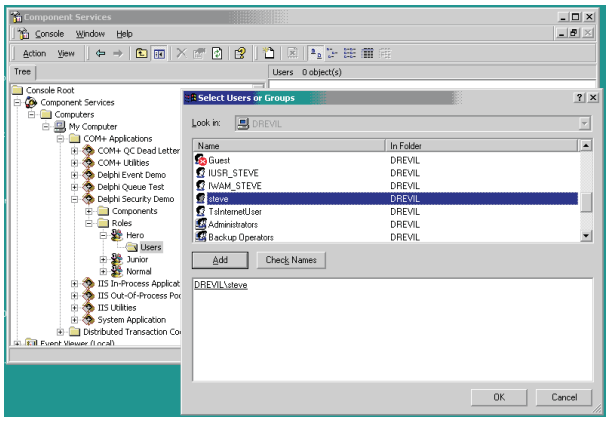
COM Corner: COM+ Security

by Steve Teixeira

As the introduction of one new technology quickly follows another in today's insanely paced world of software development, I occasionally reflect fondly on the old days of PC software development, when applications consisted of a .EXE or a .COM file and a network was a place to share data files with your co-workers. Business applications today often consist of multiple types of user interfaces (Windows, web-based, Java, etc) communicating with software components distributed across a network, which in turn communicate with one or more database servers on the network. As our ability to successfully tame multiple components into an application increasingly becomes a barometer for our success as developers, so does our ability to enable these components to communicate in a trusted environment. This means building security into distributed applications which enables components to know who is using them and what level of service they should offer.

The notion of security has become common sense at this point. We all understand that most data needs to be protected. For example, human resources data shouldn't be accessible to all, sales data shouldn't be accessible to your competitors, etc. Equally, component functionality also needs to be secure: perhaps only administrators should have the right to use certain objects, or only department managers should have access to a particular business rules engine. In practice, however, building this type of security into distributed applications can be a time-consuming process, and security features naturally take a back seat to core functionality in project schedules.

➤ *Figure 1*



➤ *Table 1*

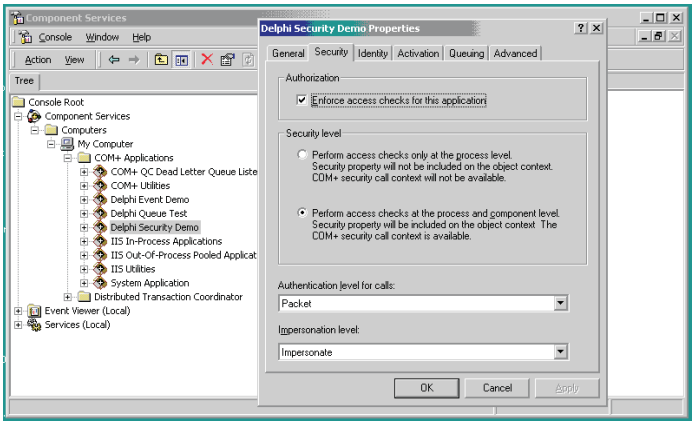
Authentication Level	Description
None	No authentication occurs.
Connect	Authenticates credentials only when the connection is made.
Call	Authenticates credentials at the beginning of every call.
Packet	Authenticates credentials and verifies that all call data is received. This is the default setting for COM+ server applications.
Packet Integrity	Authenticates credentials and verifies that no call data has been modified in transit.
Packet Privacy	Authenticates credentials and encrypts the packet, including the data and the sender's identity and signature.

COM+ provides a well constructed set of security features which address many of these issues. COM+ makes security more of an administrative issue than a programming one, and therefore helps you to spend more time developing application logic and less time writing security code. Configuring COM+ application security in the Component Services administration tool is a one-time task, and your application can thereafter remain free of security-specific code. At the same time, COM+ does provide APIs for accessing security information for those cases where you do need to go beyond the provided functionality. This article will provide an overview of security architecture for COM+ server applications and will show you how to use security in your COM+ applications.

