Keysight PNA Series DCOM Security

This paper considers a very narrow slice of DCOM security; specifically, the aspects that pertain to developing test software for the PNA. The targeted audience is advanced programmers.

PNA DCOM Programming - Configuring Security

There's a widely held misconception regarding DCOM - PNA programming and that is: the remote client must have a user account on the network analyzer in order access the PNA. Many users encounter PERMISSION DENIED errors when attempting a DCOM connection. And most solve this problem by ensuring that the remote computer and the PNA share identical user accounts with identical passwords. Doing so provides the PNA with a way to **authenticate** the caller's identify. Certainly this is one way to solve the problem, but it's not the only way.

COM uses the same security protocols as MSRPC (Microsoft Remote Procedure Calls). The basis for MSRPC security is rather simple: transactions succeed as long as the client and the server agree on the security protocol to be used. In other words, network authentication is not imposed on DCOM, although client and server can elect to use that level of security if they so desire.

An oversimplification of the basic client-server security negotiation is:

- The server application (in this case, the PNA) sets, at run time, the minimum security protocol that it requires. You can think of this as the LOW WATER MARK. A client cannot negotiate a level below this mark.
- The client (that's you), initiates a DCOM session and selects the desired level of security.
- The result of this negotiation always goes to the high bidder and thus, security is always imposed at the higher level.

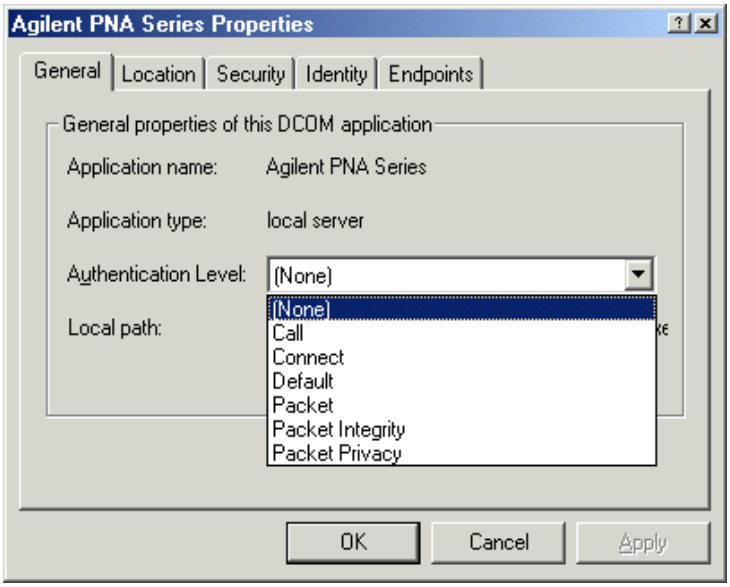
So let's look at each of the security principals in this situation and see how we can control the outcome.

The PNA's DCOM Configuration

Recall that the server sets the low water mark for security. Out of the box, the PNA's DCOM security is wide open. You can verify this by using the dcomcnfg utility to examine the security settings.

The screen shot on the right shows the first page of the dcomcnfg utility with a selection of NONE for authentication. As an administrator on the PNA, you can change this level to meet the needs of your organization.

Additionally, under the security tab you can edit the list of users allowed to access the PNA, ranging from *Everyone* to very specific users.



Client Security Configuration

This section will answer the question you've probably been asking all along. If the PNA security setting is low why do my anonymous attempts (i.e.: no account on the PNA) result in E_ACCESS_DENIED?.

The most likely reason is that you neglected to configure security for the client side application. And as a result, the defaults have been employed. The default authentication level is RPC_C_AUTHN_LEVEL_CONNECT (see below), which requires that the client be identified on the first contact. Under these conditions, if you don't share a domain with the PNA or have a local account on that machine, authentication will fail and so will your first PNA automation call.

(You might be wondering, if it is your own security level request that is hindering access to the server, why the error message could not be more illuminating. Suffice to say that helpful security hints would also help the bad guys.)

