BONUS CHAPTER 4

Building Security into Your Applications

Learn how to provide simple security in your applications

Building a secure application starts with having control of who has access to your application.

Find out how to secure your database from outside users

Securing your database provides you with a greater level of application security, and is easier to achieve than you might think.

Discover how to provide an enhanced level of security in your Web applications

Today's Internet applications require an additional step to provide a secure environment for users.

Find out how to digitally sign your ActiveX controls

Digitally signed controls provide a unique electronic fingerprint that indicates what company or person created the control.

In the not-so-distant past, applications developers focused more on how the application worked than how it might affect the user's environment. With the explosion of Internet applications and the magnitude of data that is now readily available on the Web, security is a very big concern. Today, the possibility exists for your data and applications to fall into the hands of individuals who have no right to access your private information. This chapter looks at a couple of very simple and effective ways to secure your data and guarantee that your application will not cause harm to your users' systems.Whether you are planning to create a very large client/server application or a small shareware product, security should definitely be on your short list of must-have features. In the past, VB programs were somewhat limited in the security features that could be used, especially on Microsoft Access databases. Today, however, VB programs have a wider selection of security options than ever.

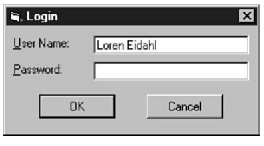
One popular means of providing security is through a serial number that is keyed to your application. This serial number typically is entered by the user during the program installation. These serial numbers have the benefit of not only verifying registered users, but also of enabling the functionality of the application.

Today's applications also need to address the ever-increasing presence of the Web. The Internet is perhaps the biggest threat to security that applications have faced to date. One of the more widely accepted methods of providing security is through the use of digital signatures attached to the controls that make up the Web application. The process of digitally signing a control does not inherently make it safe. You can take other steps designed to act as a complement to the digital signature. ■

The Logon Screen

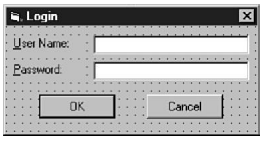
This first line of defense in any secure system is controlling the users who have access to the system. One of the most common ways to do this on an application level is through the use of a simple logon screen, as shown in Figure B4.1. Although the logon screen is a very simple form to create in Visual Basic, requiring only a handful of controls and some minor coding, it has the potential to wield a lot of power. The logon screen can be used as a simple gateway into your system with hardcoded values, or it can be an excellent tool to control how your application functions for each user. This section will take a look at how to create a simple logon screen that can be scaled to whatever level of sophistication you need.

FIG. B4.1
A typical logon screen can act as a gatekeeper to applications.



To create the logon screen, start a new project in Visual Basic. Add controls to a standard form so that it is similar to Figure B4.2. After completion of the logon form, you can begin entering the code.

FIG. B4.2
Creating a logon screen in Visual Basic requires only six controls and five minutes.



The actual code that comprises the logon project is very straightforward. The code for variable declarations is shown in the following code line. The code that resides in the `Click` events of the OK and Cancel buttons is shown in the following piece of code and Listing B4.1, respectively.

```
Public OK As Boolean
```

Listing B4.1 Validate.txt—Validating the User's Entry with the *Click* Event of the OK Button

```
Private Sub cmdOK_Click()
 If txtPassword.Text = "Password" And txtUserName.Text = "Breanna" Then
        OK = True
        Me.Hide
        MsgBox "Logon Successful!", , "Login"
    Else
        MsgBox "Invalid Password, try again!", , "Login"
        txtPassword.SetFocus
        txtPassword.SelStart = 0
        txtPassword.SelLength = Len(txtPassword.Text)
    End If
End Sub
```

In Listing B4.2, the accepted values of `"Password"` and `"Breanna"` have been hardcoded into the code. A message box is displayed if any values other than what are hardcoded are entered into the text boxes.

Listing B4.2 reset.txt—Resetting the Logon Form with the *Click* Event of the Cancel Button

```
Private Sub cmdCancel_Click()
    OK = False
    Me.Hide
End Sub
```

As you can see in the previous code listings, creating a logon form is indeed a very easy process. Now that you have a basic idea of what's involved in this process, you can add a few features to make it more interesting.

Extending the Logon Screen

One of the more popular techniques for logon forms is to place the user's user name into the username field of the logon screen. The user name is actually a system variable name that the computer uses to recognize each user, such as in a network environment. In a Windows 95 environment, the system stores the name that you used when you installed Windows in the user name variable. The code in Listing B4.3 is a replacement for the code that you already entered in Listing B4.1. The code in Listings B4.4 and B4.5 show what additional code is required.

Listing B4.3 click.txt—A Direct Replacement for the Existing Code of *Click* Event on the OK Button

```
Private Sub cmdOK_Click()
    If txtPassword.Text = "Password" Then
```

continues

Listing B4.3 Continued

```
        OK = True
        Me.Hide
        MsgBox "Logon Successful!", , "Login"
    Else
        MsgBox "Invalid Password, try again!", , "Login"
        txtPassword.SetFocus
        txtPassword.SelStart = 0
        txtPassword.SelLength = Len(txtPassword.Text)
    End If
End Sub
```

Listing B4.4 init.txt—Initializing Variables and Calling a Function that Retrieves the User Name From the System

```
Private Sub Form_Load()
    Dim sBuffer As String
    Dim lSize As Long
    sBuffer = Space$(255)
    lSize = Len(sBuffer)
    Call GetUserName(sBuffer, lSize)
    If lSize > 0 Then
        txtUserName.Text = Left$(sBuffer, lSize)
    Else
        txtUserName.Text = vbNullString
    End If
End Sub
```

