PINNACLE

FoxTalk

Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

The Kit Box

More Than One Way to Skin a Fox

Andy Kramek and Marcia Akins

7.0 8.0 DOWNLOAD

This month, Andy Kramek (with the trusty help of Marcia Akins) is looking for a way to make his forms more appealing. Despite VFP's enormous flexibility in most things, it's actually rather limited when it comes to the appearance of the user interface. Fortunately the FoxPro community, in the shape of Pablo Molina of La Rioja, Argentina, comes to his rescue with a very cool implementation of "Skins" for Visual FoxPro forms.

Andy: I've just been creating a whole bunch of forms for a new application I'm working on and I'm getting very bored.

Marcia: What do you mean "bored"? I know you prefer playing in the database to working with UIs, but that's no excuse for saying it's boring.

Andy: Not with doing it (though it is a challenge at times), but with the results. Everything is in the standard Windows "American Civil War" colors—just blue and gray! It's so uninteresting.

Marcia: Well, you can change your Windows color scheme easily enough. There are all sorts of different color sets that ship as standard, and you can even create your own. Just right-click on the Windows desktop, choose Properties from the pop-up menu, and then go to the Appearance tab. You can define exactly how the various UI components will look. If you're running Windows XP (and are using VFP 8.0) you can even use "themes" within your application.

Andy: But all these things (including themes) do is affect all components

September 2003

Volume 15, Number 9

8.0 Applies to VFP v8.0
7.0 Applies to VFP v7.0
6.0 Applies to VFP v6.0
5.0 Applies to VFP v5.0
3.0 Applies to VFP v3.0

DOWNLOAD

Accompanying files available online at www.pinnaclepublishing.com/ft

UNIX MAC DOS WIN

Applies specifically to one of these platforms

Securing VFP Data

Alf Borrmann

What VFP lacks in comparison to the big iron databases are special security mechanisms. Everyone who has a copy of VFP or who can deal with ODBC may manipulate your data without running your business rules. Alf Borrmann shows you how you can store your VFP data in a Windows network environment securely using standard Windows routines.

SINCE version 3.0, VFP has been a real entity relationship database engine that supports stored procedures, referential integrity, triggers, and more. The only difference (besides the limited DBF file size) between VFP and the big iron databases like Oracle or Microsoft SQL Server is the lack of security integrated into the engine (well, there are some other things, but security matters most).

If you develop a VFP application that has to access data located in a VFP database, you have to store the required files in a directory where the application's users can see them. This means that users who are able to start the VFP application also have access to the data files. Even if they aren't capable of starting a VFP development version where they can BROWSE (or worse, ZAP) your data, they at least have access via ODBC, and can manipulate the data in Excel or other ODBC-enabled applications. Sometimes this leads to VFP being excluded from the list of acceptable applications in some enterprises.

But if you're allowed to use Windows in these enterprises, it should be completely acceptable to secure the data using standard Windows mechanisms to stay in business. Let's take a look at those mechanisms, and I'll show you how to use these within a VFP application. All you need is VFP and a few Windows API calls—so everything you need should already be installed on your system.

The principle

The idea behind the technique I'll show you is to separate the user rights needed for starting the application from those user rights necessary for accessing its data. You have to configure one user account (or a group) that's capable of starting the EXE but can't see or even find the database files. You then need another account for read/write access to the data. This account isn't an equivalent to a "real" user. Set up this way, a user or the group member can start the application, but can't manipulate the database—and your data is pretty safe against getting accessed without your application, isn't it? But if the users can't see the data, how does your application? The key to achieving this is called "impersonation."

Perhaps you're already familiar with this concept from your Internet Information Service (IIS) that ships with Windows NT and higher: If you set up a folder to be accessible through the Internet, you can configure Anonymous access (in Administrative Tools click on Internet Information Service, find a Web site on your computer and open the Properties dialog, open the Directory Security page, and click on Edit for Enable Anonymous access). If you allow Anonymous access, no one connecting to this directory will be asked for a username or password. Instead, the Windows thread representing these connections has its own account. You can configure the account's credentials to be used as well. Set up this way, the access to this site is *impersonated* because Windows can't identify the actual user reading the data.

If the application running between the user and the data isn't IIS but your own VFP application, you can do the same—limit access to the directory containing the data to just one user account. Your application must reside in another directory from which a separate list of users can start it. When your application starts it may get the current user's information and verifies his rights. Then the program impersonates and accesses the data. This way you can manipulate the data from within your application, but no user can see it from outside.

How do you do it?

Here's how to accomplish this:

1. Create a directory to contain the data that the application will use.
2. Set up a new user account and make it the only one that has complete rights to this data directory (maybe you want to grant access to an additional *real* user like yourself or the administrator; that's completely okay).
3. Make your application "impersonation aware" by taking the following steps.