To configure security for your application, you must call CoInitializeSecurity, a method on the COM API exported by ole32.lib. For C++ programmers, this is a trivial exercise. There's a bit more work involved for VB and VEE clients, but this paper will provide examples.

CoInitialize Security

Here's the call to CoInitializeSecurity in it's least interesting form, but one that basically asks for anonymous COM control.

```
int main()
{
  CoInitialize(NULL);
  CoInitializeSecurity(NULL,      //security descriptor
                                  0,                      // authn svc entries
                                  NULL,                   // authn svcs
                                  NULL,                   // reserved
                                  RPC_C_AUTHN_LEVEL_NONE,
                                  RPC_C_IMP_LEVEL_ANONYMOUS,
                                  0,                      // authn info
                                  0,                      // capabilities
                                  0);                     // reserved

            // do work here

  CoUninitialize();
  return 0;
}
```

There are several aspects to configuring security settings as you can see from the above call and depending on your application you may need to pay close attention to all of them. For more information consult the documentation for the specific method call.

The fifth argument in this call is the most critical to this discussion. This value indicates the client's expectation for authentication. Authentication is performed when the server verifies that the caller is who he says he is. Authentication requires that the server be able to locate the client's identity in the local accounts data base (Local Service Authority) or via the domain controller. The choices for authentication are as follows:

RPC_C_AUTHN_LEVEL_DEFAULT
RPC_C_AUTHN_LEVEL_NONE
RPC_C_AUTHN_LEVEL_CONNECT

RPC_C_AUTHN_LEVEL_CALL
RPC_C_AUTHN_LEVEL_PKT
RPC_C_AUTHN_LEVEL_PKT_INTEGRITY
RPC_C_AUTHN_LEVEL_PKT_PRIVACY

RPC_C_AUTHN_LEVEL_NONE means no authentication is performed. In other words, if the client and server can agree on no authentication, then anonymous calls to the server will be allowed. The rest of the values employ authentication in ever increasing intervals. RPC_C_AUTHN_LEVEL_DEFAULT is synonymous with RPC_C_AUTHN_LEVEL_CONNECT, demanding authentication when the client first connects to the server - a good choice if you want to identify your callers with minimal network overhead. All other levels require authentication on a per-call or per-packet basis which can be quite costly in terms of performance.

Many of you are likely reading this to discover how to defeat DCOM security, not employ it. But for those that require authentication, you should also pay attention to the impersonation level. If the authentication level is NONE (anonymous calls) then impersonation is meaningless. But when authentication is required and the caller identifies himself, the impersonation setting limits the extent to which the server can perform actions using the caller's identity.

VB / VEE Clients

VB and VEE clients both have two environments to contend with: the design/debug IDE environment provided by the Visual Basic IDE, and the run time environment for compiled code.

Setting Security for Compiled VB Apps (.exe)

Your compiled exe can call CoInitializeSecurity very much the same way that C++ clients do. You can do this by prototyping the COM API call with the VB Declare statement, as shown below.

Note that the code fragment in the sample below is contained in a module, not a form. This is because you must execute this code prior to loading any Active-X controls, and the VB form is such a control. So add a module to your program with the call to CoInitializeSecurity. Then, under project properties, set the startup object to Sub Main. Compile the program and run the .exe directly. Use this link for the complete code sample.

If you try to run this code from the IDE, the CoInitializeSecurity call will fail. This situation is addressed later.

```
' Function Declaration
Private Declare Function CoInitializeSecurity Lib "OLE32.DLL" ( _
  pSD As Any, ByVal cAuthSvc As Long, asAuthSvc As Long, _
  pReserved1 As Any, ByVal dwAuthnlevel As Long, _
  ByVal dwImpLevel As Long, ByVal pAuthInfo As Long, _
  ByVal dwCapabilities As Long, pvReserved2 As Any) As Long

Sub Main()
  Dim hr As Long
  Dim AuthnSvc As Long
  AuthnSvc = RPC_C_AUTHN_DEFAULT

  hr = CoInitializeSecurity(ByVal 0, -1, AuthnSvc, ByVal 0, _
                            RPC_C_AUTHN_LEVEL_NONE, _
                            RPC_C_IMP_LEVEL_IDENTIFY, 0, 0, ByVal 0)
  If (S_OK <> hr) Then
    MsgBox "CoInitializeSecurity failed: 0x" & (Hex(hr)), vbCritical, "App Init Failure"
    Exit Sub
  End If
  form1.Show  'Show the starting form; assumes the form's name is "form1"
End Sub
```