Role-Based Security

COM+'s security architecture is often referred to as *role-based*. Rather than managing accounts for individual users, COM+ applications rely on categories or groups of users, referred to as *roles*. Roles work hand-in-hand with the operating system-based security, as the members of roles are the user accounts on the Windows 2000 server or domain. Roles can be created on an application-by-application basis using the Component Services administration tool, and the process is rather straightforward. Right click on the `Roles` node of the COM+ application in the tree on the left in the Component Services administration tool. Once a role has been added, another right click can add users to the role. Figure 1 illustrates the process of adding users to a role.

You can see from Figure 1 that, in this example, the COM+ application has three roles, called `Junior`, `Normal`, and `Hero`. These are simply names I made up to indicate three different groups of users I plan to



➤ *Figure 2*

provide differing functionality for in my COM+ application. Noteworthy is the fact that the actual authentication is handled automatically by the OS, and COM+ builds on top of those services.

Role-Based Security Configuration

Arguably the slickest aspect of COM+'s role-based security system is the fact that security can be established at the application, component, interface or even method level! This means that you can control which roles have access to which methods without writing a line of code.

The first step to configuring COM+ application security is to enable security at the application level. This is done by editing the properties of the application in the Component Services administration tool and then switching to the `Security` tab, shown in Figure 2.

Application security is enabled when the `Enforce Access Checks for this Application` checkbox is checked. This dialog also enables selection of the security level, which can be set to perform security checking at the process level only, or at the process and component level.

Enabling security only at the process level has the effect of locking the front door to the COM+ application, where all members of roles assigned to the application have the key to that door. When this option is selected, no security checking will be performed on the component, interface, or method level, and security context information will not be maintained for objects running in the application. This type of security is useful when you do not need granular security control, but simply wish to limit overall access to the COM+ application to a specific group of users. This type of security also has the advantage in increased performance, since security checks do not need to be made by COM+ during execution of the application.

Enabling security at the process and component level ensures role-based security checks will be made at the component, interface and method levels, and security context information will be available to objects in the application. While this provides maximum control and flexibility, note that the performance of your COM+ application will suffer slightly, due to the increased level of management that COM+ will need to perform during execution.

The dialog shown in Figure 2 also provides for configuration of the authentication level of the COM+ application. The authentication level determines the degree to which authentication is performed on client calls into the application. Each successive authentication level option provides for a greater level of security: the options are shown in Table 1.

Note that authentication requires the participation of the client as well as the server. COM+ will examine the client and the server preference for authentication and will use the maximum of the two. The client authentication preference can be set using any one of the following techniques.

- The machine-wide setting specified in the Component Services administration tool (or DCOMCNFG on non-Windows 2000 machines).
- The application level setting specified in the Component Services administration tool (or DCOMCNFG on non-Windows 2000 machines).
- The process level setting specified programmatically using the `CoInitializeSecurity` COM API call.
- An on-the-fly setting that can be specified programmatically using the `CoSetProxyBlanket` API.

Finally, the dialog shown in Figure 2 allows configuration of the

➤ *Table 2*

Impersonation Level	Description
Anonymous	The client is anonymous to the server.
Identify	The server can obtain the client's identity, and can impersonate the client only to perform Access Control checking.
Impersonate	The server can impersonate the client while acting on its behalf, though with restrictions. The server can access resources on the same computer as the client. If the server is on the same computer as the client, it can access network resources as the client. If the server is on a computer different from the client, it can only access resources that are on the same computer as the server. This is the default setting for COM+ server applications.
Delegate	The server can impersonate the client while acting on its behalf, whether or not on the same computer as the client. During impersonation, the client's credentials can be passed to any number of machines. This is the broadest permission that can be granted.

application impersonation level. The impersonation level setting dictates to what degree the server application may impersonate its client in order to access other resources on behalf of clients. Table 2 explains the options for impersonation level.

