Global Information Assurance Certification Paper

Jeremy Hewlett
GCIH v2.1a, option 2

Support for the Cyber Defense Initiative:

Messenger Service abuse via Microsoft RPC

Abstract

The material in this paper covers some possible attack vectors making use of Microsoft's End Point Mapper service. The Mapper's primary function is the same as that of the UNIX Port Mapper, which is to associate dynamically allocated ports to a service. When clients need to communicate with an RPC-enabled service, they will check with the Mapper to determine the service's port. The Mapper service listens at port 135 on both TCP and UDP under Microsoft Windows.

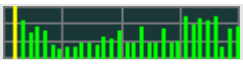
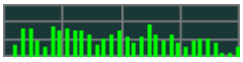
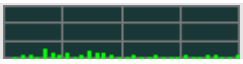
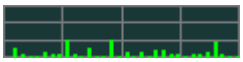
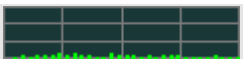
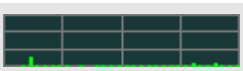
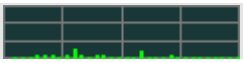
At the time of this paper, connections to port 135 were becoming popular; so various avenues of exploitation of this service have been explored. However, this paper focuses primarily on one type of Mapper abuse that is being used to send unsolicited messages to users logged on to the target machine. It is the author's belief that the majority of connection attempts to port 135 are because of this type of abuse, commonly called Messenger Spam.

The aim of this paper is to discuss what the exposure is, how and why this type of abuse works, methods of prevention, and suggestions for fixing it. While this paper will focus primarily on one specific product that many people are using, several other variants are also explored.

Part 1 – Targeted Port

On *January 11, 2003 18:49 pm GMT*, the current CID graph appeared as the following:

Top 10 Ports

Service Name	Port Number	30 day history	Explanation
netbios-ns	137		
http	80		HTTP Web server
ms-sql-s	1433		Microsoft SQL Server
netbios-ssn	139		Windows File Sharing Probe
Microsoft-ds	445		
???	4662		
ftp	21		FTP servers typically run on this port
ms-term-serv	3389		
domain	53		Domain name system. Attack against old versions of BIND
???	135		

Much of the above are the same types of ports one would generally expect to see in a listing of the top ten most attacked ports. Many of the ports above have remained popular due to their overwhelming number of vulnerabilities or exposures. So, for this reason I cut out those ports to leave me something that would be more challenging. This filtering left me with ports 135 and 4662. A cursory search on www.google.com says that port 135 is related to Microsoft RPC services, and the Microsoft URL http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/ittasks/tasks/adrepfir.asp confirms what I have been seeing. Port 4662 is related to eDonkey, which is another peer-to-peer file sharing application that touts itself as "*the next generation of file sharing.*"[1] Edonkey can be located at

[1] "eDonkey2000" 28 January 2003 URL: http://www.edonkey2000.com (28 January 2003)

http://www.edonkey2000.com/. Anyway, Microsoft RPC services are much more interesting than peer-to-peer, so off I go!

Targeted service

What other applications or services might be associated with this port? Well, let's take a look at what the Internet Storm Center has to say about it:

Services registered for this port (from Neohapsis)

Protocol	Service	Name
tcp	epmap	DCE endpoint resolution
tcp	loc-srv	NCS local location broker
udp	epmap	DCE endpoint resolution
udp	loc-srv	Location Service

ISC doesn't seem to tell me where exactly at Neohapsis this is coming from, however a little searching revealed the above can be found at http://www.neohapsis.com/neolabs/neo-ports/. The official IANA port mapping from http://www.iana.org/assignments/port-numbers also agrees with the above, listing port 135 as:

> *epmap 135/tcp DCE endpoint resolution*
> *epmap 135/udp DCE endpoint resolution*[2]

I've included a snippet from ISS' site, which nicely summarizes all of the above into a few sentences:

> ***Port 135 loc-srv/epmap***
>
> *Microsoft DCE Locator service aka. end-point mapper. It works like Sun RPC portmapper, except that end-points can also be named pipes. Microsoft relies upon DCE RPC to remotely manage services*[3]

Which can be found in its entirety at http://www.iss.net/security_center/advice/Exploits/Ports/135/default.htm.

[2] "PORT NUMBERS" 11 February 2003 URL: http://www.iana.org/assignments/port-numbers (13 February 2003)
[3] "Port 135 loc-srv/epmap" URL: http://www.iss.net/security_center/advice/Exploits/Ports/135/default.htm (1 February 2003)

Description

Using the above ISS URL again, we can learn of a few services that use this port.[4] They are (I've included my own descriptions):

- ***DHCP server*** – a DHCP server allows an administrator to easily pass IP addresses, network routes, DNS servers, or other critical information that DHCP enabled clients will need setting up their interface.
- ***DNS server*** – as the Internet communicates by IP (eg: 63.100.47.46), and not names (eg: www.sans.org), DNS servers provide that name to IP mapping.
- ***WINS server*** – WINS is basically the same as DNS, except that it's a Microsoft protocol, and maps workstation names to IPs.

Let's find out what else, shall we? As it turns out, ISS is a great resource for this. They provide the following links and Knowledge Base articles suggesting many parts of the Microsoft operating system are dependant on port 135:

http://www.iss.net/security_center/advice/Exploits/Ports/groups/Microsoft/default.htm
http://support.microsoft.com/default.aspx?scid=KB;en-us;q150543
http://support.microsoft.com/default.aspx?scid=KB;en-us;q176466

Summarizing the above, we can add (again with my own descriptions):

- ***Microsoft Exchange Server*** – Microsoft's version of a mail server
- ***RPC services*** – Basically works the same was as the UNIX RPC services. When an RPC service starts, they don't generally have static ports, so they register themselves with the portmapper, allowing clients to ask the portmapper what port a certain service is running on.
- ***Microsoft remote administration services***

Protocol

As noted earlier, IANA and others list port 135 as being assigned to both TCP and UDP. A netstat on a Windows 2000 workstation machine agrees (extraneous output snipped):

```
H:\>netstat –an
Active Connections
Proto  Local Address          Foreign Address        State
TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
UDP    0.0.0.0:135            *:*
```

[4] "Port 135 loc-srv/epmap" URL: http://www.iss.net/security_center/advice/Exploits/Ports/135/default.htm (1 February 2003)

TCP and UDP are both protocols that allow machines to communicate and share information with each other. The differences between the two are many. In short, TCP is a reliable protocol, unlike UDP. Specifically, TCP is a connection-oriented protocol, which means that before the two sides can communicate, they must establish a connection with each other. TCP accomplishes this by using flag bits for the conversation setup. These flags are, in order, SYN, SYN/ACK, ACK.