Setting Security for the VB IDE

When you are developing and/or debugging, most often your program is executing in the context of a development environment. These environments, such as VB and VEE, host Active-X controls, so by the time *your* program executes,

CoInitializeSecurity will already have been called. And since CoInitializeSecurity can be called only once per process, your attempts to call it again, using the example above, will fail. So while working in the IDE we need to employ another solution. We want to be able to specify the security settings for VEE and VB so that when those applications start up, the desired security settings are employed. We can do that via the registry.

In cases where an application does not call CoInitializeSecurity directly (like VB and VEE), COM will first check the registry for the existence of an AppID key. If found and the key contains named security values, those values will be used in place of defaults. So by adding AppID keys for VB or VEE, we can achieve the desired security configuration while we're running our code in the development environment. See *Adding an AppID Key* below.

Setting Security for Compiled VEE programs (.vxe)

Compiled VEE code is not machine code. It requires the VEE runtime environment as a host. So like the VB IDE (see above), we can generate a registry AppID key to control the DCOM security settings for the vee runtime. See *Adding an AppID Key* below.

Setting Security for the VEE IDE

The VEE development environment can be configured by running dcomcnfg. In the dcomcnfg dialog locate the VEE Callable Server. Set the authentication level to NONE on the General tab of the properties sheet.

Adding an AppID Key

Replace *YourClient* with the actual name of the .exe.

VB6: vb6.exe
VEE: veerun.exe

Replace *YourGuid* with a GUID. A GUID is a 128-bit number formatted for registry use as follows

{2B0E70DA-A075-42d4-894F-C1C0D0B4843A}
{20D9AD2D-6DA8-46f9-AC05-255FFCC969F1}
{ABA0E9B5-06AD-446f-91CF-05690B718691}

You can use the guidgen.exe or uuidgen.exe tools provided with Visual Studio to generate your own guids.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\AppID\yourclient.exe]
@="YourClient"
"AppID"="{YourGuid}"
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\AppID\{YourGuid}]
@="YourClient"
"AuthenticationLevel"=dword:00000001
```

Summary

Hopefully this paper has made the rules for DCOM communication more explicit if not oversimplified. The basic things to remember are these:

- The server sets the minimum allowable security and with respect to the PNA, security settings can be set by the administrator using DCOMCNFG.
- The client must explicitly specify security settings or be configured with the defaults.

- The default authentication level is "Connect".
- If authentication is required, the server must be able to identify the client. This requires a local account for the client or the same or trusted domain.

One more thing: Events

If you want to receive events from the PNA, then you must implement an event handler. In VB and VEE this a rather simple task. However, be aware that dispatching events reverses the roles of client - server. The same security rules apply. But during the event call, your event handling app is the server and the PNA is the client. You have to think in reverse. If both sides are set for anonymous operation (rpc_c_authn_level_none) receiving events is rather painless.

The classic problem: You've created an account on the PNA to overcome the symptoms of Connect level security (i.e.: you didn't call CoInitializeSecurity), but find that you can't receive events. The dreaded PERMISSION DENIED has returned. Keep in mind that by default, the PNA runs as the interactive user. With authentication set to Connect, your workstation will need to authenticate the PNA's logged in user in order to receive events. To accomplish this, you must either use the CoInitializeSecurity method described above or use the registry setting described above in "Adding an AppID Key".

If all this is a bit confusing, there are two VB example programs that deal with events. The download package includes a six-page document that goes into depth reqarding events and permissions. Download this 60kB zipped file which includes both the programs and the Word document. It also mentions C# and .Net issues.

Last Updated: Apr 10, 2009