Like authentication, impersonation can only be accomplished with the consent of the client. The client's consent and preferences can be established exactly as authentication, using Component Services administration tool, DCOMCNFG, or the `CoInitializeSecurity` and `CoSetProxyBlanket` API calls.

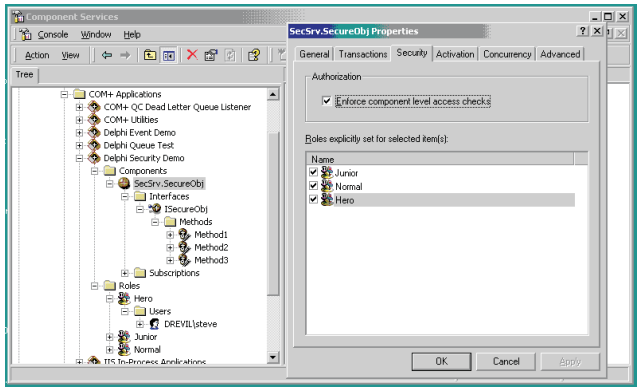
Once application security has been configured, security can then be configured for components, interfaces and methods of the application. This is done in a similar manner by editing the properties of the item in the tree and choosing the security tab. This will invoke a dialog with a page similar to that shown in Figure 3.

The dialog shown in Figure 3 is fairly straightforward; it enables you to specify whether security checks should be enabled for the item and which roles are to be allowed access to the item.

Multi-Tier Performance

When designing multi-tier applications that employ COM+ security, there are a number of performance considerations you should weigh. First and foremost, always bear in mind that one of the primary goals of a multi-tier system is to improve overall system scalability. One mistake that often compromises scalability and performance is to over-secure an application by implementing security at multiple tiers. A better solution would be to leverage COM+ services by implementing security only or mostly at the middle tier. For example, rather than impersonating the client in order to gain access to a database, it is more efficient to access the database using a common connection that can be pooled among multiple clients.

➤ *Figure3*



Programmatic Security

Up to now, I've focused primarily on declarative (or administration-driven) security, however I did mention that it is also possible to program security into COM+ applications. The most common thing you might want to do is determine whether the caller of a particular method belongs to a specific role. This enables you to control not just method access, but method behavior too, based on the role of the client. COM+ provides not one but two means for making this determination. There is a method of `IObjectContext` called `IsCallerInRole`, which is defined as:

```
function IsCallerInRole(
  const bstrRole: WideString):
  Bool; safecall;
```

This function is used by passing the name of the role in the `bstrRole` parameter; it returns a `Boolean` value indicating whether the current caller belongs to the specified role. A reference to the current object context can be found by calling the `GetObjectContext` API, which is defined as:

```
function GetObjectContext:
  IObjectContext;
```

The code in Listing 1 checks to see if the caller is in the `Hero` role prior to performing a task.

➤ *Listing 1*

```
var
  Ctx: IObjectContext;
begin
  Ctx := GetObjectContext;
  if (Ctx <> nil) and (Ctx.IsCallerInRole('Hero')) then begin
    // do something interesting
  end;
end;
```

Similarly, an `IsCallerInRole` method is also found on the `ISecurityCallContext` interface, a reference to which can be obtained using the `CoGetCallContext` API. This version of the method is actually preferred, simply because `ISecurityCallContext` makes handy a lot of other security information, such as the caller plus its authentication and impersonation level.

Summary

My hope is that by now you have a feel for how COM+ security works and how you can employ it in your own COM+ applications. COM+ makes integrating very powerful security into your applications a matter of a few mouse clicks in the administration tool (or a few lines of code using the admin API), but it doesn't sacrifice power for all its ease of use. The point is to get application developers out of the security programming business and into thinking about the core goals of their applications.

Steve Teixeira is CTO of DeVries Data Systems, a Silicon Valley interactive architect, and co-author of *Delphi 5 Developer's Guide*. You can reach him at steve@dvdata.com