Chapter 1

Understanding .NET Security

- Understanding the .NET Framework Security Enhancements
- Defining the Security Issues .NET Doesn't Handle
- .NET Framework Architectural Considerations

Read any trade press magazine and you'll likely run into one or two articles that consider the latest security threat, at least one about the latest security break-in, and several concerning the latest virus. All of these articles make it sound as if every piece of software you own is under attack. In many ways, your software is under attack—even the software you write. Crackers aren't particular; they'll use any hole they find to get into your system and cause damage.

NOTE For the purposes of this book, the term *cracker* will refer to an individual that's breaking into a system on an unauthorized basis. This includes any form of illegal activity on the system. On the other hand, the term *hacker* will refer to someone who performs authorized (legal) low-level system activities, including testing system security. Hackers also help find undocumented solutions to many computer needs and create workarounds for both hardware and software problems. In some cases, you need to employ the services of a good hacker to test the security measures you have in place, or suffer the consequences of a break-in.

The problem with most software now is that it was written before security threats like the ones today were known. In addition, many developers went to school at a time when security courses were rare or nonexistent. Microsoft is hoping to help with the developer training problem at least. As part of their Trustworthy Computing Initiative, Microsoft recently sponsored a new type of security class at the University of Leeds in England (see `http://www.infoworld.com/article/03/03/21/HNmsteachhack_1.html` for details on this story). The thing I find interesting about this course is that it employs hacking as part of the curriculum so the students can actually see both the positive and negative parts of applied security.

This chapter provides an overview of what the .NET Framework can do and what it can't do to secure your systems. The book will discuss both the positive and the negative issues of using .NET as a security solution. You may be surprised to learn that .NET isn't a total solution—that there are many areas of your system that it can't protect. A realistic and honest evaluation of any security technology must include both elements. However, you'll also learn that the .NET Framework provides many options not available in previous Windows technologies and that's a step in the right direction.

Along with the practical issues of data and infrastructure protection, this chapter discusses a few necessary theoretical details. For example, you'll learn some new information about the .NET Framework security architecture. I refer you to existing details in the help file and various Web sites to fill in the areas that other sources have already discussed to death.

Finally, you'll see a few simple coding examples that demonstrate specific coding concepts. The examples in this chapter focus on usability or architectural issues. Chapter 2 actually begins showing how to use the .NET Framework namespaces and classes to create secure applications. Consider the examples in this chapter a preview of the information to come. They're also simpler than the examples that follow—this is a starting point.

NOTE All of the examples for this book are available in both Visual Basic and Visual C#. However, most of the code in the text will appear in Visual C# for consistency reasons. I'll provide you with Visual Basic differences as needed. Otherwise, you can assume the principles I show in Visual C# translate directly to Visual Basic.

An Overview of .NET Framework Enhancements

When Microsoft started designing .NET, the world of computing was moving from the local area network (LAN) and wide area network (WAN) to the Internet. The individual user and group approach Windows used didn't necessarily reflect the best way to pursue security in a distributed environment. In addition, the current environment was too open to potential attacks from outside.

To overcome the security problems inherent in Windows, Microsoft enhanced the role-based security approach originally found in COM+ to work as a general programming methodology. In addition, the managed environment maintains better control over resources that tend to create problems in the unmanaged Windows environment. These two security changes, along with the object-oriented programming strategy of the .NET Framework, summarize what you'll find as security enhancements in the .NET Framework. Here's a list of the critical security enhancements for the .NET Framework.

Evidence-based Security This feature determines what rights to grant to code based on information gathered about it. The Common Language Runtime (CLR) examines the information it knows about an assembly and determines what rights to grant that code based on the evidence. The evidence is actually matched against a security policy, which is a series of settings that define how the administrator wants to secure a system.

Code Access Security The CLR uses this feature to determine whether all of the assemblies in a calling chain (stack) have rights to use a particular resource or perform a particular task. All of the code in the calling chain must have the required rights. Otherwise, CLR generates a security error that you can use to detect security breaches. The purpose of this check is to ensure that a cracker's code can't intercept rights that it doesn't deserve.

Defined Verification Process Before the Just-in-Time (JIT) compiler accepts the Microsoft Intermediate Language (MSIL) assembly, it checks the code the assembly contains for type safety and other errors. This verification process ensures that the code doesn't include any fatal flaws that would keep it from running. The checks also determine whether an external force has modified strongly named code. After these checks are performed, JIT compiles the MSIL into native code. The CLR can run a verified assembly in isolation so that it doesn't affect any other assembly (and more important, other assemblies can't affect it).

