

# TROUBLESHOOTING RPC ACROSS FIREWALLS

Colin Bowern

Applications that want to talk to other servers will often use the Remote Procedure Call (RPC) infrastructure to communicate instead of inventing their own protocol. From an IT perspective this is a pain because we like known protocols with defined ports. From a development perspective this is nice because we don't want to write more code than we have to. More modern applications are going to take advantage of technologies like Windows Communications Foundation (WCF) which enables administrators to configure how the applications will talk, including the transport protocol (e.g. TCP, HTTP, custom protocols), along with the transport-specific properties (e.g. ports). While we wait for the uptake on WCF we will likely be stuck with RPC-based applications for quite some time. With that being the case it is best to understand how RPC works and the implications brought forth by firewalls.

I've read very few good explanations for RPC that a typical systems administrator would understand (read: without the developer-speak), so here goes:

1. The client application opens up a dynamic outbound port and makes a call to the server's RPC Endpoint Mapper. There are several ways that you can talk to the endpoint mapper:
  - Local RPC - it just connects
  - Named Pipes - `\pipe\epmapper`
  - TCP - Port 135/tcp
  - UDP - Port 135/udp
  - HTTP - Port 593/tcp
2. The RPC Endpoint Mapper processes the request for access to the application. It will handle authentication and a number of provisioning processes. The server-side service will be launched using either its own custom container (e.g. WMI on Windows Vista and later uses `%SYSTEMROOT%\SYSTEM32\WBEM\UnSecApp.exe`, Windows Media Server uses `%SYSTEMROOT%\SYSTEM32\Windows Media\Server\WMServer.exe`), `dllhost.exe`, or `svchost.exe`.
3. With the service launched on the server side it will then tell the client that it needs to continue the conversation with the service on a specific port. This port will be located in the 1024+ range on most Windows machines, however in Windows Vista and Windows Server 2008 this is now 49152 - 65535 by default. If you are stuck on older versions of Windows, here's some more ammo to get that upgrade going. In the mean time I would recommend that you at least take a look at setting a specific RPC endpoint range.
4. The client will continue its conversation with the service it needed on the port returned by the endpoint mapper. You can see all of this happening using `NETSTAT -B`:

Proto	Local Address	Foreign Address	State
TCP	192.168.0.111:135	192.168.0.110:49228	ESTABLISHED
RpcSs			
[Svchost.exe]			
TCP	192.168.0.111:49153	192.168.0.110:49228	ESTABLISHED
[Dllhost.exe]			

The output shows you the client is communicating from port 49228 to 135/tcp, and subsequently established a connection to the service on port 49153 to a service which uses `dllhost.exe` as a container. Note that the `SvcHost.exe` exposes what service it is exposing through the port (more on that in a minute).

# TROUBLESHOOTING RPC ACROSS FIREWALLS

Colin Bower

Most administrators and security professionals get anxious when a service is dynamically allocating random ports in a large range. This really goes against the grain of well known ports that have specific purposes that they can lock down. As a result of the uproar several services like NTDS, Netlogon, and FRS (don't forget the DFSRDiag StaticRPC port noted in KB823017 under Distributed File Replication Service) have enabled you to specify static endpoints. This isn't something new, if the application uses Distributed COM (DCOM) then there is already an option to lock the application to a specific endpoint. Making these changes on an application that hasn't been tested with it though could bring unanticipated results, so consult your application vendor before going hog wild in the Component Services management console.

I would argue that even though ports can be locked down that unless your firewall does deep packet inspection (this is where your choice of ISA Server pays off, even though it can be painful to configure) all I need to know is how to launch a service using that port to get around your firewall (port 80 comes to mind as a favourite port to launch a service under). This deep packet inspection is exactly what Windows Firewall with Advanced Security offers in Windows Vista and Windows Server 2008. After troubleshooting a problem with remotely administering IIS on Server Core recently I noticed that the out-of-the-box firewall has added a nifty RPC packet inspection layer. Hopefully we will see IIS product group take up the use of Svchost or a custom container soon. In the mean time here is an example of a rule group that, when enabled, will allow you to remotely manage Task Scheduler:

```
Rule Name:      Remote Scheduled Tasks Management (RPC-EPMAP)
-----
Description:    Inbound rule for the RPCSS service to allow RPC/TCP
                traffic for the Task Scheduler service.
Enabled:        No
Direction:     In
Profiles:      Domain,Private,Public
Grouping:      Remote Scheduled Tasks Management
LocalIP:       Any
RemoteIP:      Any
Protocol:      TCP
LocalPort:     RPC-EPMAP
RemotePort:    Any
Edge traversal: No
Program:       C:\Windows\system32\svchost.exe
Service:      RPCSS
InterfaceTypes: Any
Security:     NotRequired
Action:       Allow
```

```
Rule Name:      Remote Scheduled Tasks Management (RPC)
-----
Description:    Inbound rule for the Task Scheduler service to be
                remotely managed via RPC/TCP.
Enabled:        No
Direction:     In
Profiles:      Domain,Private,Public
Grouping:      Remote Scheduled Tasks Management
LocalIP:       Any
RemoteIP:      Any
```

