

# Secure Programming with .NET

*David Wong and Rohyt Belani* 2002-11-26

## Secure Programming with .NET

by [Rohyt Belani](#) and [David Wong](#)

last updated November 26, 2002

---

At the core of Microsoft's .NET initiative is the goal of interconnecting businesses, users, applications, and data. However, with all the concerns regarding security and privacy of data, many individuals and companies are reluctant to connect their business systems and place their data in reach of hackers thousands of miles away. Microsoft understands the challenges and concerns facing early adopters of their technology, and has made security one of their top priorities. The fundamental pillar for building applications is the security surrounding the .NET framework and the security services it provides. In this article, we will provide an overview of .NET framework security features and provide practical tips on how to write secure code in the .NET framework. More importantly, we will discuss which pitfalls to avoid.

### Overview of .NET Framework Security

Traditionally programs executed on a local disk where the administrator installed them. However, the advent of distributed computing and the .NET framework have, in essence, eliminated these boundaries of execution. The boundaries are now the seemingly infinite limits of the Internet. The transition to this new environment of dynamic downloads and remote execution has demanded a revision of the traditional security model of "trusted" code.

Security models of operating systems have thus far associated security with users and groups. This means that users, and all code running on their behalf, are either permitted or not permitted to perform operations on critical resources. The .NET framework provides a developer-defined security model called role-based security that functions in a similar vein. In addition to this, the .NET framework provides code-specific security known as code-access security (also referred to as evidence-based security). This facility is fundamental to ensure the secure execution of mobile code. With code access security, a user may be trusted to access a resource but if the code the user executes is not trusted, then access to the resource will be denied.

The security functionalities of the .NET framework reside in the common language runtime (CLR) sub-system. These functionalities reside atop those provided by the operating system

and complement them. However, the CLR security is more granular and configurable as is that provided by the underlying operating system, thus allowing delegation of security of managed code to the CLR (by the operating system).

The following sections provide an overview of the fundamental building blocks of the .NET framework security model: namely, role-based security, code access security and stack walking.

## Role-Based Security

In a business environment, roles define authorization policies. For example, a clerk in a bank may only be authorized to perform a transaction up to a certain monetary limit. The manager, on the other hand, may have a higher limit and the vice-president a higher limit still. The .NET framework's role-based security model is based on a similar concept.

This model is based on the notion of a principal or user on whose behalf the code executes. The identity of the principal can either be based on a Windows account or can be custom defined specific to the application. Apart from identity, the principal has one or more role memberships. These role memberships (for example, the role of clerk or manager) determine the principal's authorization to perform a particular transaction or access a particular protected resource.

## Code-Access Security (Evidence-Based Security)

Role-based security that is user-identity based mandates the operation of all code running on behalf of a particular user with the same access rights and privileges. This model of security did not pose any problems in a world of disconnected desktop machines. However, in the current context of networked computers, treating all code running on behalf of the same user equally may not necessarily be the safest strategy. The origin of the dynamically downloaded or executed code is of significance too, and needs to be incorporated in the security model. This brings us to the concept of code-access security, which authenticates code itself, rather than the user of the code, based on its origin. Information about the origin of the code is commonly referred to as *evidence*. For example, evidence considered for authorization in the .NET framework consists of the following pieces of information:

- **Zone:** The same concept as zones used in Internet Explorer.
- **URL:** A specific URL or file location that identifies a specific resource

- **Hash:** The hash value of an assembly generated with hash algorithms such as SHA1
- **Strong Name:** The strong name signature of an assembly. Strong names represent a versioned, cryptographically strong way to refer and identify an assembly or all assemblies of a particular signing party.
- **Site:** The site from which the code came. A URL is more specific than the notion of a site; for example, [www.microsoft.com](http://www.microsoft.com) is a site.
- **Application Directory:** The directory from which the code is loaded.
- **Publisher certificate:** The Authenticode digital signature of the assembly.

The mapping between the assembly evidence and the permissions granted to the assembly are determined by the practicable security policy, implemented by the `System.Security.SecurityManager` class. Thus, code-access security can be thought of as a function with two input variables (evidence and administrable security policy) and an assembly specific set of permissions as output.

## Stack Walking

When are the permissions generated by the code-access security module checked? The permissions are checked during a process known as a *stack walk*. A stack walk operates as follows. When a new method is called, a new activation record containing the return address, the parameters passed to the method, and any local variables is placed on the stack. This record is popped off the stack on returning from the method. As a result, the stack grows and shrinks during the course of program execution. Before granting access to a method, a protected resource may demand a stack walk which, as the name implies, entails walking through all the records in the call chain and determining if they have appropriate access rights for the requested resource. This procedure ensures a higher level of safety than checking only the permission set of the immediate caller.