   Register these Windows API calls:

```
Declare Short LogonUser in Win32API;
 String lcNewUserName,;
 String lcDomainName,;
 String lcPassWord,;
 Integer lnLogonType,;
 Integer lnLogonProvider,;
```

```
 Integer @lnUserHandle

Declare Short ImpersonateLoggedOnUser;
 in Win32API;
 Integer lnUserHandle

Declare Integer WNetGetUser in Win32API;
 String @lcName,;
 String @lcUser,;
 Integer @lnBuffersize

Declare Short RevertToSelf in Win32API

Declare Short CloseHandle in Win32API Integer
```

Now you're ready to impersonate your application.

First, try to logon the user you configured for accessing the data. The logon type and the logon provider can be standard values of 3 (logon to a Win2000/XP server) and 0 (NTLM), respectively. For more information, refer to the Windows API documentation:

```
lcNewUserName = <name of the account that has;
 access to the data>
lcDomainName = <name of the domain the;
 account belongs to>
lcPassWord = <you're guessing, right>
lnUserHandle = 0

lnSuccess = LogonUser( lcNewUsername,;
 lcDomainName,;
 lcPassWord,;
 3,;
 0,;
 @lnUserHandle)
```

This call creates a new user handle and stores it to lnUserHandle. The return value of this call is a non-zero value if the call succeeded, and a value of zero if it didn't. But beware: The return value only says whether the call was made without errors. No errors simply means that it did find a logon type and a logon provider for the values specified. The returning value doesn't tell you whether the logon itself succeeded. The function stores a handle (simply an integer value) to the variable given as the sixth parameter. You can use this value to impersonate the user represented by the handle:

```
lnSuccess = ImpersonateLoggedOnUser( lnUserHandle)
```

This call also returns a value based on whether it succeeded. But again: This also doesn't give you information about whether the impersonation itself was achieved. You need to test the newly logged in user by getting the current session's username:

```
lcName = chr( 0)
lnBufferSize = 64
lcUser = Replicate( lcName, lnBufferSize)

If WNetGetUser( @lcName, @lcUser, @lnBufferSize) = 0
  lcUser = Left( lcUser, At( Chr( 0), lcUser) - 1)
  llSuccess = lcUser = lcNewUserName
Endif
```

If the logon did fail, the session is typically running under the "Guest" login. By comparing the current username with the desired name, you can verify that everything went as planned. If something went wrong, you can simply return to the user session that ran previously by calling the RevertToSelf function:

```
If !llSuccess
  * show some error messages here
  RevertToSelf( )
Endif
```

There may exist cases where you have to access resources that aren't accessible to the special user account you've configured. Using these resources can be accomplished in several ways. First, you can call the RevertToSelf function before accessing the resource and invoke the ImpersonateLoggedOnUser function for getting back. Second, you can have more than one user handle representing several user accounts. Those accounts could belong to different user groups that define the rights needed for the resources. You can switch between the different accounts with subsequent calls of ImpersonateLoggedOnUser.

After using the handles, you should release them by calling the CloseHandle function:

```
CloseHandle( lnUserHandle)
```

Tips and traps

Be aware that all of the code that runs after the impersonation only has rights limited to the user account you created. This is also true for all calls you have to make to OLE servers you're instantiating and the OLE server objects themselves.

As a VFP developer, you know that it's not a good practice to store usernames or passwords in your code. But if they identify an important user account, where else should you store them? One possibility is to use an INI file. Yes, using the encryption functionality that's buried in the Windows API you can securely save this information. For applying these functions, have a look at the _CryptAPI class in _Crypt.VCX of the FoxPro Foundation Classes.

One drawback to this approach is that inside of your application you can't identify the current user with built-in VFP functions like ID() or Sys(0). To get this information, you should create a user object that contains all of the information on the logged-in user *before* you impersonate (but since you're a good OO developer, you already did it this way). You may also create several account objects that isolate the utilization of special resources and their accounts.

For a secure application, you should consider protecting your code against decompiling. If someone can read your encryption strategy, he can figure out the key you're using for storing the user values in the INI file.

Conclusion

The technique shown pulls VFP in-line with other databases that handle security by themselves. In addition, it has the possible advantage of getting rid of an additional user management layer. You can simply manage your users via the built-in Windows mechanisms and the user interface the administrator already knows. ▲

Alf Borrmann is a .NET developer located in Berlin, Germany, who began his career in 1990 using FoxPro 1.0. Since then he's specialized in object orientation, project management, and teaching courses on software development. alf.borrmann@denk-modell.de.

Great Linux EULA Controversy...

Continued from page 8

computer on the Internet, and that second computer's CPU was running Microsoft Windows—would that satisfy the "in conjunction with" terminology?