A typical connection would go as such:

Client (I want to connect (SYN)) → Server
Server (Hello, you wanted to connect? (SYN/ACK) → Client
Client (Yes, please! (ACK)) → Server

These steps of setting up a connection ensure that both the Server and Client have a guaranteed, reliable connection (theoretically – TCP can't do anything about routes going down during communication). TCP also employs other methods of reliability after setup, such as Sequence numbers (which ensures data is ordered in the right succession).

UDP on the other hand uses a "best-effort" method. It is left up to the receiving application to do any error checking. This protocol is usually used for programs that require low overhead, like video, audio, and even some multiplayer games - because of the low overhead, UDP is in many situations faster than TCP.

However, we're really talking about RPC here, so let's get into some brief detail of how that works. More in-depth detail will be included much later in this paper.

In simplest terms, RPC (Remote Procedure Call) is a client calling a function (also called a stub, or client stub) locally that sets up an RPC connection with the server and sends the client's data. On the server side, the server takes the data off the network, which is then passed up to the server stub. The server stub in turn passes the client's data up to the application for processing. After processing the data, the application returns the output back to the server stub. The server then sends the processed data back over the network to client. The client's network functions receive the returned data from the network, hands it off to the client stub, and then passes the data back up to the client application. All of this is done transparently, so from the user's point of view the application is executing entirely on the local machine.

It is also interesting to note that there are several types of RPC implementations, DCE (Distributed Computing Environment), Sun's RPC, and Microsoft's implementation (compatible with DCE). This paper will however deal exclusively with RPC as it relates to DCE specifications.

As a side note, some typical RPC applications that one may run into, in addition to the above Microsoft apps listed earlier, are:

- ***NFS*** (used for mounting remote file systems on a local machine)
- ***NIS/NIS+/YP*** (used for allowing other machines to use a central repository of files, such as passwords, host mappings, etc)

Microsoft provides some insight into the inner workings of RPC at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/microsoft_rpc_model.asp, including graphics and diagrams as well. It's an interesting read.

Vulnerabilities

Well, this is where it gets interesting and challenging because, what IS the vulnerability or reason for being included in the Top 10?

Neohapsis (and thus CVE) report the vulnerabilities as:

Vulnerabilities for this port (from CVE)

CVE ID	Protocol	Source Port	Targetport
Description			

Yeah, not a whole bunch to go on, is it? Well, there's got to be something, somewhere. Let's start performing some searches. First stop, www.microsoft.com. I managed to find a few posts specific to RPC and port 135. Though, they're all old, at least in calendar days.

The URL http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B305399 is entitled "Malformed Request to RPC Endpoint Mapper can Cause RPC Service to Fail,"[5] originally released on September 10, 2001. The issue here is that a malformed packet to the RPC service can cause the service, and possibly COM functions, to crash. There is also another article, http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-066.asp, but it seems to be an almost identical advisory. The only real difference is that it mentions Windows 2000.

There is a CVE number (CAN-2001-0509) that relates to the above and can be viewed at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0509.

A different vulnerability at http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B162567 describes an issue in which telnetting to port 135 will cause the CPU to hang at

[5] "Malformed Request to RPC Endpoint Mapper can Cause RPC Service to Fail" 16 October 2002 URL: http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B305399 (1 February 2003)

100%. The article says it was last revisited on 8/9/2001, and effects Windows NT 3.51/4.0.

So basically what we have so far is just a denial of service vulnerability. Hmm, I'm not entirely convinced this is the real culprit of port 135 being on the Top 10 list. I'm not so sure that a DoS would top the charts, just doesn't seem too likely to me. I would have expected to at least hear of some news on major attacks going around targeting these ports. At this point, I decided to search www.google.com and mailing list postings for what other people have been seeing. At the time of this writing, a method of spamming going by the name of "Win popup Spam" is becoming (is still) popular. I had been hearing of friends and colleagues complaining about this, so I had a feeling that this was going to be a large part of what my searches in news groups and mailing lists turned up.

Indeed, I did find quite a bit of postings of various people complaining about this type of Spam. I've included some URLs and excerpts below:

http://marc.theaimsgroup.com/?l=incidents&w=2&r=1&s=port+135&q=b
http://marc.theaimsgroup.com/?l=incidents&w=2&r=1&s=popup+spam&q=b
http://www.dslreports.com/forum/remark,4699774~root=security,1~mode=flat
http://lists.jammed.com/incidents/2002/10/0120.html
http://groups.google.com/groups?selm=avrcld%24o8o%40netnews.hinet.net&oe=UTF-8&output=gplain

In http://marc.theaimsgroup.com/?l=incidents&m=103487466228680&w=2, "H C" writes:

> *Many of the posts to this list have clearly shown that this "messenger spam" is not, in fact, coming in over TCP port 139 (as works w/ 'net send' and the use of the NetMessageBufferSend() API)...rather, it's coming in over DCOM/RPC, and is initiated w/ a UDP query to port 135, the portmapper. The open port could also be used for malicious purposes.*[6]

Also, in SANS NewsBites, December 18, 2002 Vol. 4, Num. 51 (http://www.sans.org/newsletters/newsbites/vol4_51.php):

> *A number of companies have figured out how to exploit Windows Messenger Service to send pop-up spam to Internet users. AOL now block the ports that Messenger Service uses. Messages are accepted by default in Windows 2000, NT and XP; Windows 95, 98 and Me do not have the service enabled.*[7]

[6] C, H. "RE: Source of Windows PopUp SPAM." 17 October 2002 http://marc.theaimsgroup.com/?l=incidents&m=103487466228680&w=2 (13 February 2003)
[7] "SANS NewsBites December 18, 2002 Vol. 4, Num. 51" 18 December 2002 URL: http://www.sans.org/newsletters/newsbites/vol4_51.php (13 February 2003)

Lastly, my personal favorite, a Microsoft KB article (http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q330904):

> *A Messenger service window appears that contains an Internet advertisement that is similar to the following text:*
>
> > *Messenger Service*
> > *Message from* **source** *to* **your_computer_name.ISP_name** *on* **date time**
> > **Message Text**[8]

I find this exposure intriguing, ingenious, and amusing. Since I have little information about the internals of how this type of Spam, or the MS RPC protocol works, this will be a good exercise for myself. I've also come across many mailing list posts with such questions as "what is this" and "how do I stop it," so it is my aim for this paper is to provide detailed research in what it is, how it works, and how to stop it for anyone else that is confused or bombarded with this.

