Global Information Assurance Certification Paper

A Guide to Building Secure and Scalable Web Applications Using Microsoft COM+

Ben Bowne

Practical Assignment for SANS GSEC Certification

Assignment version 1.4b, Option 1

Abstract

This goal of this paper is to provide information that will help the application security analysts or developer to design, build and troubleshoot applications using the Microsoft COM+ application security framework. The main points of this paper are 1) a foundation of knowledge on the COM+ framework and its security model, 2) applying that model to the business requirements of an example application in multiple configurations, and 3) to point out some of the shortcomings of the COM+ security model as they relate to scalability and manageability. This paper assumes that the reader is somewhat familiar with basic information security concepts, web application development, and C++ programming.

The COM+ Framework and Application Security

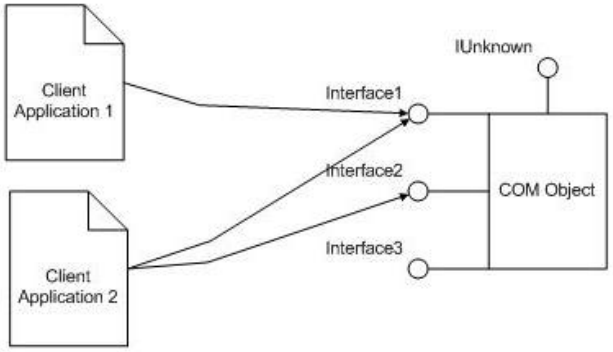
The current environment for the development of web-based business applications lies in several maturing and competing technologies. One of the most pervasive models being used in the web application industry today is the Microsoft COM+ application model. Microsoft's implementation of COM+ has the power to build highly secure and highly scalable applications. This paper will attempt to address both of these topics. At its root, COM+ is a model for exposing code functionality in a scalable object-oriented fashion. As its name suggests, COM+ is an extension of the earlier Component Object Model (COM) framework developed by Microsoft. One of the extensions of COM+ that is discussed here is the security model available. In order to understand the details of COM+ and its security model it is helpful to review the Component Object Model and its fundamentals.

Built on top of the Component Object Model, COM+ uses the same basic reference architecture to do work. There are two base elements in the component-based architecture: clients and servers. In each of the implementations of a COM based framework discussed here the concept of clients and servers persists. Clients are essentially requesting that server component do some piece of work for them. Server components tend to be finite bits of code made available to clients. Clients can take on a multitude of forms, but are generally considered to be users of programs requesting that work be done by server components.

COM applications may be built and housed on a per machine basis, or their functionality may be made available across a network. Although security becomes a concern when accessing code remotely, the benefits are great in a scalability and reusability sense. At the heart of the entire framework is the idea of an object model. In the COM object model, pieces of code that perform finite functions (server components) are treated as objects with identifiable interfaces.[1] COM interfaces are the way in which code exposes its functionality to people or programs that need this functionality. Each interface in COM is exposed by using a unique interface identifier or what is called an IID[2]. These unique identifiers are registered on the machine that the client code is running on, either locally or across the network. Using uniquely referenced and registered interfaces, the COM framework can be used as a scalability tool for application development and deployment, as many client resources can use the same COM interfaces within an organization. The following picture illustrates this concept of a client accessing a COM interface via a unique interface. This type of communication is common between

[1] See "Introduction to COM."
[2] See "Interfaces."

applications running over web servers, but can be accomplished outside of the context of network-based communications as well:



In the illustration above, two different applications are using the same COM object through different interfaces. The COM object may be providing a variety of services to a multiple number of applications. Data manipulation is one of the more common tasks of COM objects, but is by no means the only function provided. One point to note from the picture is the inclusion of the Iunknown interface. This is a special interface that is used by COM to navigate to the other interfaces on the object. This is one of the factors of Microsoft's COM object mode that has allowed it to be extended to meet a diverse set of business situations.

The client applications illustrated here may be any code running on the Windows machine, or a machine attached to the network. The connection to the COM object illustrated here may also be either through the underlying pluming of COM, known as Remote Procedure Calls, (RPC) or though some other proxy, such as a web server via the http protocol. These applications can be documents, programs or web elements. In the scenario pictured above, the COM object is referred to as the server object and the applications it is serving are considered clients. When looking at the physical implementation of the COM mode abstraction, many times, the code that is used to provide the functionality of COM objects is found in dynamic link library files, or files with the .dll extension. This is not a hard and fast rule though, as standard executables, or files with an .exe extension, can also be exposed by the COM framework.