Role-based Security If you know how role-based security works in COM+, you have a good idea of how it works in .NET. Instead of assigning security to individuals or groups, you assign it based on the task that an individual or group will perform. The Windows security identifier (SID) security is limited in that you can control entire files, but not parts of those files. Role-based security still relies on identifying the user through a logon or other means. The main advantage is that you can ask the security system about the user's role and allow access to program features based on that role. An administrator will likely have access to all of the features of a program, but individual users may only have access to a subset of the features.

Cryptography The advantages of cryptography are many. The concept is simple—you make data unreadable by using an algorithm, coupled with a key, to mix the information up. When the originator supplies the correct key to another algorithm, the original data is returned. Over the years, the power of computers has increased, making old cryptography techniques suspect. The .NET Framework supports the latest cryptographic techniques, which ensures your data remains safe.

Separate Application Domains You can write .NET code in such a way that some of the pieces run in a separate domain. It's a COM-type concept where the code is isolated from the other code in your program. Many developers use this feature to load special code, run it, and then unload that code without stopping the program. For example, a browser could use this technique to load and unload plug-ins. This feature also works well for security. It helps you run code at different security levels in separate domains to ensure true isolation.

The sections that follow detail the enhancements found in the .NET Framework. There's a lot more to .NET security than you might think. The major shift in strategy may mean that you have to rework your programs, use a new strategy, or rely on alternatives such as using PInvoke (see Chapters 14 and 15 for details).

TIP Microsoft has made changes to the security implementation for the .NET Framework 1.1. For example, applications assigned to the Internet zone now receive the constrained rights associated with that zone, rather than no rights at all. See the overview at `http://msdn.microsoft.com/netframework/productinfo/` for additional security updates.

Using Role-based Security

Role-based security asks the question of whether some entity (a user, the system, a program) is in a particular role. If it's in that role, the entity can likely access a system resource or application feature safely. The concept of a *role* is different from something more absolute like a *group*. When you're a member of a group, you have the same access whether you access the system from a local machine or the Internet. A role does include the idea of group membership, but this is membership based on the environment—the kind of access requested in a given situation from a specific location. An entity's security role changes, rather than being absolute.

Many of the role-based security features you need appear as part of the `System.Security.Principal` namespace. You'll learn more about this namespace and the class it contains in Chapter 2. However, let's look at a simple example, Listing 1.1, of how role-based security can work for checking a user's information. (You can find this code in the `\Chapter 01\C#\RoleBased` or `\Chapter 01\VB\RoleBased` folder of the source code located on the Sybex Web site.)

Listing 1.1 Using the *IsInRole()* Method

```
private void btnTest_Click(object sender, System.EventArgs e)
{
   WindowsPrincipal  MyPrincipal;   // The role we want to check.
   AppDomain         MyDomain;      // The current domain.
   StringBuilder     Output;        // Example output data.
   Array             RoleTypes;     // Standard role types.

   // Set the principal policy for this application.
   MyDomain = Thread.GetDomain();
   MyDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

   // Get the role and other security information for the current
   // user.
   MyPrincipal = (WindowsPrincipal)Thread.CurrentPrincipal;

   // Get the user name.
   Output = new StringBuilder();
   Output.Append("Name: " + MyPrincipal.Identity.Name);

   // Get the authentication type.
   Output.Append("\r\nAuthentication: " +
      MyPrincipal.Identity.AuthenticationType);

   Output.Append("\r\n\r\nRoles:");

   // Create an array of built in role types.
```

```
      RoleTypes = Enum.GetValues(typeof(WindowsBuiltInRole));

      // Check the user's role.
      foreach(WindowsBuiltInRole Role in RoleTypes)
      {

         // Store the role name.
         if (Role.ToString().Length <= 5)
            Output.Append("\r\n" + Role.ToString() + ":\t\t");
         else
            Output.Append("\r\n" + Role.ToString() + ":\t");

         // Store the role value.
         Output.Append(
            MyPrincipal.IsInRole(WindowsBuiltInRole.User).ToString());
      }

      // Output the result.
      MessageBox.Show(Output.ToString(),
                      "User Role Values",
                      MessageBoxButtons.OK,
                      MessageBoxIcon.Information);
   }
```

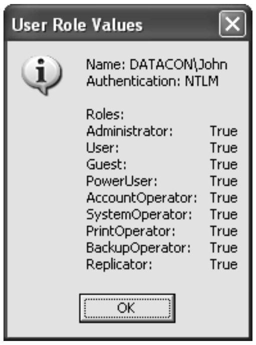
The code begins by obtaining the domain for the current thread. The concept of the application domain appears as the Separate Application Domains bullet. This call is the demonstration of how the feature works. The program is executing in an application domain and you can obtain information about that domain. In this case, the code sets the security policy for this domain equal to the same policy Windows uses. The application is now executing with the same policy that the user has when working with Windows. You could theoretically change that policy depending on conditions such as user location.