Part 2 – Specific Exploit

Exploit Details

So, what are we looking for in a possible tool? We're looking for anything that makes use of the Windows' Messenger service with the intent of sending advertisements. This should not be confused with Microsoft's Instant Messenger software, "Windows Messenger" (http://www.microsoft.com/windowsxp/windowsmessenger/getstarted/default.asp) . The names are similar, and it can be confusing. Windows Messenger is a real-time chat client, similar to "Yahoo! Messenger," or "AOL Instant Messenger." The Windows Messenger **service**, in a perfect world, would be used for administrative tasks of informing users about important system events. An example might be to tell employees a critical machine will be going down soon. However, this service isn't well protected in a default install of Windows, so as a result, this allows anyone to use the service and port to send whatever message they deem appropriate. This will work on versions of Windows that are NT 4.0 (Server, Workstation, Enterprise), 2k (Professional, Server), or XP (Professional, Home). According to the SANS NewsBites (http://www.sans.org/newsletters/newsbites/vol4_51.php) and PCworld (http://pcworld.shopping.yahoo.com/yahoo/article/0,aid,107754,00.asp), Windows 95/98/ME are not vulnerable as they do not have the service enabled. I tried tacking down 95/98/ME machines to test this, but I was only able to find Windows 98SE machines, and all four that I looked at were not listening on UDP port 135.

[8] "Messenger Service Window That Contains an Internet Advertisement Appears." 31 January 2003. URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q330904 (13 February 2003)

This is such a simple new trick, but it works well. As Email Spam filters become smarter, and administrators crack down on mail relays, etc, what else is there to do but find another way in? Here's where it gets more interesting, though. There have been some discrepancies on mailing lists about whether or not this type of activity is coming in over standard NetBIOS (ports 137 - 139), or whether it is coming in over RPC (port 135). As it turns out, this Messenger Spam (though really it's just "net send") can do either. As more administrators are beginning to block access to NetBIOS to rid themselves of the woes associated with that, these port ranges are becoming unavailable for use. If these ports are unavailable, port 135 (using the UDP protocol) will be used instead. Such products, like Direct Advertiser will skip the NetBIOS ports entirely and jump right to 135 to deliver the payload. Cute.

This type of tomfoolery is what amuses me the most, and it is a good example of how security is an ongoing processes. Current standards of protecting a server or service won't be foolproof forever. Blocking the usual NetBIOS ports has worked well in protecting the end user, but now there is another way to abuse a user's desktop – through RPC.

NOTE: The following products below seem to be quickly moving targets. I frequently try to check up on their products and website to update this paper with their changes. I'd like to think I've made a good effort at it, but sadly, at the time the reader reads this paper, it's quite likely many things have changed already.

Direct Advertiser

Coming up with a tool that fits the bill as described above was pretty easy, really. It seems that most people are pointing their fingers at a product called Direct Advertiser, from http://www.DirectAdvertiser.com/. Their product claims that it can quickly put a message (AKA marketing babble) right onto the desktop of the user sitting at the computer without having to wait for the user to check mail, or email filters getting in the way. Their customers seem to have good things to say about it, http://www.DirectAdvertiser.com/testimonials.html.

Interestingly, this company seems to be anti-Spam. They've got a link on how to disable receiving these messages over the Internet (http://www.DirectAdvertiser.com/optout.html), as well as the standard request of "Please don't Spam, this software isn't intended for that." Though, one could intelligently argue that is what they all say, right? ☺

The program software requirements seem to be standard – Windows platform (2k, XP), 128 Megabytes of RAM, and some sort of connection to the Internet ranging from dialup to DSL or better.
The Direct Advertiser product can be purchased for $700 per license, or $200 for a 30-day license. A demo can be downloaded from http://www.DirectAdvertiser.com/download/directadv_demo.exe (for v1.0, ~482KB)

It's interesting to note that Direct Advertiser claims to be the original, and that there are others out there trying to do a knock-off of them. Huh… Go figure…

Description of variants

Finding variants was a bit of a tougher endeavor. Most everything I found was either related to Direct Advertiser, products preventing this type of Spam (more on this later), or how to prevent web browser pop-ups. I did however manage to stumble across a few other products at, http://BroadcastAdvertiser.com/, http://BroadcastMarketer.com, http://InstantAdvertiser.com, and http://SlySender.com. There's actually some pretty good commentary at http://www.badads.org/december02.shtml about all five of these companies. It even includes names and phone numbers of the people who own the companies.

Broadcast Advertiser
Hmm, there are quite a few similarities here between the products of Direct Advertiser and Broadcast Advertiser. Even their web page has a similar format and layout – though, the anti-Spam icon on the main page is a nice touch when needing that little extra bit of originality. Initially when I started on this paper, Broadcast Advertiser had a product called QuickSend v1.2, which appeared to be nearly the same product as Direct Advertiser's in every way. However, this product has since disappeared, the links are gone, and so I've pulled discussions and comparisons of their products (QuickSend v1.2 and BA v2.0.1) from this paper in favor of their new product, BA v3.0.1.

The functionality of their 3.0.1 product seems to far outperform that of Direct Advertiser, but the requirements are nearly the same (supporting 2000, NT and XP). Their demo can be retrieved from http://BroadcastAdvertiser.com/demo.htm, and weighs in at approximately 1.3MB. The cost of the product will set the reader back $149 for the Standard edition, or $249 for the Pro edition. The only discernable difference (from the web page) in the Pro is that it allows one to send up to 10,000 simultaneous messages.

Both products come with a database of 1,926,333 IP ranges from around the world, and include their free "Campaign Database Builder" software.

Other features include:

- Creates a list of IPs having been successfully targeted.
- Saves one's progress in order to continue later.
- Importing of custom IP ranges.
- A status report of IPs having been successfully targeted.
- High levels of granularity, such as targeting IPs by zip codes or countries (this is unique).