COM Process Activation

Another technical point to take away from the architecture of COM is the way in which objects are created for the client to use. We have the concept of the object with its interface, and we have the notion of a file that contains the code to build the object, and we also have the idea of the client making a request for an object. At runtime there is an important process that is necessary for all of this to work. When applications reference a particular interface as they run, Windows uses what is known as the Service Control Manager, or SCM as the object activation service for COM. This service is usually invoked for base COM applications by calling the CoCreateInstance or CoGetClassObject methods. These calls are C++ methods that are a part of the Win32 application programming interface. When these method are referenced, the service control manager locates the proper file and creates and instance of the object locally for

the client application to use. The SCM then returns a pointer reference for the COM object to the client application.[3]

COM Process Identities

One of the points of the COM model that is important to security is the idea of creating objects in and out of what is known as a process. The concept of a process is very important to the way in which security is handled in the Windows environment. A process is simply the collection of what are known as threads. By its nature, the Windows operating system is a collection of different running processes. An example of a process and a thread would be an application, take Microsoft Outlook for example, that is running and is displaying multiple windows to the user via the console. The application that is running and is interacting with the user is the parent process and each window would be a thread of that process. This can be verified for the Microsoft Outlook application by using the Task Manager application and selecting the process tab. A list of currently running processes will be displayed to the user. The outlook.exe process will show up as a running process. In this case the outlook.exe process is considered to be running as a server application. The client in this case is the person editing documents.

Each process is associated with a particular identity or security principal. A security principal is simply a known entity such as a person, or a machine account. Processes must run with the security context of a principal. Usually this principal is either the user that started the process, the local system, or some other predefined identity. When a COM object is created by the SCM it can either be created in- process or out-of-process. When creating and in-process object, the identity of the parent process is inherited, and is subsequently used to make security decisions. When creating out-of-process objects, an entirely new process is created and a new security principal must be assigned to the process.[4] In early versions of the COM service control manager, out-of-process objects typically inherited the credentials of the local system, a highly privileged account, when created in a separate process. This is important to know for future discussions of the security model as well as a foundation for the evolution of COM.

The Arrival of MTS

With the advent of COM, Windows developers were beginning to move toward a model of reusable, scalable services. One of the early limitations of this system was the management of COM objects accessible over the network as well as the management of the security context that objects inherited via processes. As a part of the Windows NT 4.0 operating system, Microsoft introduced what is known as Microsoft Transaction Server, or MTS. One of the advantages of using MTS is that it provided a way for managing collections of COM objects that were to be accessed either locally, or via the network. A second major advantage was the control over security settings provided by the MTS framework. Before we can discuss the variations on security provided by MTS, a basic overview of its capabilities is necessary.

In MTS, COM components are grouped into what are known as packages. MTS uses two types of packages, server packages or library packages. Server packages are typically used as the main container for distributing work and typically call library

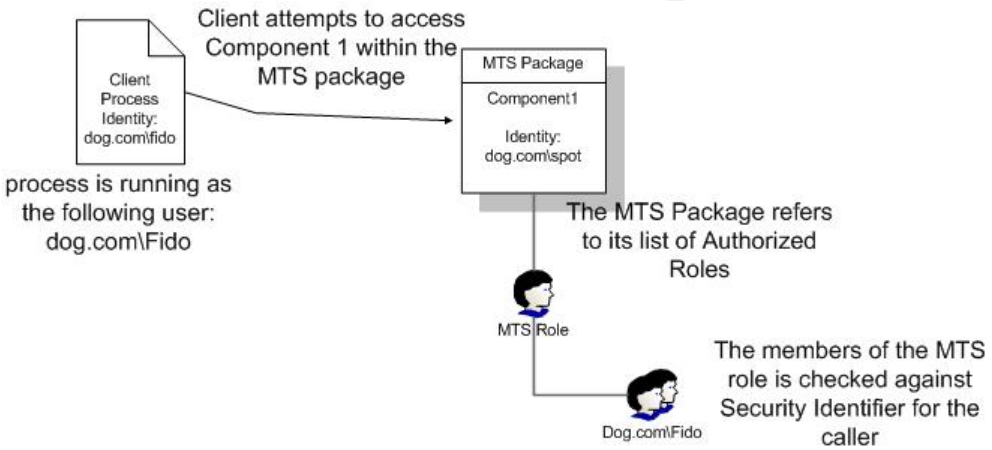
[3] See Ewald, Timothy. "Use Application Center or COM and MTS for Load Balancing Your Component Servers."