# TROUBLESHOOTING RPC ACROSS FIREWALLS

Colin Bowers

```
Protocol:      TCP
LocalPort:    RPC
RemotePort:   Any
Edge traversal: No
Program:      C:\Windows\system32\svchost.exe
Service:      schedule
InterfaceTypes: Any
Security:     NotRequired
Action:       Allow
```

In the above example you can see that the first call will come into SvcHost.exe which hosts the RPC EndPoint Mapper (noted as Service = RPCSS, LocalPort = RPC-EPMAP). The subsequent calls will be to a dynamic RPC endpoint, defined in the second protocol, hosted by SvcHost.exe called Schedule. Now here's the bad part - it appears to only work for services hosted by SvcHost.exe. If the service uses DllHost.exe then you are out of luck. If you have a custom container then you can just lock down to that already - no need to worry.

## UPDATE

Earlier in the post I referred to an issue managing IIS on Server Core using the Microsoft.Web.Administration assembly. This was what got me started down the path and why I wrote this post. My conclusion was to add a firewall exception for dllhost.exe to allow remote procedure calls to connect with it. Simple enough you might think, but this does open up a whole host of potential security issues. Mike Volodarsky from the IIS product group has taken the problem away to the configuration team to come up with an official solution. In his post he mentioned that the AHAdmin component was the one we were communicating with. That got me thinking - how would I have figured that out on my own? That leads us to part two of this saga - how to determine which component DllHost is representing.

After running NETSTAT -B I was able to see my RPC endpoint mapped to DllHost.exe. Opening the firewall to that particular container leaves a lot exposed to the world, so I really want to lock it down. Earlier in the post I mentioned that you can lock down DCOM objects, the trick is to know which one to lock down. Since COM has some restrictions like length of object names coupled with developer habits of choosing the most acronym loaded names it is not easy to figure out which component it is that you need to lock down. I set PowerShell on an infinite loop to make calls to my RPC endpoint to ensure it stayed alive during this investigation. NETSTAT -B gave me some clues to look for in the RPC endpoint port number. The next tool I pulled out was Process Explorer. Thankfully it works on Server Core (I think I would be lost sometimes without this tool). Sorting the process list by name allowed me to locate instances of DllHost.exe quickly. Opening up the properties I clicked on the TCP/IP tab to confirm that it was indeed the instance I wanted based on the ports on which the process was bound. Clicking back to the Image tab the magic missing link in the Command Line text box:

```
DllHost.exe /Processid:{9FA5C497-F46D-447F-8011-05D03D7D7DDC}
```

Using this piece of information I opened up the Registry Editor, went to HKEY\_CLASSES\_ROOT\AppID as indicated in the KB217351, and located the GUID from

# TROUBLESHOOTING RPC ACROSS FIREWALLS

Colin Bown

the command-line. This led me to the AHAdmin object which appears to be the object being remotely instantiated (now Mike had given this away in his post, but the story sounds much more dramatic if you put that to the side for a second). I inserted the Endpoints value into the GUID key as per the KB article, and restarted the server, and kicked off my PowerShell script calls once again. This time NETSTAT -B gave me a different result - it was now using my static RPC endpoint! With that now in place I updated my new firewall rules and made some security folks very happy:

```
NetSh AdvFirewall Firewall Add Rule Name="Remote Web Server Management (RPC)"
Dir=In Action=Allow Program="C:\WINDOWS\SYSTEM32\dlhhost.exe" Protocol=TCP
LocalPort=49494
```

```
NetSh AdvFirewall Firewall Add Rule Name="Remote Web Server Management (RPC-
EPMAP)" Dir=In Action=Allow Program="C:\WINDOWS\SYSTEM32\SvcHost.exe"
Service=RPCSS Protocol=TCP LocalPort=RPC-EPMAP
```

Now I'm left waiting to see if I messed anything up. Setting a static endpoint could have dire consequences depending on how the application is written. Since AHAdmin is a black box I'll wait for the product group to provide some more official guidance on securing the service. In the meantime the new knowledge around tracing RPC calls from client to server to object is a handy one that has helped me confirm the problem and identify a potential solution.