Broadcast Marketer
As with the other products, BroadcastMarketer's product makes use of the Messenger Service and requires Windows 2000 or XP. Their demo can be found at http://www.broadcastmarketer.com/download/bmtrialpackage.exe (for v2.0, ~590KB). The cost is $499.95.

Broadcast Marketer's EULA specifically forbids reproducing any part of their software or reducing it to a written form. I will abide by their wishes and this is as far as I will go with this product.

Instant Advertiser
As of the time of this writing, I could not get their website, http://www.instantadvertiser.com, to respond. Apparently there are no DNS records for their site. Maybe business hasn't been so good for them? ☺ Hopefully this situation will resolve itself in the near future so that I can include some content here.

Sly Sender
Interestingly, this product is shareware (software that is freely distributed, but is often limited to a certain trail time length, or the software is in some way crippled until it is registered). Hence, the only places I've really seen to download it are places where many other shareware products are offered, such as: http://www.sharewarejunction.com/info.asp?ProductID=8465 and http://www.softdeko.8m.com/mass_mailing_software.htm (although all these places just link back to their home page). Their home page is at http://www.SlySender.com/, complete with flash animation too. They've got a little demonstration on their site about how to use the product, which can be viewed at http://slysender.com/usage.html. The current version of Sly Sender is 4.0.

Their feature set appears to be basically the same as all the others:

- Log successful sends to file
- Sends to IP ranges
- Keep a record of successes and failures
- Verify whether a remote IP will accept a message (this is unique)

Version 4.0 is available for download from the above listed URLs. The required OS to run the software is NT, XP, or 2000 and the cost is $159.99 (as stated at the primary site).

In a nutshell, the above products seem to be generally the same, more or less. In actuality, I would really expect them to be, because all one really needs is to be able to quickly locate IPs and send them a message. It's funny to see the sale pitches on the commercial products' web sites. They'll pitch things like "our product is the only one that can save your progress" and then I'll go to look at another product vendor, and they'll say the same thing. Some will claim that this

type of activity is not Spamming, yet then proceed to convince the end user that this is better than email because it will hide your IP, and the end user won't get into trouble with his ISP for mass emailing. However, I digress. For the remainder of this paper, I'm going to focus on the product from Direct Advertiser. Direct Advertiser seems to be the most popular, based on the number of "press releases" I've seen and the number of people on various lists pointing their fingers at this product. Some of these can be found at http://www.theregister.co.uk/content/archive/27634.html, http://news.zdnet.co.uk/story/0,,t269-s2126516,00.html, http://www.wired.com/news/technology/0,1282,55795,00.html, http://archive.nznog.org/2002-10/msg00092.html, and http://www.gdargaud.net/Hack/NoSpam.html.

Based on the research done by browsing the respective products' websites, they all seem to be similar in their feature set. The following section will detail the differences in actually installing and interacting a bit with them. I feel this would give the reader a better understanding of the differences. Direct Advertiser's product is first, as in order to compare the products to Direct Advertiser we must first understand the feature set and functionality of Direct Advertiser.

Direct Advertiser:
The product doesn't seem to tell me its version number, only that it is a demo. I can assume that it is version 1.0, as that is what is listed in the screenshots from their site.

After running the setup binary, the user is presented with an End User License agreement with various clauses to which a user must abide by. This is a typical license, mentioning such things as no liability for any damage this product causes, mentions this software requires a license, and only one copy of the software may be used with this license. There is also an anti-Spam policy, which specifically says this product should not be used for spamming. If someone is caught spamming with this product, they reserve the right to refuse to do tech support, upgrades, or any further business with the individual(s).

After starting the application, one is prompted to enter their email address that they used for their payment. "Demo" will also work, if just demoing the product. The demo version will only send a message to one IP address, as opposed to the registered version, which can send to a range of IPs.

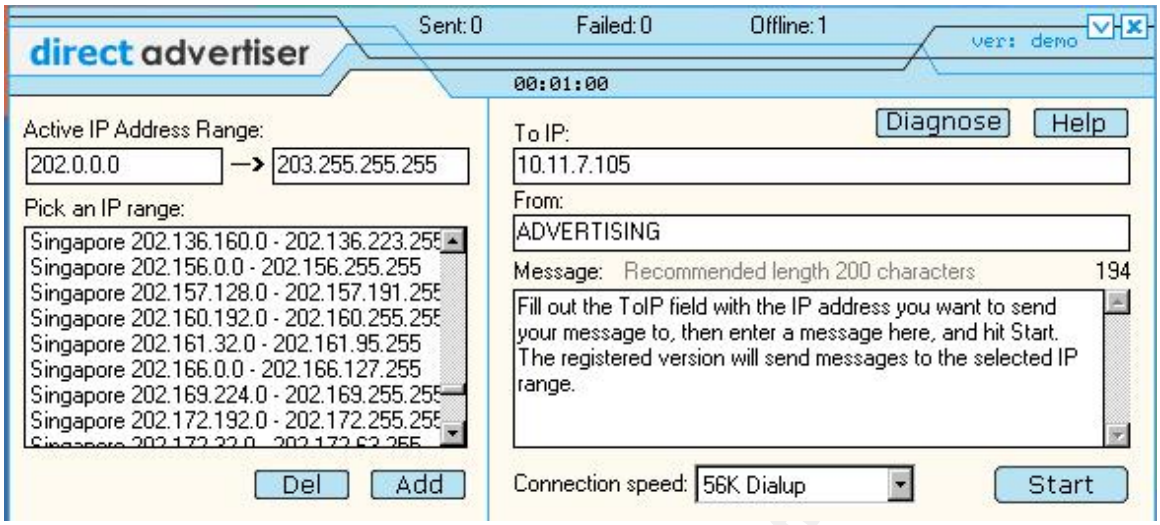
The application itself is pretty simple and straightforward. On the left-hand side, the user is presented with a choice of where to send the message. One can either pick a range of IPs, or a predefined range from the scroll down box. This scroll down box includes various default IP ranges, such as U.S. (ex: "US-Qwest_Communications, US-AT&T, etc), Bangladesh, Estonia, Russia, and so on. Changes can be made to the list with the "Add" and "Del" buttons.