[4] See Brown, Keith. "Exploring Handle Security in Windows."

packages assist in the functionality being delivered by a collection of COM objects. Each package can contain one or more components, each component, one or more methods.

One departure from the way in which MTS security works in comparison to base COM is the idea that all COM objects managed within MTS will run in their own process, rather than in the process of the calling application.[5] The identity of that process could then be managed through the MTS interface. In addition to the above-mentioned changes, another big security related change to base COM that was brought about by MTS was the introduction of the concept of security roles. Through MTS, security roles could be assigned to individual components. These roles governed which clients had the ability to instantiate objects. Through this framework roles could be created as abstractions of the business users defined for an application. Roles could then be mapped to the components or groups of components that used by specific groups of users.

In addition, MTS provided a way of associating roles for an application back to groups defined by the Windows domain security model. Once this connection was made, each role could be *authorized* to create and use a specific component, or set of components. Authorization in MTS is determined by evaluating whether or not the identity calling the MTS component is a member of a predefined role. This authorization process is illustrated below:



Since the security context of the calling process is a member of a known role for this MTS package, the user is authorized and the Service Control Manager creates an instance of the Component1 object. This object is created in a new process using the identity of the MTS package.

Authentication in MTS and COM+

A brief side note on how security identifiers are evaluated as is depicted above will be helpful for subsequent sections. When the MTS or COM+ framework is evaluating whether or not a caller is a member of a certain group, the security identifiers, or SIDs of both the caller and the group are evaluated. SIDs are unique strings used to identify a security principal. Both users and groups have SIDs. SIDs for an individual can be generated either by the local machine authority, or in the case of a domain, by the Windows domain controller. Typically, the SIDs for an individual, and the groups they

[5] See Cambra, Troy. "Developing a Visual Basic Component for IIS/MTS."

are a member of, are assembled into a collection known as an access token.[6] When a caller is authenticated, or there identity is verified on a system, an access token is built for that user. When MTS roles are defined for a package, the MTS framework does the checking necessary to verify if SIDs in a users access token match the SIDs of the groups defined for a particular MTS role. This is known as a declarative security check. Programmatic security checking can also be done within the context of MTS and COM+. Programmatic security checking is described later in this paper.

Overview of COM+ Security

With the introduction of the Windows 2000 platform, Microsoft revamped and added functionality to the MTS interface and called this new management tool COM+. Although the differences between MTS and COM+ are numerous, for this discussion, the basic behavior of objects is the same. For the purposes of brevity, the security configuration abilities and security behavior of COM+ will be the main areas of focus. The remainder of this paper will focus on COM+ security, conceptually applying it to an example application, and some suggestions for dealing with some of the security problems associated with scaling COM+ applications.

Configuring COM+ security

Almost all of the security settings discussed here can be configured using the COM+ Microsoft Management Console Snap-In. This console is the main graphical window into COM+. The three main security configuration options in COM+ are the following:

- Authentication options
- Authorization options
- Identity options

Authentication Options in COM+

Authentication options deal with the way in which COM+ verifies the identity of the calling application or user. There are several options for the way in which authentication can be handled. The underlying technologies used for authenticated can be its own topic of discussion. For the purposes of this paper, we will assume that all clients and servers are using the Kerberos technology for authentication. This assumes that clients and servers are operating in a Windows domain environment. Essentially, all of the COM+ options for Authentication affect the same piece of communication, the data being sent between the COM+ application and the requesting client across the network. They are listed below in the order of the amount of security each level of authentication gives the application from least to greatest. In addition, there are some comments as to when each level should be used. The options for Authentication on COM+ packages are as follows[7]:

None – No authentication takes place when this setting is enabled. If the user is not authenticated, no access checking can happen. Using this setting will not allow you to enable authorization. It is also not a good setting from a defense-in-depth standpoint.

[6] See "Security Identifiers."
[7] See Eddon, Guy. "The COM+ Security Model Gets You Out of the Security Programming Business."