Now that the code has set the security policy for the thread, it uses that information to create a `WindowsPrincipal` object, `MyPrincipal`. This object knows all kinds of security information about the user. The code shows how you can obtain the username and the method of authentication used.

The most important use for `MyPrincipal` is to determine which roles the user is in. The book hasn't actually defined any roles yet, so the example uses the `WindowsBuiltInRole` enumeration to check the standard types. If the user is in the requested role, the `IsInRole()` method returns true. This value is converted to a string and placed in Output. Figure 1.1 shows typical output from this example. Of course, the results will change when you run the program on your system because the dialog box will reflect your name and rights.

FIGURE 1.1:

A view of the output from an *IsInRole()* method check



```
Name: DATACON\John
Authentication: NTLM

Roles:
Administrator:      True
User:               True
Guest:              True
PowerUser:          True
AccountOperator:    True
SystemOperator:     True
PrintOperator:      True
BackupOperator:     True
Replicator:         True
```

The important concept to take away from this example is that role-based security performs a similar task to standard Windows security, but using a different and more flexible technique. Because of the differences between Windows security and role-based security, you may need to rely on the standard Win32 API version, especially when working in an environment that has a mix of managed and unmanaged code. Chapters 14 and 15 discuss the Win32 API approach for the managed environment.

Executing Code in the Managed Environment

Instead of simply monitoring and managing the user and other entities that want access to resources or applications, the .NET Framework also monitors the code. Code access security is an important feature because it places the security burden on the code itself, which makes circumventing security measures much more difficult. Because the focus is on the code, a cracker can't use impersonation techniques. It's still possible to attack the code, but most crackers will look for easier targets.

The .NET Framework uses two techniques to ensure proper code access: *imperative* and *declarative*. Imperative security relies on classes that you use within your code and CLR interprets at runtime, while declarative security relies on attributes you place at the head of an element and CLR interprets at link time. You can find a complete description of the differences and usage techniques in Chapter 4. For now, let's look at the easier of the two: imperative security. Listing 1.2 shows an example of imperative security used for gaining access to a file resource. (You can find this code in the `\Chapter 01\C#\Imperative` or `\Chapter 01\VB\Imperative` folder of the source code located on the Sybex Web site—make sure you change the resource file location to match your system.)

Listing 1.2 Relying on Imperative Security for File Security

```
private void btnDeny_Click(object sender, System.EventArgs e)
{
   FileIOPermission  FIOP;       // Permission object.
   Stream            FS = null;  // A test file stream.

   // Create the permission object.
   FIOP = new FileIOPermission(FileIOPermissionAccess.Read,
                               "D:\\Temp.txt");

   // Deny access to the resource.
   FIOP.Deny();

   // Try to access the object.
   try
   {
      FS = new FileStream("D:\\Temp.txt",
                          FileMode.Open,
                          FileAccess.Read);
   }
   catch(SecurityException SE)
   {
      MessageBox.Show("Access Denied\r\n" +
                      SE.Message,
                      "File IO Error",
                      MessageBoxButtons.OK,
                      MessageBoxIcon.Error);
      return;
   }

   // Display a success message.
   MessageBox.Show("File is open!",
                   "File IO Success",
                   MessageBoxButtons.OK,
                   MessageBoxIcon.Information);

   // Close the file if opened.
   FS.Close();
}

private void btnAllow_Click(object sender, System.EventArgs e)
{
   FileIOPermission  FIOP;       // Permission object.
   Stream            FS = null;  // A test file stream.

   // Create the permission object.
   FIOP = new FileIOPermission(FileIOPermissionAccess.Read,
                               "D:\\Temp.txt");

   // Allow access to the resource.
```

```
      FIOP.Assert();

      // Try to access the object.
      try
      {
         FS = new FileStream("D:\\Temp.txt",
                             FileMode.Open,
                             FileAccess.Read);
      }
      catch(SecurityException SE)
      {
         MessageBox.Show("Access Denied\r\n" +
                         SE.Message,
                         "File IO Error",
                         MessageBoxButtons.OK,
                         MessageBoxIcon.Error);
         return;
      }

      // Display a success message.
      MessageBox.Show("File is open!",
                      "File IO Success",
                      MessageBoxButtons.OK,
                      MessageBoxIcon.Information);

      // Close the file if opened.
      FS.Close();
   }
```

The `btnDeny_Click()` method will always fail because the imperative security call, `FIOP.Deny()`, denies access to the file. Notice how the code initializes the `FileIOPermission` object before using it. The code requires a full path to the file in question. The test actually occurs when the code uses the `FileStream()` constructor to try to open the file. When the code fails, the `catch` statement traps the error using a `SecurityException` object, `SE`.