On the right-hand side are a number of different items that are mostly related to sending the message. Here the user can see what IP the message is being sent to, from who (this is something like "WEBPOPUP" or "ADVERTISING." This is basically who the messenger services lists as the sender) and the message to be sent. A nifty little feature with the message box is that it will keep count of the number of characters that have been typed in so far. This is useful, as they recommend a length of 200 characters. Lastly, the user can also select their connection speed, or diagnose the computer. Not sure the real function of this is yet, it seems silly and inaccurate at best. Clicking on the "diagnose" button provides the user with the following information:

```
Diagnostic Information
**************************
Date/Time of diagnostic: 1/25/2003 4:38:48 PM
RAM: 256 MB
CPU Speed: 850.0 MHz (estimated)
Ping 1: 4194324 ms
Ping 2: 4194324 ms
Ping 3: 4194324 ms
Estimated upload speed: 641.03 KBytes/sec
Message-length: 194 characters
From-length: 11 characters
Connection-speed: 56K Dialup
```

That's some amazing upload speed for a 56K dialup. Apparently the diagnostic Connection-speed is taken directly from what the end user has set their speed at. Looking at tcpdump, the ping routines appear to ping three hosts, **w1.rc.vip.scd.yahoo.com**, **altavista.com**, and **www.google.com**. The ping routine does not seem to catch errors, as can be observed from the ludicrous ping times above (my external firewall is not allowing any ICMP into the network).

Lastly, at the top, is some status information in the form of how many messages were successfully sent, failed, or attempts to offline machines. Attempts to send a message to a machine that is either firewalled or offline will quickly elicit a message box saying the host is not available. The following is a screen shot of the application:



Broadcast Advertiser (v3.0.1 Demo):
The first screen that is presented to the user during installation is a typical "we recommend you close all applications before proceeding with the install" message box. Immediately following that the user is presented with a basic EULA that's really there only to tell the user Broadcast Advertiser is not responsible for any damages that could be caused by their product.

Right off the start, this product is unlike Direct Advertiser in every way. First, the application goes right into the main screen, there is no popup box requiring the user to first enter an email address or any other information in order to start the program. My first impression after starting it up is of "wow, this looks nice." Like all the others, the unregistered version is crippled. This demo will only send one message at a time. When registered, the application says it will support up to 10,000. As this demo will allow one to use ranges (albeit one IP at a time), this product would seem to be a better choice for anyone who wanted to send Messenger Spam and not register the product.

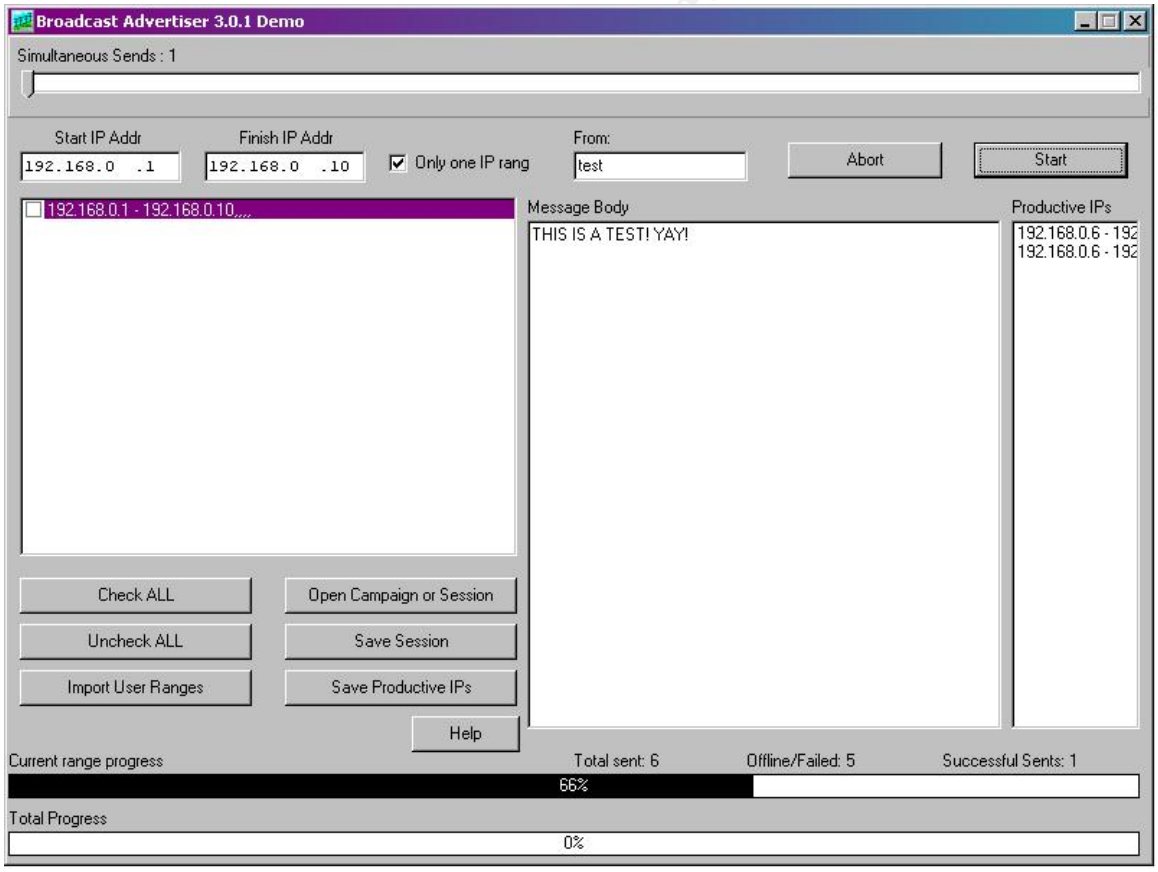
So far, this seems to be the most useful interface. At the top, is what appears to be a sliding bar that will represent the number of simultaneous connections (I can't test this, as this is grayed out in the demo). On the left there are two input boxes where the user can enter starting and finishing IPs to make up a range. There are no automatically imported ranges present after starting the application. Ranges can be imported manually from files that are included with the product, and with their newly updated database for version 3.0.1 the user can target cities, states, countries, zip codes and the like. They have truly put a large amount of work into the database.