Connect – This level of authentication only authenticates clients the first time they connect to the server application. All subsequent packets sent back and forth from the client to the server are not authenticated. This option leaves the door open for man-in-the-middle attacks. This option should only be used when only trusted clients are permitted on a network such as in a local area network setting.
Call – This option authenticates the identity of the client every time the client makes a call to the server. Since a call to the server can contain more that one packet, this option does not authenticate all traffic back and forth from the client to the server. Again, this setting should only be used in a local area network environment.
Packet – When this option is set each packet received by the server is authenticated to ensure that the client is an authorized user. The contents of each packet are not examined and thus packet integrity is not guaranteed.
Packet Integrity – When this setting is used the server authenticates that each packet was sent by a valid client as well as verifies that the data being sent in each packet has not been changed. This is achieved by verifying a packet checksum value created by the client. This setting provides verification of the client as well as message integrity. While the message itself cannot be changed, packets are sent across the network in clear text.
Packet Privacy – This setting provides the greatest amount of protection as all packets are authenticated, checksums are calculated and sent, and the contents of all packets are encrypted using a session key established by the client and server. This setting has the largest performance impact of all authentication methods in COM+.

Authorization Options in COM+

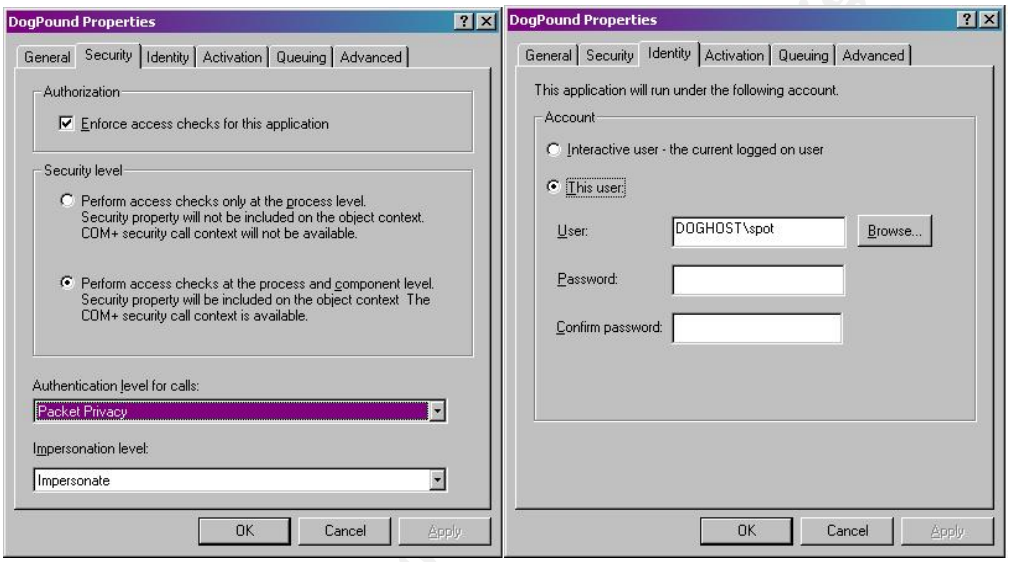
Authorization options within COM+ are relatively few when viewed from the COM+ management console. Administrators of COM+ applications have the choice to enforce access check for an application. If access checking is not enabled, role based security for a COM+ application cannot be used. Once access checking has been enabled, COM+ developers can set up security roles for the application. Authorization roles are typically and abstraction of business rules for the functionality of the application itself. The application of roles will be illustrated in the example application CarDealer shortly.

Identity Options in COM+

The final category of security options for COM+ deals with the *identity* of the COM+ application. Identity options can only be set on COM+ server packages. Under the Identity tab of the COM+ management console within Windows 2000 there are two options for which identity a COM+ package can use. These options are either to run the package as the interactive user or as a specific identity. Running an application as an interactive user means that the application is running as the currently logged in user. This can be problematic if the application resides on a machine that has no interactive users. This setting should only be used when debugging applications that are running locally. As a rule of thumb, an identity should always be used to run COM+ server packages. Another tip that will lessen an applications exposure to such coding flaws as buffer overflows is to run COM+ applications using an identity that is not an administrator. Running code as an administrator can be dangerous for several reasons and the identity set in COM+ is typically where this mistake is made[8].

[8] See Brown, Keith. "Defend Your Code With Top Ten Security Tips Every Developer Must Know."