Listing B4.5 getname.txt—This Code Is an API Call to the *GetUserName* Function

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias
➥"GetUserNameA"(ByVal lpbuffer As String, nSize As Long) As Long
```

`GetUserName` is a function that retrieves the name of the current user. In a non-networked environment, this is the name of the person who is registered to the software. The `GetUserName` function returns a non-zero value on a successful operation and a zero if there is an error. The parameters that `GetUserName` can take are described in Table B4.1.

Table B4.1 *GetUserName* Parameters

Parameter	Description
`lpBuffer`	A string pre-initialized to the length `nSize`. It is loaded with the user name.
`nSize`	A long variable initialized to the length of `lpBuffer`. On return, it contains the number of characters loaded into `lpBuffer`.

▶ **See** "Calling Basic APIs and DLLs," in Chapter 46.

The Future of the Logon Screen

As mentioned earlier in the chapter, the logon screen can be a major player in your application's security system. This chapter has already touched on a few ways that you can use the logon screen to provide a level of security to your applications. Now, a few ways to increase the functionality of the logon screen will be presented. The following suggestions should provide a few starting points toward that goal:

- The logon screen can very easily be tied to a table in a database that is queried to determine proper logon and password values. This eliminates the need to have hardcoded values (which is not very secure).
- The logon screen can be coded so as to verify the user name against the system name to determine a proper match. Currently, the user name is displayed, thereby increasing the possibility of a security breach by 50 percent.
- The logon screen can be tied to a table that, based on a user's login, automatically adjusts the application to fit the desired setup profile or to enable specific program functionality.

The actual implementation of the preceding ideas is left up to you. However, you can see from the list that the possibilities are indeed intriguing. The next section explains how to secure your database to provide an even greater level of protection for your data.

Creating a Secure Database

In the early days of Visual Basic, the application programmer had very little control over database security. If a programmer chose to use the Jet engine, the security options became even fewer. The only way to write applications that used a truly secure database was through some type of ODBC connection to a SQL database engine. The Jet engine at that time had a very limited security system that was implemented through the System.MDA database. Fortunately for developers using Visual Basic 5, those days are long gone.

Visual Basic 5 gives application developers who use the Jet engine a number of effective ways to implement security on their databases, ranging from a simple password to user- and group-specific rights.

Securing your application against unregistered users is a good first step. Now look at how to secure the data that your application depends on. With the popularity of the Jet database engine comes the possibility of increased security threats to your data. An unsecured Jet database is open to anyone who has access to the correct version of Microsoft Access. With the introduction of the Jet 3.x database engine, Microsoft provides the tools needed to adequately secure any application that relies on the Jet 3.x database engine, including the following methods:

- `OpenDatabase`
- `NewPassword`
- `CreateWorkspace`

- `CreateUser`
- `CreateGroup`

Some of these methods are new, and some are enhanced to use the new security features. The following sections take a look at these methods.

Using the *OpenDataBase* Method

For most Visual Basic programmers, the `OpenDataBase` method is a familiar one. One of the nicest things about this method is that you now can pass a password that is to be used on the database via the connect string. This password approach can be used on both Jet 3.x databases and ODBC databases, such as FoxPro or SQL Server. The syntax for the `OpenDatabase` method is as follows:

```
Set database = workspace.opendatabase(dbname, options, read-only, connect)
```

Table B4.2 describes the parameters of the `OpenDataBase` method.

Table B4.2 *OpenDatabase* Method Parameters

Parameter	Description
`Database`	An object variable that represents the `Database` object that you want to open.
`Workspace`	An optional object variable that represents the existing `Workspace` object that will contain the database. If you don't include a value for `Workspace`, `OpenDatabase` uses the default `Workspace`.
`dbname`	A String that is the name of an existing Microsoft Jet database file, or the data source name (DSN) of an ODBC data source.
`options`	An optional `Variant` datatype that sets various options for the database, as specified in Settings.
`read-only`	An optional `Variant` (Boolean subtype) value that is `True` if you want to open the database with read-only access, or `False` (default) if you want to open the database with read/write access.
`connect`	An optional `Variant` (String subtype) that specifies various connection information, including passwords.

The code segment in Listing B4.6 illustrates how the `OpenDatabase` method can be used with a database that uses a password.

Listing B4.6 getpass.txt—Supplying a Password to Either a Jet or an ODBC-Type Database

```
Dim workJet As Workspace
Dim dbData As Database
```

```
Set dbData = workJet.OpenDatabase("Users", _
        dbDriverNoPrompt, True, "ODBC;DATABASE=Sample;PWD=mypassword")
```

Using the *NewPassword* Method

The NewPassword method enables you to change the password of an existing user account. This method applies only to a Microsoft Jet Workspace. The syntax for the NewPassword method is:

```
Object.NewPassword oldpassword, newpassword
```

Table B4.3 describes the parameters of the NewPassword method.

Table B4.3 *NewPassword* Method Parameters

Parameter	Description
object	An object variable that represents the User object or a Jet 3.x database object
oldpassword	A String that represents the current setting of the Password property of the User object or a Jet 3.x database object
newpassword	A String that is the new setting of the Password property of the User object or a Jet 3.x database object

The following notes apply to the use of the NewPassword method:

- Passwords are case-sensitive.
- If a database currently has no password, the Jet engine will create one by passing a zero-length string for the old password.
- The OldPassword and NewPassword objects can be up to 14 characters in length and can include any characters except null.
- To clear the password, use a zero-length string for NewPassword.
- To set a new password, you must log in as the user whose account you are changing or be a member of the Admin group.

The code segment in Listing B4.7 illustrates how to create a new password on a Jet database by using the NewPassword method.

Listing B4.7 changepass.txt—Creating or Reassigning a Password to a Jet 3.x Database

```
Dim usernew as user
Dim strPassword as string
strPassword = Inputbox("Enter New Password")
Select Case Len(strPassword)
```

continues

Listing B4.7 Continued

```
     Case 1 to 14
          usernew.Newpassword "oldPassWord", strPassword
msgbox "Password Change Sucessful"
            Case is > 14
          MsgBox "Password is too long!"
     Case 0
          exit
End Select
```

CAUTION

If you lose or forget the password to your database, you cannot recover the password or reopen the database.

Using the *CreateWorkspace* Method

The `CreateWorkspace` method enables you to create a new `Workspace` object. This new object can be used for either the Jet database engine or an ODBC type of connection. The syntax for the `CreateWorkspace` method is:

```
Set workspace = CreateWorkspace(name, user, password, type)
```

Table B4.4 describes the parameters of the `CreateWorkspace` method.

Table B4.4 *CreateWorkspace* Method Parameters

Parameter	Description
`Workspace`	An object variable that represents the `Workspace` object that you want to create.
`Name`	A String that uniquely identifies the newly created `Workspace` object.
`User`	A String that uniquely identifies the owner of the new `Workspace` object.
`Password`	A String that contains the password for the new `Workspace` object. The string can include any ASCII character, with the exception of `0` (`NULL`), up to 14 characters. To clear a password, set the `NewPassword` argument of the `NewPassword` method to a zero-length String (`""`).
`Type`	Optional. This indicates the type of `Workspace`. To create a Jet `Workspace`, use `dbusejet`. To create an ODBC `Workspace`, use `dbuseODBC`. If the `type` argument is omitted, the `defaulttype` property of `DBEngine` will determine the type of datasource to be used.

NOTE You can have more than one Workspace open at a time, such as a Jet `Workspace` and an ODBC `Workspace`.

The code segments in Listings B4.8 and B4.9 show how to use the `CreateWorkspace` method to create a Jet `Workspace` and an ODBC `Workspace`.

Listing B4.8 jetwrk.txt—Creating a Jet *Workspace*

```
' Create a Jet Workspace
Dim workJet as Workspace
Set workJet  = CreateWorkspace("JetWorkspace", "admin", "Password", dbUseJet)
Workspaces.Append workJet
```

Listing B4.9 odbcwrk.txt—Creating an ODBC *Workspace*

```
' Create a ODBC Workspace
Dim workODBC as Workspace
Set workODBC  = CreateWorkspace("ODBCWorkspace", "admin", "Password", dbUseODBC)
Workspaces.Append workODBC
```

If you want to delete a `Workspace` from the `Workspaces` collection, you first need to close all open databases. Second, you need to issue a `Close` method on the actual `Workspace` object, as described in the following lines of code:

```
workODBC.close
workJet.close
```

Using the *CreateUser* and *CreateGroup* Methods

The `CreateUser` method enables you to create a new `User` object. This applies only to the those `Workspaces` that use the Microsoft Jet engine. The syntax for the `CreateUser` method is

```
users = object.CreateUser(name, pid, password)
```

Table B4.5 describes the parameters of the `CreateUser` and `CreateGroup` methods.

Table B4.5 *CreateUser* and *CreateGroup* Method Parameters

Parameters	Description
`user`	This object variable represents the `User` object that you want to create.
`object`	This object variable represents the `Group` or `Workspace` object that relates to the new `User` object.
`name`	This optional Variant uniquely identifies the new `User` object.

continues

Table B4.5 Continued

Parameters	Description
`Pid`	This optional Variant is contained in the pid of a user account. This identifier can be anywhere from 4 to 20 alphanumeric characters in length.
`Password`	This optional Variant contains the password for the new `User` object. The password can be up to 14 characters in length. To clear a password, set the `NewPassword` argument of the `NewPassword` method to a zero-length string (`""`).

The code segments in Listings B4.10 and B4.11 show how to use the `CreateUser` method and the `CreateGroup` method to create new users and groups.

Listing B4.10 newuser.txt—Creating a New User in a Jet Workspace

```
' Create and append a new user
Dim usernew as User
Set userNew = .CreateUser("NewUser")
userNew.PID = "PID123456789"
userNew.Password = "NewPassword"
.Users.Append userNew
```

Listing B4.11 newgrp.txt—Creating a New Group in a Jet Workspace

```
'Create and append a new group
Dim groupNew as group
Set groupNew  = .CreateGroup("NewGroup", "PID123456789")
.Groups.Append  groupNew
```

Putting the Pieces Together

So far you have seen how to create a logon screen that accepts a user ID and password and to create a secure database. The next step is to actually combine them to create a usable application.

Listing B4.12 shows how you might create a Logon screen that attaches to a secure database. The logon screen checks to make sure that the user ID and password are entered in the test database. Although this example is not as robust as it could be, it does illustrate how easy it is to create a secure gateway to your application.

Listing B4.12 Logonscrn.txt—Combining a Logon Screen with Database Connectivity

```
Option Explicit

Public wrkJet As Workspace
Public DB As Database
Public RS As Recordset
Public userid_Login As String
Public userid_password As String
Public LoginSuccessfull As Boolean

Private Sub cmdCancel_Click()

    'set the global var to false
    'to denote a failed login
    LoginSuccessfull = False
    Me.Hide
End Sub

Private Sub cmdOK_Click()

    Dim db_prefix As String
    Dim db_password As String

    ' prefix + password to open secured database as required
    ' by the connect portion of the opendatabase()
    db_password = "secure"
    db_prefix = ";pwd="

    ' Create Microsoft Jet Workspace object.
    Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)

    ' Open Database object from saved Microsoft Jet database
    ' for exclusive use.
    Set DB = wrkJet.OpenDatabase("C:\testdata\secure.mdb", False,
    ➥ True, db_prefix + db_password)

    ' openrecordset as a snapshot ordered by keyvalue.
    ' this will allow use to grap a image of the table without another
    ' user affecting our data.
    ' we can also assign the various fields brought out by the SQL statement
    ' to global vars for latter use by the application.
    Set RS = DB.OpenRecordset("SELECT User_ID, Password " & __
        "FROM Security ORDER BY User_Id", dbOpenSnapshot)
    Do While True
        userid_Login = txtUserName
        userid_password = txtPassword

        ' assign criteria string for use by rs.findfirst
        userid_Login = "User_Id = '" & userid_Login & "'"
        userid_password = "Password = '" & userid_password & "'"
        '
```

continues

Listing B4.12 Continued