The `btnAllow_Click()` method looks similar to the `btnDeny_Click()` method. However, this method succeeds because the code calls the `FIOP.Assert()` method. You can use the `Assert()` or `Demand()` methods to allow access to an object. One note on this example: make sure you always use the `Close()` method to close the file after a successful open. Otherwise, your code could potentially cause problems on the host system.

Security Problems .NET Can't Stop

Security is a problem precisely because it's a system designed by one person to thwart access to resources by another person. What one person can create, another can just as easily destroy.

Consequently, security is an ongoing process. You need to know the threats your system faces so that you can monitor and protect against them. Although the .NET Framework provides a good basis for building applications with robust security, it can't protect you from other sources of security problems. The sections that follow detail these security problems. They provide suggestions on how you can use .NET along with other strategies to at least keep problems to a minimum. Make sure you look at Chapter 3 as well because it discusses how you can avoid many of the common errors and traps associated with using .NET as a security strategy.

Stupid User Tricks

You've likely seen all of the jokes, heard all of the stories, and seen some of user-related problems yourself. The most horrid stories I've read recently appear in an *InfoWorld* article entitled "Stupid User Tricks." One is about a traveler who had taped both the dial-in phone number for his server and his password to the outside of his laptop carrying case. (See `http://www.infoworld.com/article/03/01/24/030127Security_1.html` for details.) Finding plenty of horror stories is never a problem—solving the security issues users create is another story.

After 17 years of consulting, one of the first security scans I perform at a client site is to check the computer, the desk and surrounding area for pieces of paper with passwords on them. Invariably, I usually find at least one piece of paper with not just one, but several passwords on it. The passwords aren't even hard to figure out. It almost seems that the user is determined to make it easy for the janitor to make off with all of the company passwords.

To an extent, the .NET security features can help overcome problems users create. No, it won't automatically detect Post-it notes placed on the monitor, but it does ensure that the user doesn't gain rights to resources by using code incorrectly. Code access security helps in this regard. For example, when a user requests access to a file, the user's rights are checked and the rights of the code are checked. If either check fails, then access to the file fails. Chapter 4 discusses code access checks in detail.

Training, well-written *policies* (instructions for handling various security situations), and *rules* (requirements that everyone must meet) with some teeth in them can also help with user security problems. However, this is one area where you'll continually have problems because people will normally find a way around rules that are inconvenient. The .NET Framework can help here as well. You can write code in such a way that the network administrator controls security policies, not the user. This technique makes it possible for the network administrator to monitor problems and ensure your program has a reasonable chance of maintaining a secure environment.

It's also important to know how to set the security policies for your application using .NET Framework features. Chapter 5 discusses policies and code groups in detail. While the techniques in this chapter won't fix every user problem, they at least make it harder for someone to destroy the security of your system by accident.

Some External Forces

Most people associate external forces that threaten security with crackers. This group does receive a lot of attention from the media, which is why they're the group that many programmers consider first (and perhaps last). While crackers do cause a great many problems, they aren't the only external force to consider.

Employees on the road or recently let go from the company can cause a great deal of harm to your applications. The problem is twofold:

- Using the old Windows security system, an application might execute at the same privilege level whether the employee accessed it from a desktop or a remote location. The .NET Framework takes this issue into consideration by adjusting the rights of an application based on the zone in which it executes.
- An employee is much more familiar with the security setup and organization of your system than any cracker. Unfortunately, this is a problem that .NET can't help you solve. Someone determined to break into your system is almost certainly going to do it. The only way to overcome this problem is to constantly monitor the system.

Other programs can also affect your application—at least if it's a Web-based application. In a day when distributed applications can rely on services that your program can provide, it's possible that some other programmer you've never met will cause a security problem on your system and not even realize it. The .NET Framework helps you solve this problem by adhering to standards-based security. New standards such as WS-Security and WS-Inspection make it easier to write programs that will work across platforms and provide a secure environment. You can learn more about these standards at Web Services Specifications site (`http://msdn.microsoft.com/library/en-us/dnglobspec/html/wsspecsover.asp`).