Also related to the Identity options for COM+ applications is the notion of impersonation. Impersonation deals with the process of making a call from one COM+ package to another secured resource. COM+ applications running in their own process, with their own identity can impersonate the calling user. The plumbing behind this technique includes adding the access token of the user to the process thread of the COM+ package. The end result of impersonation is that a COM+ package can use the security context of the caller to access resources. All of the settings described here are available in the two property pages shown below for the COM+ server application DogPound. Notice that authorization checking is enabled, the authentication level is set to packet privacy, impersonation is enabled, and the package has its own identity.

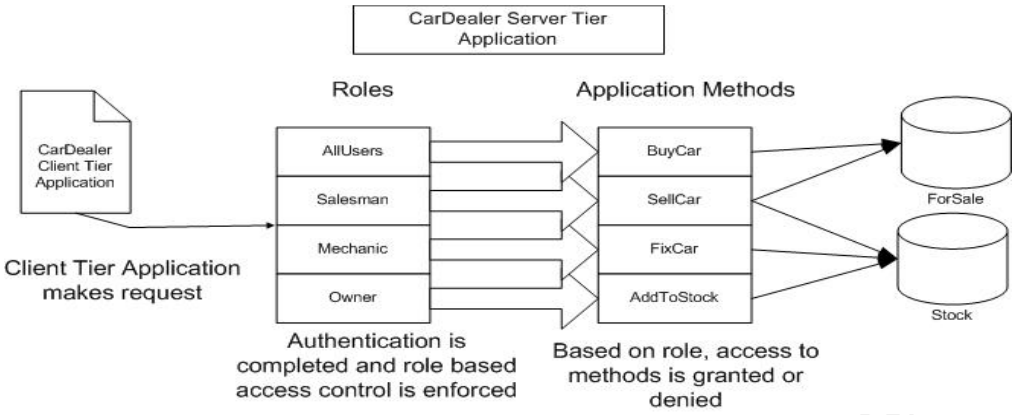


The CarDealer Example Application

For the purposes of illustrating how COM+ application security can be put to use it is helpful to have an example application. The example is a 2-tiered application called CarDealer. This simple application has four exposed components for doing work:

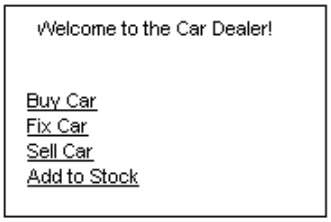
- *BuyCar* – This method is used to take an element named Car out of the CarDealer for sale database. According to the business rules for use, anyone that is allowed into the CarDealer system is allowed to use this method.
- *FixCar* – This method is used to maintain the Car elements in the CarDealer stock database. According to the business rules for use, this method should only be accessible by the Mechanic role.
- *SellCar* – This method is used to move the Car elements from the stock to the for sale database. Users that are members of the Salesman role may use this method.
- *AddToStock* – This method is used to add Car elements to the stock database. This method should only be used by the members of the Owner role.

The application is illustrated below:



In the application above, every COM+ security role defined is used as a declarative security role. What this means is that the roles act like a front door to the methods inside the CarDealer application. If a requesting client is not a member of the specified role he or she is not able to create an instance of the COM+ object. This declarative security check is done for the developer by the COM+ security subsystem. This is useful in that it takes the burden of programming security out of the hands of the developer. All the developer has to do is maintain the roles associated with each object in their application. When this application is configured the security administrator assigns one or more domain security groups to the roles defined here to tie the abstracted security roles to concrete users.

This example assumes that there are essentially 4 different options available to the client tier application regardless of which type of user accesses the CarDealer application. What this means is that a user who attempts to create an instance of a COM+ object that is not in the correct declarative security role that user will receive a permission denied message. The permission error that is generated by COM+ shows up in the application log on Windows 2000 servers as a permission denied: error 70 message. The User interface displayed by the client might look something like the one below.