```
        With RS
             RS.MoveFirst
            ‘ Find first record satisfying search string. Exit
            ‘ loop if no such record exists.
            RS.FindFirst userid_Login
            If .NoMatch Then
                MsgBox “No records found with “ & userid_Login
                ➥ & “.”, , “Login”
                txtUserName.SetFocus
                Exit Do
            Else
                RS.MoveFirst
                RS.FindFirst userid_password
                If .NoMatch Then
                    MsgBox “ You were not sucessfuly logged in!”, , “Login”
                    txtUserName.SetFocus
                    LoginSuccessfull = False
                    Exit Do
                Else
                    MsgBox “You have sucessfuly logged in!”, , “Login”
                    LoginSuccessfull = True
                    Exit Do
                End If
            End If
        End With
    Exit Do
    Loop
  ‘ close down files
  RS.Close
  DB.Close
End Sub
```

Security, the Internet, and You

Okay, suppose that you now have an application that requires users to enter their password before they can log on to the application. This password is then verified by a secure database that determines what type of functionality the user is to have. The program works flawlessly; you have had no security breaches and are feeling fairly confident with the integrity of your system and data. You've decided to take the big plunge and put your application on the Internet for all to use. Your system should be perfectly safe from intruders and hackers, right?

Unfortunately, the correct answer is almost always NO!

The Internet is not only a giant network but it's also a giant desktop environment. Due to the very nature of the Internet, applications that are secure in their closed systems are vulnerable on an open system such as the Web. So the question remains: How do you launch secure applications on the Internet?

The answer to that question is many fold. First, your application needs to be secure in a standard environment, such as your local network or your desktop. If you have implemented some of the suggestions previously mentioned in this chapter, your application has a better-than-average chance of surviving the Internet. Second, you need to provide a level of security for the basic elements of the application itself. This could be the individual controls that make up your Internet application, whether they be Java applets or ActiveX controls. And finally, you need to provide a level of security for the network that currently runs the application over the Internet. This type of security is usually implemented via firewalls and proxy servers. The topic of proxy servers is beyond the scope of this book; however, several good books by Que detail how to fully utilize this level of security:

- For more information about setting up and configuring proxy servers, see the Que book *Special Edition Using Microsoft BackOffice,* Volumes 1 and 2 (ISBNs 0-7897-1142-7 and 0-7897-1130-3).

In the following sections, you look at how you can provide a high level of security for the controls in your Web application.

Why You Should Control Security

Even if you have implemented the previously described security procedures, you've yet to deal with the sensitive issue of network security. How can users who will automatically receive copies of your control be sure that the control won't harm their systems? To ensure users that you're a responsible control programmer, you must digitally sign the control's files. This enables users to contact you should the control misbehave.

If you don't digitally sign your controls, many Web browsers will refuse to download them onto the users' systems. Microsoft Internet Explorer, for example, has three levels of security to deal with executable content that is not digitally signed. When set to the default security level (high), Internet Explorer refuses to accept the control. The next level down (medium) enables you to download insecure controls, but only after Internet Explorer gives you a chance to refuse the control (see Figure B4.3).

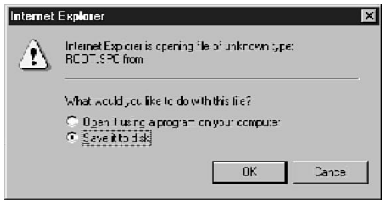


FIG. B4.3
You can set Internet Explorer's security so that you're warned about unsafe content.

The lowest level of security actually provides no security at all. This level allows all unsafe content to be loaded, without warning. Obviously, most users avoid this setting. If you'd like to experiment with your controls and Internet Explorer, you can find Internet Explorer's security setting by choosing View, Options. When you do, the Options dialog box appears. Click the Security tab to move to the security settings (see Figure B4.4) and then click the Safety Level button. The Safety Level dialog box appears (see Figure B4.5), from which you can select the security level you want. Other Web browsers have similar security options.

FIG. B4.4
The Options dialog box enables you to customize your copy of Internet Explorer 3.01.

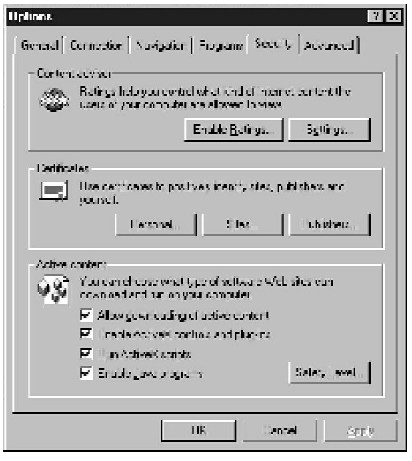


FIG. B4.5
Internet Explorer supports three levels of security to deal with unsafe content.

Determining Who Needs to Control Safety

Before Java applets and ActiveX controls entered the picture, security on the Internet wasn't quite as big a deal. After all, for the most part, you had complete control over the files you downloaded. You could take precautions to ensure that the files that you did download were safe.

With things such as ActiveX controls and Java applets, though, you have very little control over the downloading process, because these objects are downloaded automatically onto your computer. Worse, they are executed automatically. Imagine an ActiveX control that looks like an animated button but it erases your hard drive when clicked. Gives you the shivers, doesn't it?

Yes, you can set your Web browser so that it refuses to download executable content such as ActiveX controls. But then you will be unable to view many of the best Web pages on the Internet. After all, the whole point of ActiveX controls and Java applets is to give Web pages more power and pizzazz. It doesn't make sense to lock out such important enhancements when alternatives are available.

About Java Applets and Security

From a security standpoint, Java applets are much easier for programmers to handle than are ActiveX controls. First, the Java language doesn't include any commands that can access protected parts of memory. To ensure that the Java applet hasn't been modified to override these restrictions, each applet is verified by the Java interpreter before the applet is displayed in the user's Web browser. Moreover, Java applets are run under very strict file-access rules. In fact, under normal circumstances, a Java applet cannot access files at all. As a programmer, you don't have to do anything to assure the user that your applet is safe. Java itself guarantees this safety.

CAUTION

To say that Java guarantees an applet's safety is perhaps a bit strong. Many security leaks in Java have come to light in recent months. Sun Microsystems is working hard to plug these leaks and increase Java's reliability. But, as anyone who has used a computer knows, anything that can be done can be undone. There are never any absolute guarantees.

About ActiveX Controls and Security

Microsoft felt that the restrictions forced on Java applets were too severe and that, while preventing mischief, they also prevented the programmer from creating truly powerful, executable content for Web pages. Microsoft's security philosophy is that an ActiveX control should be able to do just about anything that a stand-alone application can do. Rather than cripple the control, Microsoft came up with the idea of *digital signing*. When you digitally sign a control, you tell the eventual user of the control who you are and where you can be found. Then, if your control causes havoc with the user's system, he has a way to contact you.

More importantly, Microsoft Internet Explorer can be set up to refuse all executable content that has not been digitally signed. That guy who wants to erase your hard drive with his sneaky button isn't likely to identify himself with digital signing. So he sends his control out onto the Internet unsigned. When your ActiveX-capable browser downloads a Web page containing the control, the browser sees that the control is unsigned and refuses to accept it.

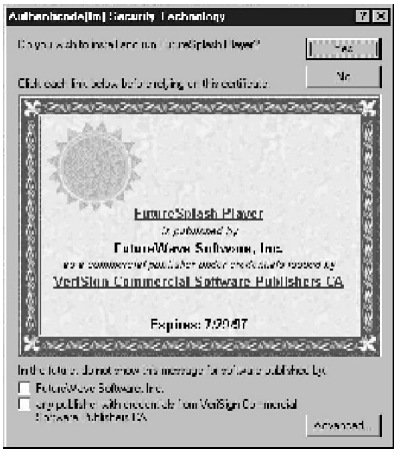
Digital signing is one way Microsoft makes using controls safer. Digital signing, however, doesn't guarantee that a control is safe. It just gives a user a way to find the responsible party should problems arise from using the control. Digital signatures, like fingerprints, cannot be faked; the signature is unique to the person or corporation it is registered to. Two other types of control certification also document a control's safety: Safe for Initialization and Safe for Scripting. These two control-certification types are discussed in Chapter 24, "Creating ActiveX Controls."

Digitally Signing Your ActiveX Controls

Digitally signing your ActiveX control gives the user recourse if the control damages the user's system. Because having a verified identity for the control's programmer is so important, digital signing requires that you work with a *certificate authority* which verifies your identity and issues a digital certificate to you. You use tools provided by the certificate authority, or included with the ActiveX SDK, to include your digital certificate with all your controls.

When a user's browser is about to install the new component, the certificate appears, telling the user who is responsible for the control and who issued the certificate. Figure B4.6 shows a digital certificate. The certificate is for an ActiveX component called FutureSplash Player, and the certificate is issued by VeriSign Commercial Software Publishers. A certificate is sometimes referred to as an X.509 Certificate.

FIG. B4.6
Digitally signing an ActiveX control includes a certificate with the control.



Before accepting the new ActiveX component, the user can click the program name to get more information about the component. When the user clicks the program name, a second instance of the browser runs and connects to the Web page containing the information (see Figure B4.7). Finally, the user also can get information about the company that issued the certificate by clicking the company's name in the certificate. Again, another instance of the Web browser runs and connects to the Web page that displays the requested information (see Figure B4.8).

FIG. B4.7
Users can get additional information about the ActiveX component that is about to be installed.

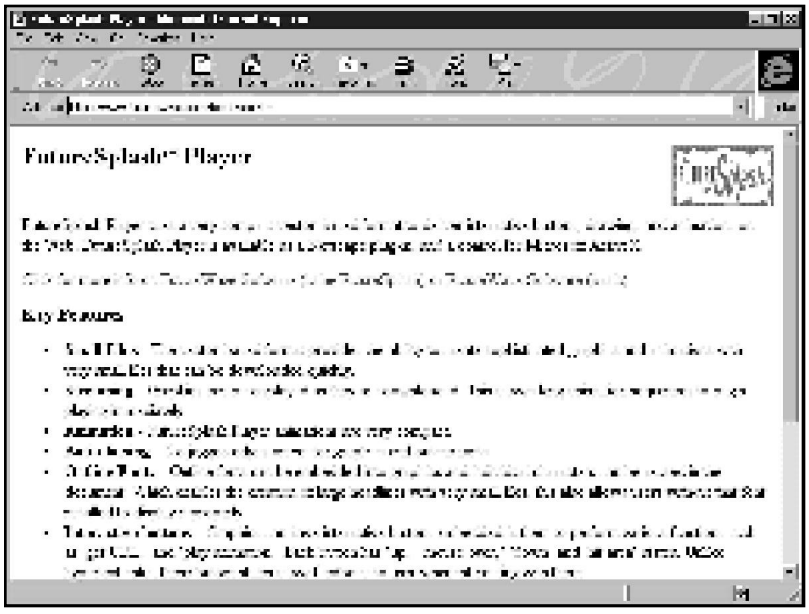
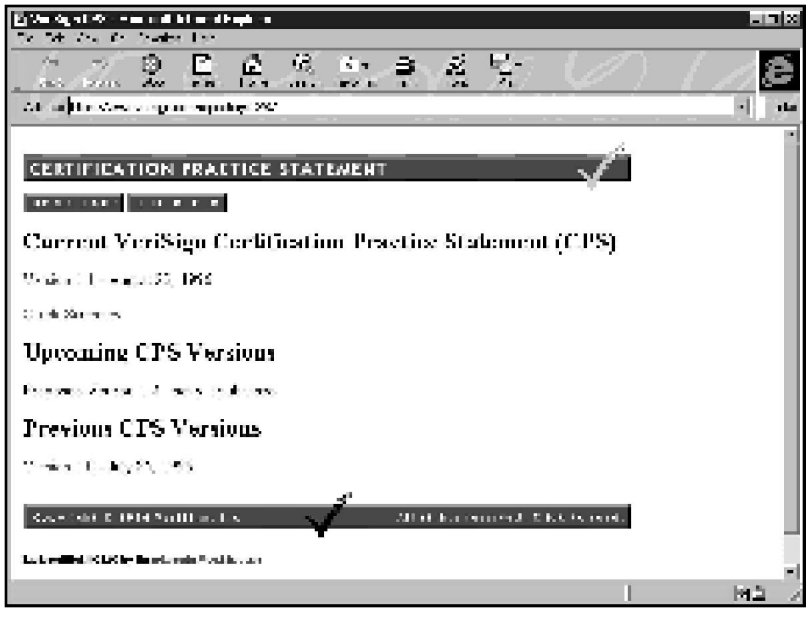


FIG. B4.8
Users can also get information about the company that issued the certificate.

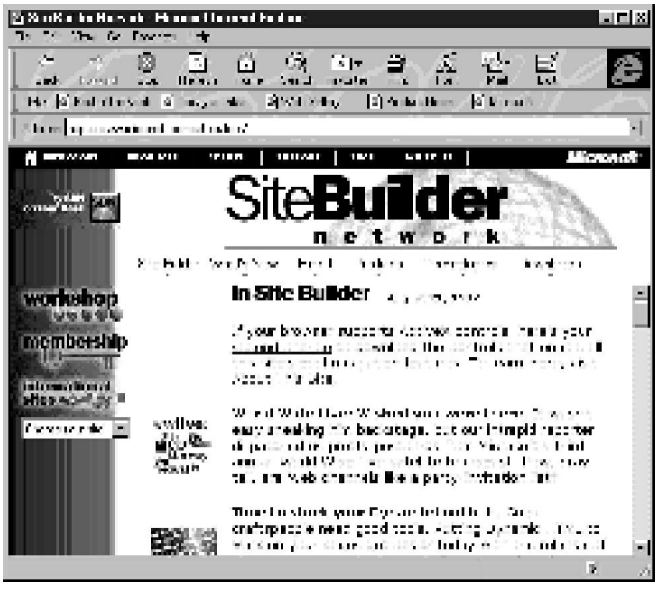


Getting the Right Stuff

All the software tools necessary for this chapter can be downloaded from Microsoft at **http://www.microsoft.com/intdev/sdk/** or at **http://www.microsoft.com/sitebuilder/**.

Microsoft developed the Site Builder Network to provide controls and cutting-edge software to Web developers like you and me. Figure B4.9 shows the opening screen from the Site Builder Network.

FIG. B4.9
Microsoft's Site Builder Network home page is a great place to get the hottest ActiveX controls.



From this site, you can download the ActiveX SDK (Software Development Kit). The SDK contains tools necessary to sign your ActiveX controls, as well as tools for those developers interested in developing ActiveX components from other platforms such as Macintosh and RISC. The SDK includes samples and help files to get you started and to answer any questions that might arise while you are working with the tools. The ActiveX SDK includes not only the tools to perform digital signing, but also test certificates that you can use for testing your control's Internet download package.

http://www.microsoft.com/msdownload/sbndownload/sbnaxsdk/sbnaxsdk.htm
To obtain the ActiveX SDK, you need to point your browser to this Web address.

You are presented with a page that asks you to pick your choice of a download site. For those of you with slow modems, you might want to plan this download over lunch. A connection to the Web at 28.8Kbps will take over 1.5 hours to download the 8+M file.

After you have successfully downloaded this mammoth file, you need to type the following on the command line:

```
c:> ActiveX -d
```

NOTE Make sure that you have saved the ActiveX.exe to the root directory on your hard drive. ■

This extracts the necessary files and creates the required subdirectories. After successfully completing the installation process, you can access all the tools from the \INETSDK directory on your hard drive.

NOTE The fully expanded SDK requires that you have at least 20M available on your hard drive. ■

Digital Signing Overview

Before you continue, getting a quick overview of the digital-signing process might be helpful. On its Web site, Microsoft lists six steps you need to follow to digitally sign your controls:

1. Download the latest version of Internet Explorer 3.01. You can find this file at **http://www.microsoft.com/ie/download**.
2. Apply for credentials from a certificate authority. You can get instructions on this step by aiming your Web browser at **http://www.microsoft.com/intdev/security/authcode/certs.htm**. You also need to visit the certificate authority's Web site to obtain an online certificate application. You can choose to apply for an Individual Software Publisher certificate or Commercial Software Publisher certificate.
3. Get the latest version of the ActiveX SDK. You can get the SDK from **http://www.microsoft.com/intdev/sdk**. The ActiveX SDK includes code-signing documentation.
4. Prepare your files for signing. For the CAB files that you create with the Application Setup Wizard, you need to add `.Set ReservePerCabinetSize=6144` to your .ddf file before creating the CAB file.
5. Sign your files. Run the signcode tool to digitally sign your files. When you run the tool without any parameters, a wizard will guide you through the process.
6. Test your signature by running the chktrust tool. If your digital signing worked successfully, you'll then see your certificate.

TIP Included in the ActiveX SDK is Webpost.exe, a Web page posting utility that is self-installing. This utility enables you to post one or more Web pages to a specific URL address. The Webpost utility is located in the \INETSDK folder.

Using Digital Signing

As mentioned earlier, your code needs to be signed to be accepted by the default implementation of Internet Explorer. Take a look at a few utilities that will help you do just that.

Microsoft's Authenticode, provided with the ActiveX SDK, contains the utilities to sign your application. Included in the SDK are a sample certificate and a private key designed to aid in the testing process of your certificates.

NOTE If you intend to sign your code for commercial purposes, you need to obtain a valid certificate from GTE or VeriSign before you sign any code for public distribution.

Using the *MakeCert* Utility The MakeCert.exe utility enables you to create a test certificate. This utility allows you to create an X.509 certificate that binds your name to a public key.

The command-line syntax for the MakeCert utility is:

```
MakeCert [options] outputCertificateFile
```

Table B4.6 shows what options can be set for the `MakeCert` utility.

Table B4.6 Command-Line Options for *MakeCert*

Option	Description
`-u:subjectKey`	Specifies the publisher keypair name. If none is found, it is created.
`-U:subjectCertFile`	Certificate with existing public key to use. Indicates the file name to use.
`-k:subjectKeyFile`	Location of subject's private-key (.pvk) file.
`-n:name`	Designates the certificates X.509 name (for example, `"CN=Missy Carlson"`).
`-d:displayname`	Designates name of the publisher displayed.
`-s:issuerKey(File)`	The location of issuer's key; a default to root key is provided for testing purposes.
`-i:issuerCertFile`	Indicates the location of the issuer's certificate.
`-#:serialNumber`	Indicates the serial number of the certificate. This is an optional value. The maximum value is two to the 31st power (2^{31}). The default value generated is guaranteed to be unique.
`-l:policyLink`	Indicates a hyperlink to SPC Agency policy info (for example, an URL).
`-I`	Explicitly specifies the certificate is allowed for use by individual software publishers.
`-C`	Explicitly specifies the certificate is allowed for use by commercial software publishers.
`-C:f`	Indicates that the publisher has met the minimal financial criteria.
`-S:session`	Indicates the session name for the enrollment session.

Option	Description
-P:purpose	Indicates why the certificate is to be generated CodeSigning (default) or Clientauth.
-x:providerName	Indicates what CryptoAPI provider to use.
-y:nProviderType	Indicates the CryptoAPI provider type to use.
-K:keyspec	Designates the key. Possible values are 'S' signature key (default) and 'E' key-exchange key.
-B:dateStart	Specifies the start date of the validity period. The default is certificate generation date.
-D:nMonths	Indicates the duration of the validity period.
-E:dateEnd	Indicates the end of the validity period; defaults to the year 2039.
-h:numChildren	Indicates the maximum number of certificates on the tree below this certificate.
-t:types	Indicates certificate type: either/both of 'E'nd-entity; 'C'ertification authority.
-r	Creates a self-signed certificate.
-m	Uses MD5 hash algorithm (default).
-a	Uses SHA1 hash algorithm.
-N	Includes the Netscape client authentication extension.

CAUTION

You cannot make a valid signed certificate by using the MakeCert.exe and cert2spc.exe. These utilities are provided for testing purposes only until you receive your real certificate from an authorized certificate authority.

The previously mentioned flags and notes are also visible by typing **MakeCert.exe** without any additional parameters.

For the -u and -k options, if the indicated subject's key (key pair) cannot be found, then it is created. For -u, it is created in the CryptoAPI keyset; for -k, it is created in a file.

Alternatively, the subject public key can be obtained from an already existing certificate by using the -U option. -U changes the default subject name to be the same as that of the indicated certificate.

A typical usage of MakeCert is:

```
c:>MakeCert -u:myKey -k: myKeyfile -n:CN=mySoftwareCompanyName myCert.cer
```

This generates a certificate file called `myCert.cer`. The public part of the key pair called `myKey` is bound to the publisher `mySoftwareCompanyName`. If the keypair `myKey` does not already exist, it is generated along with `myKeyfile`.

CAUTION

The certificate that you just created by using MakeCert is *not* a valid certificate. This certificate is designed for test purposes only until you receive a valid X.509 certificate from a valid CA.

On receipt of a valid X.509 certificate, use of the MakeCert utility is no longer needed.

Using the *Cert2SPC* Utility The next step required to digitally sign code is to create a Software Publisher Certificate (SPC) with the `Cert2SPC` program. This program combines the X.509 certificate and the root certificate into a PKCS#7 signed-data object. PKCS#7 objects allow for the inclusion of several certificates into a single object.

The syntax for `Cert2SPC` is

```
Cert2SPC cert1.cer cert2.cer. . .certN.cer output.spc
```

where `cert1. . .certN` are the names of the X.509 certificates.

`output` is the name of the SPC. This is a PKCS#7 object containing the X.509 certificate and the root certificate.

Here is an example:

```
c:>Cert2Spc root.cer mycert.cer . . . mycert.spc
```

This combines `mycert.cer` and `root.cer` to make an SPC called `mycert.spc`.

CAUTION

The SPC `mycert.spc` that you just created is *not* a valid SPC. You need to obtain a valid SPC from a certificate authority before you actually begin to sign your code.

Using the SignCode Utility The last step needed to digitally sign code is to use the SPC to actually sign a file. This is accomplished with the SignCode program. This program does the following:

- Creates a cryptographic digest of the file
- Signs the digest with your private key
- Extracts the X.509 certificates from the SPC
- Creates a new PKCS#7 signed-data object that contains the serial numbers of the certificates and the signed digest information
- Embeds the object into the file

If you have a valid SPC, you can use the SignCode utility to actually sign your code. The SignCode program has a wizard to help you do this. To sign code by using the wizard, simply type **SignCode** without any options. If you want to sign your code manually, the syntax is

```
SignCode [-prog filename -spc credentials -pvk privateKeyFile ] [options]
```

Table B4.7 lists the SignCode command-line parameters.

Table B4.7 SignCode Command-Line Parameters

Option	Description
`-prog`	The name of the file to sign.
`-spc`	The file that contains the credentials. This is usually an .spc file.
`-pvk`	The file containing the private key of the publisher. This is a .pvk file.

Table B4.8 lists the SignCode command-line options. You can include as many options as you need.

Table B4.8 SignCode Command-Line Options

Name	Description
`-name`	The name of your program.
`-info`	A location for obtaining more information about your program, such as an URL.
`-gui`	Invokes the code-signing wizard.
`-nocerts`	Indicates that you do not want any X.509 certificates embedded in the PKCS#7 signed-data object. In this case, the relevant certificates must already be stored on the client computer.
`-provider`	The name of the Cryptographic service provider to use.
`-provider`	The Cryptographic provider type to use.
`-commercial`	Indicates that the code was signed by a commercial software publisher.
`-individual`	Indicates that the code was signed by an individual software publisher.
`-sha`	Indicates that you want to use the SHA hashing algorithm.
`-md5`	Indicates that you want to use the MD5 hashing algorithm. This is the default hashing algorithm.
`-?`	Displays all the preceding options.

Here is an example of how to sign a file:

```
c:>SignCode -prog MyProgram.exe -spc Cert.spc -pvk MyKey
```

This embeds a PKCS#7 object, Cert.spc, into the digest of file, MyProgram. The digest is signed with the private key of the MyKey key pair.

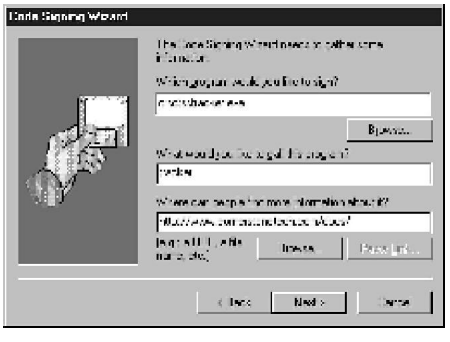
After this is done (assuming that you have a valid certificate), the file can be distributed to your customers.

The SignCode utility also allows for the use of a wizard to guide you through the process of digitally signing a file. To invoke the Code Signing Wizard, type **signcode** on the command line without any parameters. The main screen of the Code Signing Wizard, shown in Figure B4.10, is then displayed.

FIG. B4.10
The Code Signing Wizard displays the greeting screen when first launched.

Click the Next button and you see the Code Signing Wizard Setup screen, shown in Figure B4.11.

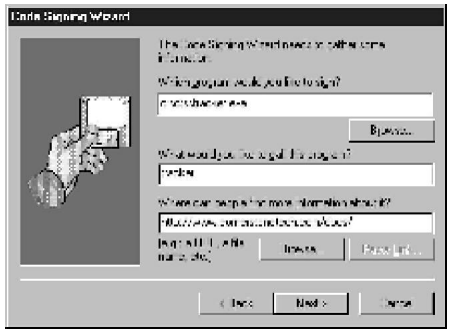
FIG. B4.11
The Code Signing Wizard Setup screen allows you to enter where additional information (such as documentation) is located.



This screen requests some background information on the program that you want to sign. This is where you enter the name and location of the program you are about to sign. You are also prompted to enter where additional information about the program can be found. After you have completed entering the information, you are presented with the screen in Figure B4.12.

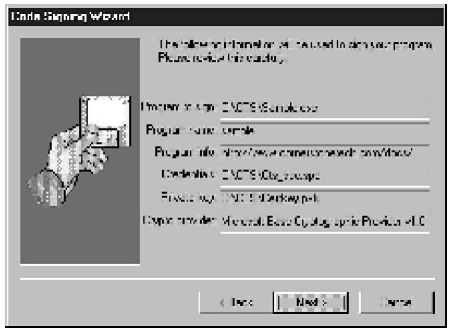
At this point, you need to fill in the credential information. You are also prompted for the location of the private key with which to sign the program and the hashing algorithm you want to use to create the digest from your program.

FIG. B4.12
The Code Signing Wizard credentials information screen requires that you have a private key to continue.



Clicking the Next button displays a review of the information that you have supplied, as shown in Figure B4.13.

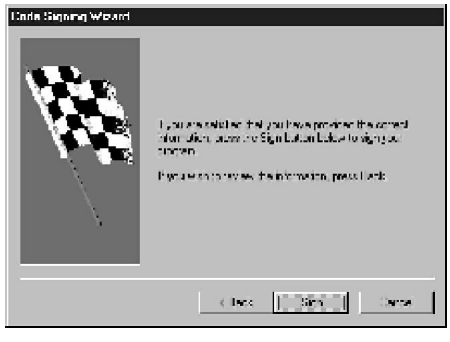
FIG. B4.13
At this point, you have a chance to make any last-minute changes before continuing.



This actually is a compilation of all the information that you have supplied in the previous steps. If you need to make any changes to any of the information presented, now would be a good time to do that. You can use the Back button to return to the previous screen.

After verifying that the appropriate information is displayed, click the Next button to view the final phase of the code-signing process, as displayed in Figure B4.14.

FIG. B4.14
The Code Signing Wizard Confirmation screen is you last chance to abort the process should you need to make any changes.



Clicking the Sign button starts the signing process. On successful completion, the Code Signing Wizard displays the screen as shown in Figure B4.15.

FIG. B4.15
Congratulations! You have successfully signed your control!

The only remaining task is to verify that the program actually was signed correctly.

Using the PeSigMgr Utility The PeSigMgr program checks to see if SignCode was successful. This means the file should have a PKCS#7 object embedded in it. Here is the syntax:

```
PESIGMGR [options] signedfile
```

Table B4.9 contains a listing of the command parameters used with PeSigMgr.

Table B4.9 PeSigMgr Command-Line Parameters

Parameter	Description
`-l`	Lists all the certificates in an image.
`-a:<-filename ->`	Adds a certificate file to an image.
`-r:<-index ->`	Removes a certificate `<-index ->` from an image.
`-s:<-filename ->`	Used with `-r` to save the removed certificate.
`-t:<-CertType ->`	Used with `-a` to specify the type of certificate, where CertType can be X509 or PKCS7. The default is PKCS7.
`-?`	Displays all the options.
`Signedfile`	The name of the signed file you want to check.

To check for the existence of signed code within the MyProgram.exe file, execute the following statement:

```
c:>PeSigMgr -l MyProgram.exe
```

If a certificate is found in MyProgram.exe, PeSigMgr returns the number, revision, and type of the certificate. A sample response is

```
>Certificate 0 Revision 256 Type PKCS#7
```

NOTE PeSigMgr does not provide for any signature verification. Its sole purpose is to check that SignCode actually did its job. ■

Using the Chktrust Utility The purpose of the ChkTrust program is to check the validity of the signed file. The syntax is

```
CHKTRUST [switches] signedfile
```

Table B4.10 lists all the ChkTrust command parameters.

Table B4.10 ChkTrust Command-Line Parameters

Parameter	Description
`-c`	Designates a cabinet file
`-i`	Designates a PE image file
`-j`	Designates an ActiveX file

ChkTrust performs the following actions when invoked:

1. Extracts the PKCS#7 signed-data object.
2. Extracts the X.509 certificates from the PKCS#7 signed-data object.
3. Computes a new hash of the file and compares it to the signed hash in the PKCS#7 signed-data object.
4. Verifies that the signer's X.509 certificate points back to the root certificate and that the correct root key was used.
5. Verifies that the code has not been modified or tampered with, and that the vendor was authorized to publish the file by the root authority. After the proper verification is made, ChkTrust returns a `0`.

To perform validity checking on a file, execute the following line:

```
ChkTrust MyProgram.exe
```

On successful completion, ChkTrust returns

```
Result: 0
```

If for some reason the validity checking is not successful, the return code is a six-digit number representing the signature of the file that you are checking.

From Here...

This chapter looked at ways to provide security to your applications on several different levels. It explains how you can provide security to your application through the use of a simple logon screen that can easily be scaled to whatever needs you have. The chapter also explores ways to protect your data by using simple methods available to the Visual Basic programmer. Finally, this chapter shows how easy it is to digitally sign your Active X controls for use on the Web.

For more information on ActiveX controls, you might want to check out the following chapters in this book:

- Chapter 24, "Creating ActiveX Controls," explains in detail how to build your first ActiveX control. You also are introduced to a few special items designed to make the process easier.
- Chapter 25, "Extending ActiveX Controls," takes you to the next step in your ActiveX education by showing how to add additional functionality to your base control.

For more information on database programming, the following chapters might be helpful:

- Chapter 28, "Building Database Applications," walks you through the steps needed to build a database application with data controls.
- Chapter 33, "Database Access with ActiveX Data Objects (ADO)," explains in detail how to connect your ActiveX control to a database by using the ADO.