Inept network management can also cause serious problems. A network administrator could restore old policies to a server or overwrite the recently patched files for your application. Newer versions of Windows have some protection for both of these issues. For example, Windows File Protection can help ensure only the latest version of your files remain on disk. The .NET Framework also helps by making version checks before it loads files and executes them. An overwritten file would have the wrong version number and .NET would display an error indicating this fact. This fix may not make the user very happy, but at least you'll know the cause of the problem. Chapter 6 tells you more about validation and verification issues.

Another source of external problems is the connections you create with other businesses. Distributed applications are becoming more common as companies invest more in Web services. The problems with distributed applications are many and you can't expect the .NET Framework to solve them all. However, in Chapter 9 you learn how to protect the .NET-specific areas of a Web server. Chapter 10 shows how to protect .NET application generated data. Finally, Chapter 11 demonstrates methods for protecting the connection between a partner and the organization when using a .NET end-to-end solution. Many of the strategies hinge on standards-based application development such as using the WS-Security standard for your application.

Wireless devices present one of the most prevalent and least understood security problems for developers today. Some wireless networks aren't secured at all. Anyone with the proper equipment can come along and access the network from outside the company. Some people are using this technique to spread illicit e-mail using company Internet connections without anyone knowing. The .NET Framework can't do anything about this kind of security problem. However, once you have some basic security in place, Chapter 13 can show you how to make best use of the features the .NET Framework does provide for security of wireless application communication.

Something's always attacking your data, even nature. Fortunately, keeping your data safe, or at least knowing when someone has tampered with it, is relatively easy because this is an area where security professionals have spent a lot of time. First the bad news—if you have any kind of data transfer at all (even on a LAN), be prepared for situations when that data is compromised. The good news is that the .NET Framework uses the latest technologies to secure your information. Chapter 7 discusses cryptographic techniques.

Poorly Patched Systems

Patches are problematic no matter how you look at them. First, there's the problem that the existence of a patch tells many users that the program wasn't well tested, so it failed in use. This perception is fueled by the media in many cases. In fact, the perception is warranted, in at least a few cases, because the product was rushed to market. However, perceptions aside, even if the program has bugs that will cause security problems, the user of that program has to be encouraged to apply the patch to fix the problem.

Second, there's the issue of compatibility. Some patches actually cause more problems than they fix by creating compatibility problems or introducing new problems onto a system. Consequently, many administrators set up test systems to check the patch for problems before they install it on their production system. Extensive testing means the production system is vulnerable longer—perhaps encountering a security problem while it waits.

Although these issues are probably not going away any time soon, you still need to patch your system to ensure it has the latest security features and that any security holes are fixed. You need to consider the results of not patching your system at all and how it will affect any programs you produce on the system. With this in mind, you'll want to use three tools to check your system for problems before you begin developing a new program.

Windows Update Always check Windows Update to verify the status of any patches for your system. Windows Update works automatically. All you need to do is select the updates you want to download. The one mistake that some developers make is not reading the information about a patch before they install it. Sometimes this leads to problems on a development machine. You can find Windows Update at `http://windowsupdate.com/`.

Microsoft System Update Services (SUS) Microsoft SUS can reduce the burden of updating multiple machines for a business. This system helps you download and distribute updates in a safe environment. Learn more about this product at `http://www.microsoft.com/windows2000/windowsupdate/sus`.

Microsoft Baseline Security Analyzer (BSA) Windows setups can become quite complex. It's easy to miss an update when you have multiple pieces of software to consider. Add to this problem, the issue of individual driver updates, and security can become a considerable problem. The Microsoft BSA won't solve all of your problems, but it can help by providing a comprehensive list of problems within a system. Learn more about this product at `http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/Tools/MBSAhome.asp`.

All of these update tools should tell you something about your own programs. When I start the Acrobat Reader, it automatically asks if I want to scan for the latest updates. Other programs on my machine are starting to do the same thing. The .NET Framework provides more version checking features than the Win32 API ever thought about providing. The goal of these features is to prevent problems such as DLL hell, an issue that still plagues Windows systems. You can easily use version checking to provide update checking for your application. (Make sure you still give the user the choice of actually installing the update.)

Inept Enterprise Policies

I actually knew of a business that had a policy in place that made every network user an administrator. It was a moderately sized business that lacked a real network administrator. The person who was supposed to administrate the network was simply too lazy to set up the required features and didn't want to hear a lot of user complaints about not being able to access a particular file or feature. That business may still be around, but I found it quite easy to leave before the inevitable security bomb exploded in someone's face. Fortunately, not every business has a policy as absurd as this one.