There are a various number of operations that can be performed on the ranges. The user can "Check ALL" or "Uncheck ALL," which will determine which ranges will have a message sent. The user can also open a range, session or campaign (which best I can tell is just a list of the areas the user might want to send

messages to), as well as save them. There's a nice feature here that will also allow someone to save the IPs where a successful message was sent. This is useful for weeding out IPs that don't exist, are firewalled, or don't have the Messenger Service running. To the right-hand side of the screen is an area that allows the user to define the "From" for the message, as well as the message, but it does not keep track of how many characters have been typed. There is also no recommended character limit for the message. It does however keep status of total messages sent, failed, and successes. Though, failed sends don't seem to elicit any message - they are just silently skipped over. The user can see what ranges had successful sends to the far right under the "Productive IPs" box.

Another neat aspect is that there are progress bars for the current range (how close to the end of the range is the sending process), and the overall total progress. There is no "connection speed" select, or a "Diagnosis" button to be found.

The following is a screenshot:



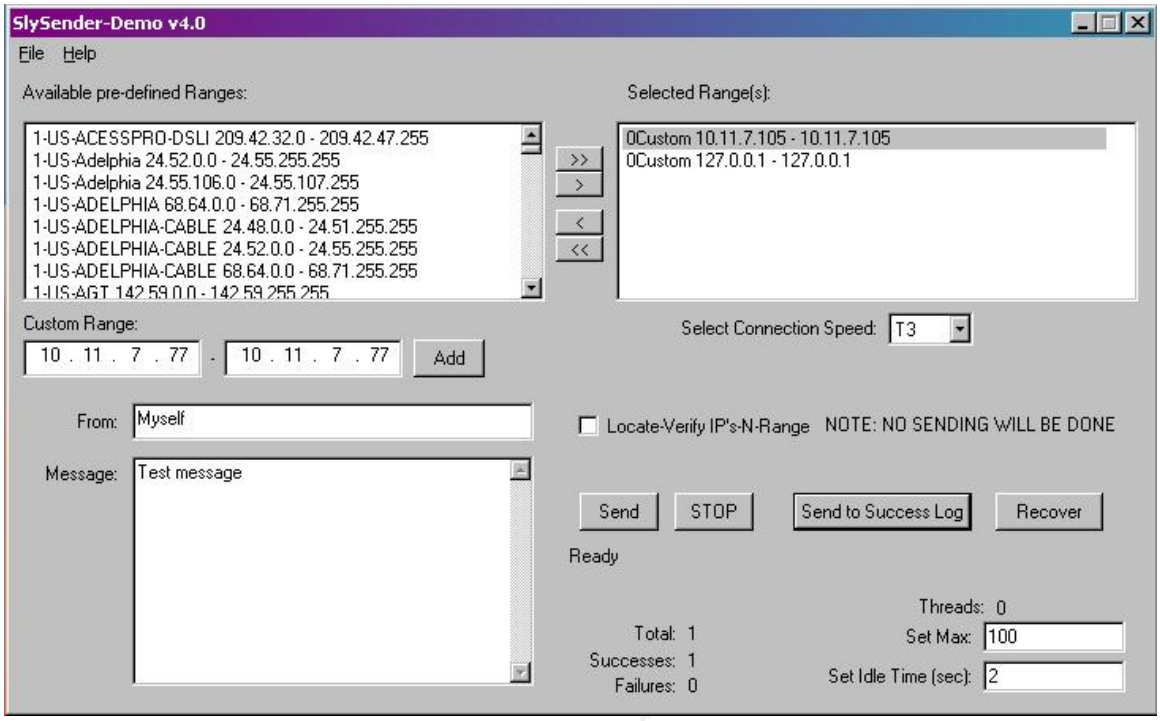
Sly Sender (4.0):
Like the other software, this install starts out with the usual babble of closing all programs. Sly Sender also has its own EULA, saying among the usual things, that this software may violate your ISPs Terms of Service and the ISP should be

consulted before running their software. The EULA also mentions that the Messenger Service must be started. This demo is limited to sending to one IP at a time. It also has crippled some other functions as well, like "Send to Success Log."

Sly Sender also has a very nice interface. Incidentally, it looks somewhat similar to the Broadcast Advertiser 3.0.1 demo. It has all the usual, expected input boxes such as the pre-defined IP ranges, custom ranges, From, Message box, connection speed, etc. What this product does that is interesting is that it has a "Locate-Verify IPs" button. This function, instead of sending a message to a host, will just test if the host can/will accept a message. Another unique setup item here is the use of setting thread preferences. The max is 99999, and the minimum is 1. In addition to that, the user can also set the idle time for a thread. Sly Sender's help is also unique unto itself - they're audio files! None of that ever-so-cumbersome reading to get help! Lastly, the program offers a "Recover" button, incase the sending computer crashes. Sly Sender can only keep the next ~40,000 IPs that it had in memory before the crash, so pressing "Recover" will only recover those next IPs.

As a side note, I apparently can enter only two IPs into the demo product - after which I can't enter anymore, or remove what I've added. The only button (the << symbol) that would appear to remove them is disabled in the demo version. Through a bit of sleuthing, I've found that Sly Sender stores these IPs in *C:\winnt\win.ini*, so changing or removing IPs would can be done from there. Cumbersome, but it works.

Below is a screenshot of the application:



Network Traffic Differences:
While testing the products above, I also had tcpdump running so that I could gauge how each product utilizes the network, and whether they did anything unique among themselves. The results were that all tests returned pretty much the same type of network traffic. All focused on UDP port 135, skipping any normal NetBIOS traffic on port 139 altogether. The only real differences were in Direct Advertiser and Broadcast Advertiser v3.0.1. Both of these products attempted a single ICMP echo request first before attempting to deliver their payload. Please see “Appendix A” at the end of this paper for the tcpdump logs.

Now that we have a good understanding of what the products are capable of, let’s discuss the protocols that are used to deliver these marketing messages.

Protocol Description

So, how does all this come together, and what does it mean? Let's start out with a better description of what the Messenger Service is, and then we'll go into the protocols that deliver these marketing messages. As noted earlier, the Messenger Service's primary function is to deliver messages to a user's desktop. There are two ways to create a message, "net send," or by the Alerter Service. "Net send" can be invoked by end users in a command shell to send messages to various users, computers, domains, etc, and the Alerter Service is in charge of informing users and computers of Administrative alerts. More information on “net send” can be found at the end of the section entitled “*How the exploit works*” later in this paper.

