Secure Coding Guidelines

Securing resource access

Security-neutral code

Application code that isn't a reusable component

Managed wrapper to native code implementation

Most application code can simply use the infrastructure implemented by .NET. In some cases, additional application-specific security is required, built either by extending the security system or by using new ad hoc methods.

Using .NET enforced permissions and other enforcement in your code, you should erect barriers to prevent malicious code from accessing information that you don't want it to have or performing other undesirable actions. Additionally, you must strike a balance between security and usability in all the expected scenarios using trusted code.

This overview describes the different ways code can be designed to work with the security system.

Securing resource access

When designing and writing your code, you need to protect and limit the access that code has to resources, especially when using or invoking code of unknown origin. So, keep in mind the following techniques to ensure your code is secure:

Do not use Code Access Security (CAS).

Do not use partially trusted code.

Do not use the AllowPartiallyTrustedCaller attribute (APTCA).

Do not use .NET Remoting.

Do not use Distributed Component Object Model (DCOM).

Do not use binary formatters.

Code Access Security and Security-Transparent Code are not supported as a security boundary with partially trusted code. We advise against loading and executing code of unknown origins without putting alternative security measures in place. The alternative security measures are:

Virtualization

AppContainers

Operating system (OS) users and permissions

Hyper-V containers

Security-neutral code

Security-neutral code does nothing explicit with the security system. It runs with whatever permissions it receives. Although applications that fail to catch security exceptions associated with protected operations (such as using files, networking, and so on) can result in an unhandled exception, security-neutral code still takes advantage of the security technologies in .NET.

A security-neutral library has special characteristics that you should understand. Suppose your library provides API elements that use files or call unmanaged code. If your code doesn't have the corresponding permission, it won't run as described. However, even if the code has the permission, any application code that calls it must have the same permission in order to work. If the calling

code doesn't have the right permission, a SecurityException appears as a result of the code access security stack walk.

Application code that isn't a reusable component

If your code is part of an application that won't be called by other code, security is simple and special coding might not be required. However, remember that malicious code can call your code. While code access security might stop malicious code from accessing resources, such code could still read values of your fields or properties that might contain sensitive information.

Additionally, if your code accepts user input from the Internet or other unreliable sources, you must be careful about malicious input.

Managed wrapper to native code implementation

Typically in this scenario, some useful functionality is implemented in native code that you want to make available to managed code. Managed wrappers are easy to write using either platform invoke or COM interop. However, if you do this, callers of your wrappers must have unmanaged code rights in order to succeed. Under default policy, this means that code downloaded from an intranet or the Internet won't work with the wrappers.

Instead of giving unmanaged code rights to all applications that use these wrappers, it's better to give these rights only to the wrapper code. If the underlying functionality exposes no resources and the implementation is likewise safe, the wrapper only needs to assert its rights, which enables any code to call through it. When resources are involved, security coding should be the same as the library code case described in the next section. Because the wrapper is potentially exposing callers to these resources, careful verification of the safety of the native code is necessary and is the wrapper's responsibility.

Library code that exposes protected resources

The following approach is the most powerful and hence potentially dangerous (if done incorrectly) for security coding: your library serves as an interface for other code to access certain resources that aren't otherwise available, just as the .NET classes enforce permissions for the resources they use. Wherever you expose a resource, your code must first demand the permission appropriate to the resource (that is, it must perform a security check) and then typically assert its rights to perform the actual operation.