Setting enterprise policies is one case when the .NET Framework might actually increase the number of security problems your application will have to face. Microsoft assumes that every network administrator is faithful, loyal, hardworking, conscientious, and knowledgeable. The .NET Framework is set up to ensure the network administrator has maximum access to security features. Unfortunately, some network administrators just aren't up to the task and you'll have to find ways to overcome this problem.

Training is one way to help network administrators fulfill their new role with the applications you create. Many will find it quite surprising that they can control who uses a particular function within your components and specific features in an application. Chapter 4 shows you other techniques to work around this problem. The best strategy is to effectively use imperative and declarative security in your applications to ensure they maintain some level of security even if the network administrator doesn't do anything at all.

Enterprise policies are affected by the environment. A well-maintained domain setup is more secure because the policies are set on the server. Even if the user moves to another machine, you have some assurance that your program will have the proper security and work as anticipated. Unfortunately, the alphabet soup of technologies used for LAN-based applications today makes security difficult (perhaps impossible). Chapter 8 shows you how to get around some of the problems with LAN security. This chapter is an open view of what .NET can do for you and more important, what you still need to tackle using other technologies.

Problems begin with local security policies because they're often implemented unevenly and the rights don't move with the user. Code access security ensures that a rogue program can't make use of this scenario to circumvent security. You can also validate the user, computer, and other data using Active Directory. Chapter 12 discusses how you can use Active Directory to reduce some of the local security problems an application will encounter. It also demonstrates techniques for working with Active Directory security information.

Distributed applications are notorious for opening security holes because these applications make resources that are normally hidden visible to users through an Internet connection. It's amazing that some companies don't have security policies in place to protect from these problems. What's even more amazing is that a few companies actually end up opening new holes in their firewall and other security features to enable an application to work. For example, some companies still rely on Distributed Component Object Model (DCOM) applications to create distributed applications. The .NET Framework isn't the only solution at your disposal, but by using modern Web services techniques, you can greatly reduce the risks distributed applications impose.

One of the most interesting security problems is one in which a company follows all other best practices, but then fails to lock the computer up so that no one can access it physically. If someone can gain physical access to a server, they can control the server in a relatively short time. A worst case scenario occurs when the company not only leaves the computer in the open, but thoughtfully provides a monitor and keyboard so that anyone entering the room can modify the server setup. Although the best policy is to lock the computer up, there are alternatives if you must keep the computer in plain view. For example, you can add physical locking to the system. The World of Data Security (`http://www.crocodile.de/`) provides a number of specialized computer-locking mechanisms that make physical access a lot more difficult.

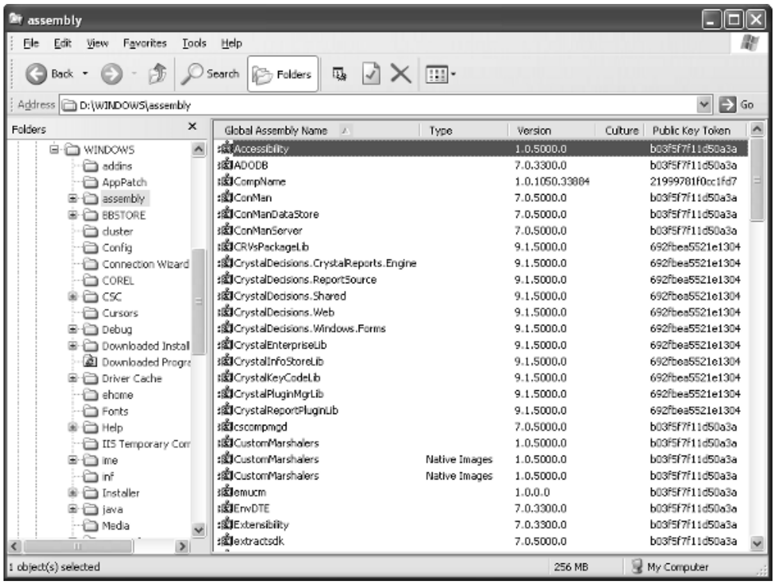
Windows File Protection Vulnerabilities

Windows File Protection is thought to be a safe way to install executable files on a system because those files are supposedly signed. Unfortunately, there's a well-known method to create files that Windows File Protection will trust even if they aren't signed by an appropriate digital certificate. You can read the detailed version of the problem on the Bugtraq site at `http://archives.neohapsis.com/archives/ntbugtraq/2002-q4/0122.html`. A Microsoft security bulletin on the topic appears at `http://www.microsoft.com/technet/security/bulletin/MS00-072.asp`.