Some example uses of "net send" might be for one user to inform another of a change of plans for lunch, or in my case where I work, inviting me to take a break and go play some foosball. Virus warnings, UPS (uninterruptible power supply) warnings, and other potentially critical system events are examples of pop-ups one might see from the Alerter Service.

Having the above in mind, there are a few conditions that must be met in order for pop-up Spam to succeed. According to Microsoft, these are (I've included my own commentary in parenthesis)[9]:

- **The Messenger service is started** (This is located in the "Services" tab under "Administrative tools.")
- **The Remote Procedure Call service is started** (Again, this is under the "Services" tab. It is likely that it is already started since so many things depend on it).
- **Inbound NetBIOS (NetBIOS over TCP/IP) and UDP broadcast traffic is turned on for your Internet connection**. (This isn't really necessary in this case, as the type of Spam being discussed here skips over NetBIOS ports completely in favor of the RPC ports. It should be noted though that Spam *could* use these NetBIOS ports.)

Now that we've got a better understanding of what the Messenger Service is, this brings us to where we left off, RPC port 135. How do the underlying protocols work together to bring us the popup Spam? The underlying delivery protocol is UDP, so I'll start there. Granted, this could really get granular and go into other protocols, such as IP and Ethernet, but that is getting off on a tangent.

UDP, or User Datagram Protocol, is an uncomplicated protocol and is often referred to as a datagram-oriented protocol. Datagram-oriented protocols haven't any means of prior session setup before data is sent from the sender to the receiver (TCP's session establishment can guarantee the remote end partner is available and ready to accept data – this is established with the SYN/SYN+ACK/ACK sequence as discussed earlier). This means that UDP provides no guarantee that any given UDP datagram will arrive at the intended destination, or that the destination even cares or is able to respond. Since this type of error checking is not built into the protocol, this leaves any error checking up to the application. UDP does however get a little bit of help from ICMP (Internet Message Control Protocol). If a UDP datagram arrives at its destination, and the port is not available, the remote host will send an ICMP "port unreachable" error to the sending host. This informs the sending host that delivery has failed. However, if for some reason the ICMP error never reaches the sending host (perhaps it is firewalled), there is no way to tell from the sending side whether the data was successfully received. This is one of the drawbacks of UDP based applications. This same type of scenario is also possible if the

[9] "Messenger Service Window That Contains an Internet Advertisement Appears." 31 January 2003. URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q330904 (13 February 2003)

remote application is accepting data, but makes no attempt to acknowledge the sender. This is where error checking in UDP applications can come into play with timeouts and resends. This type of "send and forget" personality of UDP is what gives it a lower overhead than other protocols, like TCP.

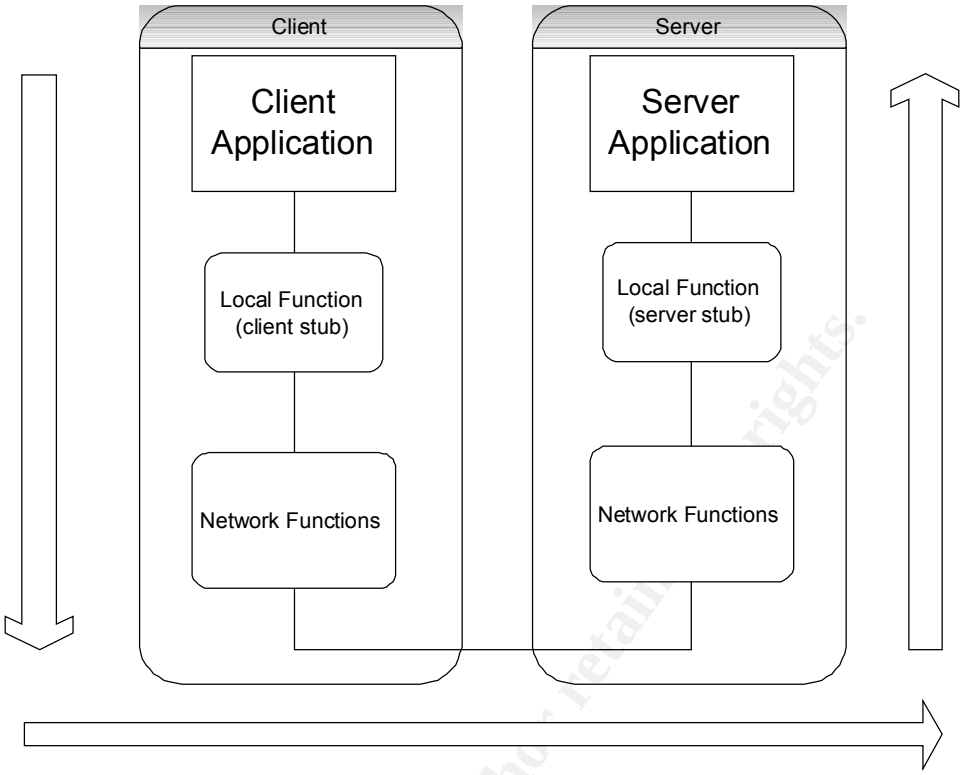
The UDP protocol has a set maximum size that can be determined with a bit of basic math. Starting with IP, the maximum size of an IP datagram is limited to 65535 bytes. IP has a minimum header size of 20 bytes or a max of 60 bytes. UDP has a static header size of 8 bytes. If we subtract all of this, we potentially have a maximum UDP payload size of 65507 bytes for data. Receiving a Messenger Popup message containing that much data is unlikely for a couple reasons. The application itself may not be capable of sending (or reading) that much data, or there may be limitations within the network stack of the host.

So, moving right along, let's dive into what RPC is - some information from earlier will be rehashed. RPCs in general, because of the number of DoS attacks and exploits commonly associated with the service, are considered risky business and so the consensus is that they are evil and should be avoided. While this is true to some extent, RPCs are still very useful and remain popular. What makes RPCs so popular is that they create a solution to common problem:

Clients have data and applications they need to interact with, but in order for everyone to have access to said data and applications, they would have to be housed on each client. This presents the problem of synchronizing the resources between all clients who have access to it. RPC provides the functionality to allow the clients to think they're accessing the data and application functions locally, but in fact are actually communicating with the RPC server. RPC facilitates this functionality through use of stubs, and as long as both sides have compatible implementations of the RPC protocol, client and server are completely platform independent of each other. A client stub is the local function that a client application is calling, which is actually a function that's located on the remote RPC server. Client stubs take the client input and package it into a form suitable for network delivery, then send it out to the remote server.