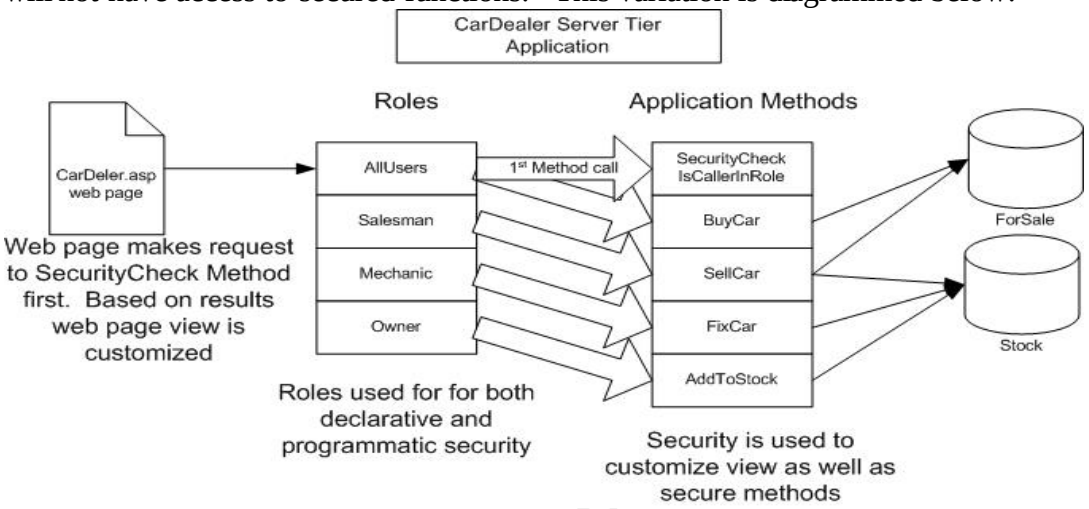


Example User Interface has all method options displayed to the user.

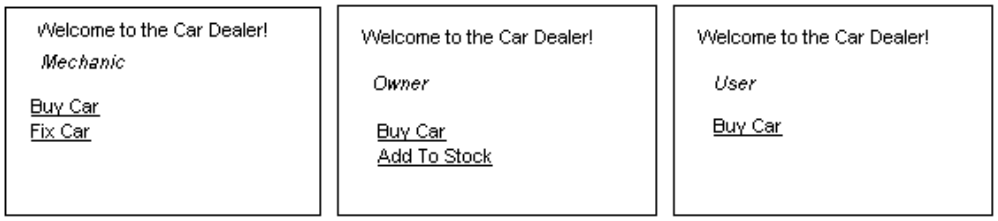
Another way to design the application is to use a web based client application such as an active server page, or .asp page that displays options to the user based on his or her role. In this scenario, the developer uses the security context of the user to present a different user interface to the client. This is a form of programmatic security. Most of the applications of programmatic security using COM+ use one of two methods, either IsCallerInRole or IsUserInRole. The subsequent section will discuss dealing with applications that use both declarative and programmatic security.

The most robust security solution for a two-tiered application that is using COM+ role based declarative and programmatic security is pictured below. In order to facilitate the use of programmatic security checking a new method has been added to the application that references the IsCallerInRole method. The SecurityCheck method is

used to implement logic that changes the User Interface of the client based on the results of the IsCallerInRole check. If implemented correctly, the end user will not be presented with choices that he or she does not have access to. Also, from a security coverage standpoint, this application does not solely rely on limiting the User Interface for authorization; declarative COM+ security roles are still validated when a client attempts to create an instance of a secured object. The end result is that friendly users should not receive permission denied errors, while malicious users that are outside of the proper role will not have access to secured functions. This variation is diagrammed below:



Example User Interfaces:



Notice that the User Interface has been customized by the logic in the application method SecurityCheck based on the results of the IsCallerInRole method. Notice also that all the User Interfaces have the Buy Car option enabled. This is because the BuyCar method has as a role the all users group.

Drawbacks to Using the Declarative and Programmatic COM+ Security Model

There are several drawbacks that should be noted when attempting to implement a declarative and programmatic role based COM+ architecture such as one described in the CarDealer application. These difficulties lie primarily in three distinct areas: complexity, administration, and performance. Each are will be treated separately.

Complexity problems can arise very quickly when trying to implement a role based security solution for COM+ applications that utilizes both a declarative and programmatic security model. First and foremost, the developer has to have the expertise to add programmatic security checks to each application. In smaller organizations this may not always be possible. The second reason complexity becomes a problem is that the system administrator must have each COM+ application configured correctly with respect to security. In order for role based security to work in COM+ it must first be enable and set up correctly from a role to group mapping standpoint. This may sound trivial but administration is sometimes overlooked, especially for large applications

where there may be many secured methods within a COM+ application. An interesting side note that needs to be mentioned here: By default in Windows 2000, COM+ authorization checking is not enabled on packages. When programmatic security calls are made in packages without authorization checking enabled using the IsCallerInRole and IsUserInRole methods, they value returned is always true, regardless of whether or not the user is in the role specified.[9] A fix for this is to make sure that developers always call a method called IsSecurityEnabled to determine if authorization checking is enabled before making their programmatic security calls. Again, this adds yet another layer of complexity.