The interesting thing about the .NET Framework is that it doesn't rely on this security mechanism. When you sign a control, component, or application, the signature is verified by the CLR. The signature uses a hash mechanism that exists outside the Windows File Protection environment, so the security of the file is ensured. You can always check the token used to secure a file with a strong name that's contained within the Global Assembly Cache by viewing the content of the `\WINDOWS\assembly` folder. Figure 1.2 shows a typical example of the content of this folder.

FIGURE 1.2:

Managed files use a unique signature system that ensures their integrity.



Global Assembly Name	Type	Version	Culture	Public Key Token
Accessibility		1.0.5000.0		b03f5f7f11d50a3a
ADODB		7.0.3300.0		b03f5f7f11d50a3a
CompName		1.0.1050.33804		21999781f0cc1fd7
ConMan		7.0.5000.0		b03f5f7f11d50a3a
ConManDataStore		7.0.5000.0		b03f5f7f11d50a3a
ConManServer		7.0.5000.0		b03f5f7f11d50a3a
CRVsPackageLib		9.1.5000.0		692fbea5521e1304
CrystalDecisions.CrystalReports.Engine		9.1.5000.0		692fbea5521e1304
CrystalDecisions.ReportSource		9.1.5000.0		692fbea5521e1304
CrystalDecisions.Shared		9.1.5000.0		692fbea5521e1304
CrystalDecisions.Web		9.1.5000.0		692fbea5521e1304
CrystalDecisions.Windows.Forms		9.1.5000.0		692fbea5521e1304
CrystalEnterpriseLib		9.1.5000.0		692fbea5521e1304
CrystalInfoStoreLib		9.1.5000.0		692fbea5521e1304
CrystalKeyCodeLib		9.1.5000.0		692fbea5521e1304
CrystalPluginMgrLib		9.1.5000.0		692fbea5521e1304
CrystalReportPluginLib		9.1.5000.0		692fbea5521e1304
cscompmgd		7.0.5000.0		b03f5f7f11d50a3a
CustomMarshalers		1.0.5000.0		b03f5f7f11d50a3a
CustomMarshalers	Native Images	1.0.5000.0		b03f5f7f11d50a3a
CustomMarshalers	Native Images	1.0.5000.0		b03f5f7f11d50a3a
emucm		1.0.0.0		b03f5f7f11d50a3a
EnvDTE		7.0.3300.0		b03f5f7f11d50a3a
Extensibility		7.0.3300.0		b03f5f7f11d50a3a
extractsdk		7.0.5000.0		b03f5f7f11d50a3a

.NET Framework Security Architecture Considerations

It's important to know something about the .NET Framework security architecture and determine how you can use that architecture to fix problems. You also need to know about issues and problems you'll face as a result of the architectural decisions that Microsoft has made. This section of the chapter discusses some architectural issues you need to consider before you begin coding. You'll learn about other architectural considerations (practical versus theoretical) as the book progresses.

NOTE Microsoft has improved the .NET Framework security features in Version 1.1, the version that I use for this book. Some of the calls in the example programs for this book don't appear in the .NET Framework 1.0. I'll let you know when you'll need Version 1.1 in most cases. However, if you run into a call that obviously isn't in the version of the .NET Framework you're using, make sure you update to the most current version.

Securing the Binary Output

Securing text usually isn't as big a problem for organizations as securing binary information. All you need for text is a good encryption technology and the code to use it. Binary data is entirely different because it usually involves complex protocols and can include code. In fact, security of some older technologies, such as DCOM, makes securing binary data almost impossible. The problem is that these older technologies often use ports indiscriminately and require the data in an unencrypted form to ensure the recipient can read it.

The .NET Framework makes it easier to secure binary data. Not only does it avoid the problem of using multiple ports, it also provides methods for encrypting and decrypting data as part of the serialization stream. You can also rely on code access security to help in this situation by controlling the `BinaryFormatter` object. All you need it a set of permissions to go with the serialization and de-serialization process. A permission could be as simple as adding the following attribute to your code:

```
[SecurityPermissionAttribute(SecurityAction.Demand,SerializationFormatter=true)]
```

If the code doesn't have the proper permissions, CLR won't allow the `BinaryFormatter` to do anything. Of course, you could just as easily use imperative security (the `SecurityPermission` class) to achieve the same goal.

Understanding the Effects of Garbage Collection

Garbage collection is a wonderful idea. It means that you spend a lot less time worrying about memory allocation and freeing resources. The Garbage Collector takes care of freeing these resources for you. Of course, you can always step in and dispose of objects when necessary. In fact, this is a recommended procedure when you have large objects and aren't sure when the Garbage Collector will kick in.