## Secure Coding Practices

Applications simply need to use the infrastructure implemented by the .NET framework to tackle most of their security concerns. However, there are few security issues that the application programmer needs to explicitly address. Some of these issues are:

### 1. Requesting minimum required permissions for execution

The .NET application programmer can explicitly specify the minimum level of required permissions for the code to execute. In this case, if the code encounters security policies that do not grant it permission to execute, the code will not run. This facility prevents ugly exception details from being thrown at the user midway through execution and can be achieved by appropriate use of the `SecurityAction.RequestMinimum` method. The programmer should also specify to the system that no additional permissions (more than those required for the legitimate execution of the code) be granted to the code. This prevents widening of security holes elsewhere in the code. This is analogous to using a UNIX account with lowest possible privileges to perform a particular task. This can be achieved by setting `Unrestricted=false` in the permission set.

## 2. Securing sensitive information

Applications that handle sensitive data need to prevent exposure of that data to malicious code. While code access security might stop malicious code from accessing resources, such code could still read values of fields or properties that might contain sensitive information. In order to keep data secure in memory it should be stored as private or internal variables, thus limiting its scope to the same assembly. The programmer must, however, be aware of the fact that the data can still be accessed by highly trusted code under reflection, serialization and debugging.

Data can also be secured as "protected", limiting its access to that particular class and its derivatives. However, the programmer must take necessary steps to ensure that all derived classes implement similar protection. Such controlled inheritance can be achieved using the `InheritanceDemand` feature.

## 3. Cautious use of LinkDemand security checks

The `LinkDemand` feature permits a programmer to alter the usual stack walk procedure to check only the immediate caller of the code at Just-in-time (JIT) compilation time rather than checking all the callers on the stack. This enhances the performance of the code but it allows malicious code to call the restricted code using authorized code. Thus, this feature should be used with extreme caution.

## 4. Validating user input

A golden rule of Web application security has been: do not trust user input. All data from the

client/user must be validated at the server to prevent any scripts and malicious hexadecimal characters from being accepted. User data is often passed as parameters to call other code on the server and, if not validated, can severely compromise the system security (e.g SQL injection attacks, Cross-Site Scripting attacks)

## 5. Cryptography

The CryptoAPI has been in the Microsoft Windows operation system for some time now. With the introduction of ASP.NET, DPAPI, CAPICOM, and new cryptographic features of IIS6, it is now possible to build Web-based application using strong cryptographic protocols that are built into the .NET environment.

### Common Pitfalls

Having discussed some recommended coding practices, we now move on to discussing some precautions to be taken to avoid common pitfalls.

#### 1. ASSERT

The ASSERT statement asserts that a programmer knows what he is doing. It is critical that callers have no control over the assert statement. All assert statements should be reviewed during code reviews.

#### 2. Unmanaged code

The use of managed code provides a lot of "free" security from the .NET framework. On the other hand, unmanaged code is not protected by the framework and should be avoided if possible. Unmanaged code is particularly dangerous if an attacker can execute other code using P/Invoke or COM Interop.

#### 3. Object references

Access control in the .NET framework is implemented to check access rights only when opening a file. If an object reference of an already open file is passed as a parameter to a function, the called function easily subverts security checks when reading and/or writing to the file. Thus designers of application APIs should pay special attention to the manner in which object references are passed.

## 4. SOAP access

Although SOAP and .NET allow complex B2B systems to share data and expand business capabilities, it is critical that access to SOAP interfaces be limited to authorized users, due to the sensitivity of the transactions. Many implementations of applications using SOAP expose too many functions or bypass the authorization and entitlement controls that normal transactions go through. The design of the application should ensure an orthogonal interface for all users and establish a single security checkpoint. Additionally, use of strong authentication such as HTTP certificates and XML signatures is recommended.

## Conclusion

The .NET infrastructure provides a framework to build secure applications by creating an environment in which applications can be controlled and security decisions made based on the evidence of the code, the role of the user, and the security policy of the environment. However, this does not mean that the programmer has no responsibilities as far as security is concerned. Although Microsoft has provided secure default settings that tackle most of the security concerns, the programmer must explicitly address some issues by ensuring adherence to secure coding guidelines and avoidance of common pitfalls.

## Relevant Links

[Secure Coding](#)

*Dave Wong, SecurityFocus*

[.NET/MSIL Malicious Code and AV/heuristic Engines](#)

*Markus Schmall, SecurityFocus*

[Privacy Statement](#)

Copyright 2006, SecurityFocus