new permissions became available, they would not be included in the list. Furthermore, refusal did not make sense for all permissions. For example, you could refuse a value for the [UserQuota](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission.userquota.aspx) property in the [IsolatedStoragePermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission.aspx).

Finally, specifying only the permissions you did not want created the potential for security vulnerabilities if you failed to identify all potentially harmful permissions.

* RequestOptional and RequestRefuse enabled developers to break homogenous domains by creating multiple permission sets within the domain.

The .NET Framework 4 removes runtime enforcement of these enumeration values. Assemblies containing the attributes that use these [SecurityAction](http://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction.aspx) values will continue to load; however, the CLR will not refuse to load the referenced assemblies or modify their grant set based upon the permission sets.

[Back to top](http://msdn.microsoft.com/en-us/library/dd233103.aspx#top)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifConditional APTCA

The conditional use of the [AllowPartiallyTrustedCallersAttribute](http://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute.aspx) (APTCA) attribute enables hosts to identify which assemblies they want to expose to partial-trust callers that are loaded within the context of the host. The candidate assemblies must already be designed for partial trust; that is, they must either be APTCA (security-safe-critical in the transparency model) or fully transparent. A new constructor for the [AllowPartiallyTrustedCallersAttribute](http://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute.aspx) enables the host to specify the level of visibility for an APTCA assembly by using the [PartialTrustVisibilityLevel](http://msdn.microsoft.com/en-us/library/system.security.partialtrustvisibilitylevel.aspx) enumeration in the constructor call.

[Back to top](http://msdn.microsoft.com/en-us/library/dd233103.aspx#top)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifEvidence Objects

Before the .NET Framework 4, almost any object could be used as an evidence object if the hosting code wanted to apply it as evidence. For example, some .NET Framework code recognized [System..::.Uri](http://msdn.microsoft.com/en-us/library/system.uri.aspx) objects as evidence. The runtime considered evidence objects as [System..::.Object](http://msdn.microsoft.com/en-us/library/system.object.aspx) references and did not apply any type safety to them.

This presented a problem because the .NET Framework imposed implicit restrictions on which types could be used as evidence objects. Specifically, any object used as evidence had to be serializable and could not be null. If these requirements were not met, the CLR threw an exception whenever an operation that required one of these assumptions was performed.

To enable constraints on the types of objects that can be used as evidence and to provide the ability to add new features and requirements to all evidence objects, the .NET Framework 4 introduces a new base class, [System.Security.Policy..::.EvidenceBase](http://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase.aspx), which all evidence objects must derive from. The [EvidenceBase](http://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase.aspx) class ensures, upon instantiation, that the evidence object is serializable. In addition, new evidence requirements can be created in the future by adding new default implementations to the base class.

**Backward Compatibility**

All the types used by the CLR as evidence objects have been updated in the .NET Framework 4 to derive from [EvidenceBase](http://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase.aspx). However, custom evidence types used by third-party applications are not known and cannot be updated. Therefore, those evidence types cannot be used with the new members that expect evidence derived from [EvidenceBase](http://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase.aspx).

[Back to top](http://msdn.microsoft.com/en-us/library/dd233103.aspx#top)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifEvidence Collections

Before the .NET Framework 4, the CLR generated the full set of evidence objects that applied to an assembly when the assembly was loaded. These objects were stored in a list, which consumers would then iterate over looking for a specific object. Therefore, all evidence was made available, whether or not it was used. For most evidence objects, this behavior was not an issue; however, for evidence objects such as [System.Security.Policy..::.Publisher](http://msdn.microsoft.com/en-us/library/system.security.policy.publisher.aspx) (which requires Authenticode verification), this behavior was inefficient.

To improve this behavior, the interaction with the evidence collection has been redesigned in .NET Framework 4. An evidence collection now behaves like a dictionary instead of a list. Instead of iterating over the evidence collection to see if a required evidence object exists, consumers can now request a specific type of evidence, and the collection returns the evidence if it is found. For example, the call StrongName name = evidence.GetHostEvidence<StrongName>(); returns a [StrongName](http://msdn.microsoft.com/en-us/library/system.security.policy.strongname.aspx) object if one exists; otherwise, it returns null.

This dictionary model delays the generation of evidence objects until they are requested. In the [Publisher](http://msdn.microsoft.com/en-us/library/system.security.policy.publisher.aspx) evidence example, the performance overhead of verifying the Authenticode signature of an assembly is delayed until that information is needed. In the most common case of full-trust applications where [Publisher](http://msdn.microsoft.com/en-us/library/system.security.policy.publisher.aspx) evidence is not needed, the verification process is avoided altogether.