The Garbage Collector also impacts security because it affects memory and resources. What happens if a cracker wants to look around in memory and determine what's going on or look for some interesting data? Your program might think that data's already gone, but there it is, in memory waiting for the Garbage Collector. The issue of waiting for the Garbage Collector is normally not something you need to consider, but it is if the object concerned contains sensitive data.

From an architectural perspective, the Garbage Collector is probably the best thing to happen to developers in a long time. However, from a security perspective, it's a problem because objects have an indeterminate life span. The solution, in this case, is to build your sensitive objects with methods for disposing of the sensitive information. You should then build your code so that it always frees objects immediately. Freeing the objects will get the Garbage Collector busy immediately, rather than waiting for the program to exit.

Considering the Requirements of Object-Oriented Programming

One of the best reasons to use .NET to build secure applications is that it's fully object oriented. You can attach security to every element of your program. The security system will faithfully check every access to every method, property, and event of every object your application creates. In general, you can bind your .NET application to the point where nothing can get in—at least not directly.

The problem with .NET security is that every access requires a stack walk and that takes time. The .NET Framework has some optimization built in to reduce the number of complete stack walks, but the basic idea is that security demands that permissions get checked every time a request is made. This level of checking makes .NET secure, but it also makes .NET slow.

As the book progresses, you'll learn several methods you can use to keep speed problems to a minimum. For example, using *declarative security*, whenever possible, will speed your application because some declarative security requirements are handled at link time, rather than runtime. In addition, declarative security attaches the security to the method, rather than to the object, whereas *imperative security* is handled completely at runtime and focuses on the objects you create in the code.

Understanding Native Code Access Concerns

Just about every developer has a wealth of native Windows code to consider as part of any application development task. Most developers have concluded that they'll need to use PInvoke to continue using the unmanaged code that they've spent so much time and resources developing. However, what happens when you mix managed and unmanaged code in the same project? Obviously, the unmanaged code won't understand role-based security or code access security.

TIP Creating PInvoke code can become difficult if you use complex data structures and require access to some of the less documented Win32 API features. You may also want to move your components and controls from the unmanaged to the managed environment. My book, *.NET Framework Solutions: In Search of the Lost Win32 API* (Sybex, 2002, ISBN: 0-7821-4134-X) can help you overcome any of the problems you'll run into when working with the Win32 API.

One solution to this problem is to use Win32 API security for the entire application. In fact, that's what many developers are doing now because that's what they understand best. Knowing that your application employs the same security techniques across all elements can add a bit of comfort in a world that's increasingly insecure. The benefit of this technique is that it's relatively easy and fast. You also know the problems involved in using this method, so you can easily watch for them as you maintain the security of your system.

Another solution is to place the managed code within a separate domain. This is the solution that affords you the best chance of catching problems early. The .NET Framework security is definitely more robust than the security used by your Win32 API code. The advantage of this method is obvious: greater security is always a good thing when you can get it. However, there are several disadvantages to consider:

- For one thing, there's a performance hit for placing the individual code elements in separate domains.
- You'll also have to consider the problems of maintaining two different security techniques until you make all of your code managed. Complexity always breeds errors, which is anathema for secure systems.

In sum, the .NET Framework architecture gives you two basic choices for handling security when you have a mix of managed and unmanaged code—a condition that will exist for quite some time in the Windows community. What you need to consider are the trade-offs between the two choices. Neither choice is a perfect solution, but one choice might work better than another for your particular application.

You also need to consider some types of native code access in light of the way that .NET handles objects. CLR checks every object for security violations, even those accessed through the interoperability layer. In some cases, you can get a performance boost for your application and reduce interoperability issues by removing these runtime checks using the `[SuppressUnmanagedCodeSecurity]` attribute.

Summary

This chapter has provided an overview of the features that make .NET Framework security unique. In addition, you've seen that this security has both strengths and weaknesses. Finally, you've learned how the .NET Framework architecture makes using this technology easier, faster, and less error prone than previous Windows security technologies.

Before you go any further, consider again the problems listed in the "Security Problems .NET Can't Stop" section of the chapter. These are real issues that you need to address before most of the techniques in this book will provide you with a reasonable assurance of secure computing. Not all of these issues have a one-time fix—security is an ongoing process. Although you can't absolutely stop users from pasting their passwords on their monitor for future use, you can certainly discourage the practice by putting policies in place and providing training.

Chapter 2 begins a detailed discussion of .NET Framework features. You learn about the namespaces and classes that you can use to create a secure application environment using the .NET Framework. Chapter 2 is also the first chapter that contains code that you can use to create secure applications: consider it a starting point. Subsequent chapters provide detailed examples that help you build robust applications with great security.