Or maybe all you have to do is go out to eBay and buy a copy of Windows 95 for $1.99, and use the CD to prop up your system unit that's running your VFP app, the runtime, and Linux. In this case, the runtime would be definitely be operating "in conjunction with a Microsoft Windows platform."

The rest of the EULA has very explicit descriptions of what is and isn't allowed. Why is this issue being covered with evasive words? Why use vague terms when there are perfectly clear ways of saying what you want to say?

There are three reasons to be vague. The first is when you don't understand the question, and thus have to resort to waving your hands because you don't know how to answer. You see this in computer stores all the time, right?

The second is when you don't want to answer the question.

And the third is when you want to confuse the issue. Which of these three do you think is the reason Microsoft chose to provide a vague and thus potentially misleading answer instead of simply answering my questions?

```
As with any contract, you should seek your own legal
counsel's advice when interpreting your rights and
obligations under the Visual FoxPro End User License
Agreement.
```

I see. So Microsoft is telling its customers that it's not going to answer their questions about its EULA, but, instead, the customer has to hire a lawyer to do so. How many companies can afford to treat their customers this way? ▲

DOWNLOAD **EULAEMAIL.TXT at www.pinnaclepublishing.com/ft**

Whil Hentzen, the Editor of *FoxTalk*, is president of Hentzenwerke Corporation (www.hentzenwerke.com), an 18-year-old firm that specializes in developing strategic database applications for Fortune 2000 firms in the manufacturing, financial, and healthcare industries. He also owns Hentzenwerke Publishing, a technical book publisher that specializes in high-end software development topics. He spends his spare time with his kids and is an avid runner. whil@hentzenwerke.com.

The Kit Box...

Continued from page 5

file (available at www.pinnaclepublishing.com/ft) includes this functionality, but it does assume that the only text files in the current working directory are valid skin configuration files. In reality we'd need to be a bit more sophisticated about this, but we're running out of space in this article, so we'll look at some implementation possibilities next time.

Andy: I have to say that we owe Pablo a huge vote of thanks. Despite the little nitpicks we've made, this is a tremendous piece of work on his part and I'm really looking forward to using skins to make my applications more interesting for my users. ▲

DOWNLOAD **09KITBOX.ZIP at www.pinnaclepublishing.com/ft**

Andy Kramek is a long-time FoxPro developer, FoxPro MVP, independent consultant, and joint owner of Tightline Computers Inc., based in Akron, OH. A veteran conference speaker, he has published widely and can be found online in the CompuServe forums (http://go.compuserve.com/msdevapps) and the Virtual FoxPro Users Group (www.vfug.org). andykr@compuserve.com.

Marcia Akins is a FoxPro MVP, independent consultant, and joint owner of Tightline Computers Inc., based in Akron, OH. A veteran conference speaker, she has published widely and is well known for her contributions to CompuServe (http://go.compuserve.com/msdevapps) and the Universal Thread (www.Universalthread.com). marciagakins@compuserve.com.

September 2003 Downloads

- 09KITBOX.ZIP—Source code to accompany Andy Kramek and Marcia Akins' column, "The Kit Box: More Than One Way to Skin a Fox"
- EULAEMAIL.TXT—Copy of the e-mail referenced in Whil Hentzen's Editorial, "The Great Linux EULA Controversy."
- 09DHENSC.ZIP—Source code to accompany Doug Hennig's article, "Base Classes Revisited."
- 09MCNESC.ZIP—Source code to accompany Paul McNett's article, "Exploring Python from a Visual FoxPro Perspective."

For access to all current and archive content and source code, log in at www.pinnaclepublishing.com/ft and enter the User name and Password at right when prompted.

User name

Password

Editor: Whil Hentzen (whil@hentzenwerke.com)
CEO & Publisher: Mark Ragan
Group Publisher: Connie Austin
Executive Editor: Farion Grove
Production Editor: Andrew McMillan

Questions?

Customer Service:
Phone: 800-493-4867 x.4209 or 312-960-4100
Fax: 312-960-4106
Email: PinPub@Ragan.com

Editorial: FarionG@Ragan.com

Pinnacle Web Site: www.pinnaclepublishing.com

Subscription rates

United States: One year (12 issues): $159; two years (24 issues): $270
Other:* One year: $189; two years: $321

Single issue rate:
$20 ($25 outside the United States)*

* Funds must be in U.S. currency.

FoxTalk (ISSN 1042-6302)
is published monthly (12 times per year) by:

Pinnacle, A Division of Lawrence Ragan Communications, Inc.
316 N. Michigan Ave., Suite 400
Chicago, IL 60601

POSTMASTER: Send address changes to Lawrence Ragan Communications, Inc., 316 N. Michigan Ave., Suite 400, Chicago, IL 60601.