Administration of COM+ role based security infrastructures can be one of the greatest challenges for large organizations with custom written applications. In the example application only four distinct roles were used and the membership of these roles was rather straightforward. When applying this model to a large organization with many applications containing many roles and their associated groups, administration frequently becomes more unwieldy that first anticipated. Another hurdle that must be anticipated is what happens when a user changes roles and must be removed from groups and added to new ones. Since only administrators can manage these changes, the time and effort necessary to complete these activities on an ongoing basis may be prohibitive.

The third area that of concern when deploying and scaling applications using declarative and programmatic COM+ roles is the area of performance. Recall from previous discussions that the plumbing that COM+ uses to evaluate whether or not a security principal is a member of a particular role involves analyzing SIDs in the client's access token and the SIDs of the groups that are members of the specific role. In the CarDealer application, this validation is performed multiple times for each user. The SIDs of the access token are compared to group SIDs once for the initial declarative security check to the SecurityCheck method, a second time during the actual programmatic security check, and again when the client SIDs are validated for the declarative security when a client attempts to do some work with one of the other methods in the application. This can be very costly to the performance of the application, especially when the number of roles increases and the programmatic security logic is detailed.

A second and equally important performance impact of using programmatic security involves using this technique on systems with more than 2-tiers. Also recall that all applications running under COM+ run in their own process using their own identity. When COM+ packages are configured to allow for Impersonation, the access token of the calling user or application is added to the thread of the process that the COM+ application is running as. This thread token is only persistent for the subsequent call and cannot be passed on to the next process in the call flow. This is good for security purposes, but bad for performance if programmatic security is used. When programmatic security is used more that two calls from the original user request, the IsUserInRole method must be used to determine what programmatic privileges should be assigned to the user. Since the process that calls this method does not have the access token of the original caller, one must be generated. This can be a performance hit to the application, as access tokens typically have to be generated by a domain controller. This method

[9] See "IsecurityCallContext::IsSecurityEnabled"

should only be used when programmatic security is necessary and performance is not the main goal of the application.

Summary

The COM+ application framework and its security model provide a basis for building secure and scalable applications on the Microsoft platform. This security model, built on previous versions of the COM model, provides a means for creating robust application security frameworks based on roles. The transition from business requirements to secured applications can be done through the use of a role based system. In addition, the roles based philosophy can be harnessed to provide a secure user and code environment through programmatic and declarative security checking. There are drawbacks to using the COM+ role based security architecture. The most commonly encountered drawbacks deal with complexity, administration and performance.

References

Brown, Keith. "Defend Your Code With Top Ten Security Tips Every Developer Must Know." Security Briefs. **MSDN Magazine**. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mts/mtxpg04_7i49.asp (9 Mar. 2003).

Brown, Keith. "Exploring Handle Security in Windows." Security Briefs. **MSDN Magazine**. URL: http://msdn.microsoft.com/library/default.asp?url=/downloads/list/security.asp (9 Mar. 2003).

Cambra, Troy. "Developing a Visual Basic Component for IIS/MTS." **MSDN**. Updated September 2001. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomser/html/vbmtsiis.asp (9 Mar. 2003).

Eddon, Guy. "The COM+ Security Model Gets You Out of the Security Programming Business." **Microsoft Systems Journal**. November 1999. URL: http://www.microsoft.com/msj/defaultframe.asp?page=/msj/1199/comsecurity/comsecurity.htm&nav=/msj/1199/newnav.htm (9 Mar. 2003).

Ewald, Timothy. "Use Application Center or COM and MTS for Load Balancing Your Component Servers." **Microsoft Systems Journal**. January 2000. URL: http://www.microsoft.com/msj/0100/loadbal/loadbal.asp (8 Mar. 2003).

"Interfaces." Visual C++: Adding Functionality. **MSDN**. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_atl_interfaces.asp (8 Mar. 2003).

"Introduction to COM." Visual C++: Adding Functionality. **MSDN**. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_atl_introduction_to_com.asp (8 Mar. 2003).

"IsecurityCallContext::IsSecurityEnabled" Platform SDK: COM+ (Component Services). MSDN. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cossdk/htm/isecuritycallcontext_3smc.asp (9 Mar. 2003).

"Security Identifiers." Platform SDK: Security. **MSDN**. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/security_identifiers.asp (8 Mar. 2003).