On the server side, the client's request is picked up by the library functions of the server and passed up to the server stub. The stub accepts the data, unpacks the data into the format required by the application, and calls the real function. The data can now be processed by the application.

Conceptually, this would look similar to the following diagram:



The RPC server's response will be nearly the same, just in opposite order. So with that in mind, there's only one problem. RPC was designed so that applications don't have a static service port (http (80) or telnet (23)). So, how do the library functions know what dynamic port a particular application is listening on? The possible range of ports starts at 1024, and can go as high as 65535 (limited by the 16-bit port field in the TCP and UDP headers). That certainly is a wide range of ports to guess where a service is! This is where port 135 (TCP or UDP) comes into play. For Microsoft, this is the Endpoint Mapper. Under Unix conventions, this would typically be called Portmap (or portmapper), which runs on port 111. These program's ports are static, and the two programs are crucial in the RPC world. The Endpoint Mapper's only function is to map service ports to their respective applications. That raises the question, "how does the Endpoint Mapper keep track of what applications are mapped to what port?" That's the job of RPC "service numbers," which are really just a unique identifier that is specific to each program. Now, gluing it all together, this process would go something like:

- RPC Endpoint Mapper starts.
- An RPC service starts. During its setup it must register its Unique Identifier (UUID) for the service it is providing with the EndPoint Mapper.
- The Mapper associates the UUID to a port for later use when clients ask for the service.

Later, when a client wants to talk to an application on the RPC server, typical

communications would go as such:

- RPC client asks the Mapper what port a specific UUID is listening on
- The Mapper checks its mapping for whether that UUID is registered, and if so, on what port it is listening
- The Mapper returns the port number (or an error if the service isn't registered)
- The client then connects to the application on the port returned
- Application responds back to client

Without unique IDs and the Endpoint Mapper, RPC would cease to function properly. We can get a listing of what services are being offered by using "rpcdump," available from the Windows 2000 Resource Kit (http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/rpcdump-o.asp). Microsoft also has another tool, Portqry.exe, available at http://support.microsoft.com/default.aspx?scid=KB;EN-US;q310298& that can also enumerate RPC services. I've included some output snippets and some brief explanations from rpcudmp and portqry below:

```
C:\>rpcdump /v /v for verbose
Querying Endpoint Mapper Database...
9 registered endpoints found.
ProtSeq:ncacn_np the type of protocol, see “Appendix B” for a list of these
Endpoint:\PIPE\scerpc what the service is connected to
NetOpt:
Annotation:Messenger Service
IsListening:NOT_PINGED
StringBinding:ncacn_np:\\\\PENELOPE[\\PIPE\\scerpc]
UUID:5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc the unique IDs, as discussed
ComTimeOutValue:RPC_C_BINDING_MIN_TIMEOUT
VersMajor 1  VersMinor 0

ProtSeq:ncacn_np
Endpoint:\PIPE\ntsvcs
NetOpt:
Annotation:Messenger Service
IsListening:NOT_PINGED
StringBinding:ncacn_np:\\\\PENELOPE[\\PIPE\\ntsvcs]
UUID:5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
ComTimeOutValue:RPC_C_BINDING_MIN_TIMEOUT
VersMajor 1  VersMinor 0

ProtSeq:ncacn_ip_tcp
Endpoint:1025 This service is listening on port 1025/tcp
NetOpt:
Annotation:
IsListening:NOT_PINGED
StringBinding:ncacn_ip_tcp:10.11.7.105[1025]
UUID:378e52b0-c0a9-11cf-822d-00aa0051e40f
ComTimeOutValue:RPC_C_BINDING_MIN_TIMEOUT
VersMajor 1  VersMinor 0
```

[...]

Portqry is basically the same, just a different format.

```
C:\>portqry -n 10.11.7.105 -p udp -e 135
Querying target system called:

 10.11.7.105

Attempting to resolve IP address to a name...
IP address resolved to dhcp-105.z.[removed].org

UDP port 135 (epmap service): LISTENING or FILTERED
Querying Endpoint Mapper Database...
Server's response:

UUID: 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc Messenger Service
ncadg_ip_udp:10.11.7.105[1026]

UUID: 1ff70682-0a51-30e8-076d-740be8cee98b
ncalrpc:[LRPC0000026c.00000001]

UUID: 1ff70682-0a51-30e8-076d-740be8cee98b
ncacn_ip_tcp:10.11.7.105[1025]

UUID: 378e52b0-c0a9-11cf-822d-00aa0051e40f
ncalrpc:[LRPC0000026c.00000001]

[...]
```

It's important to note that while one can get a listing of what ports are listening using "netstat," it will not give up what specifically is listening on that port. Netstat use in this area is only useful in confirming there is really something listening on said port. To wit:

```
netstat -a
[...]
 UDP    0.0.0.0:1026           *:*
[...]
```

However, Foundstone (www.foundstone.com) provides a tool by the name of "fort" that will tell us what ports are bound by what service. The tool can be downloaded from http://www.foundstone.com/knowledge/proddesc/fport.html. Below is a snippet of what is bound to port 1026/UDP:

```
[...]
216   services      ->  1026  UDP   C:\WINNT\system32\services.exe
[...]
```

Here we can see that services.exe has this port open. Services.exe provides the Messenger Service.

Though, let's get back to the topic here, I'm going off on a tangent. As we'll see later, the "problem" here is that the Messenger Service registers itself with the Mapper, and that is why we can send Messenger Spam via port 135.

How the Exploit Works

This section may prove to be a bit more difficult, as the service being exploited doesn't have a programmatic error that would yield administrator, heightened privileges, a remote shell, etc, nor do I have access to any source code of the tools being used. Quite simply, the heart of the problem is that unauthenticated users, either local or remote, can popup random and unwanted messages onto a user's desktop. It's also worth noting that it really isn't even just about random messages. The fact that anyone can access RPC services creates a hole that anyone can use for malicious purposes, or information gathering (remember rpcdump) as well. That unto itself could lead to trouble.

Let's think about how this exposure is exploited. Since this is commercial software, Direct Advertiser's source code is not available for browsing. We'll have to determine what is going on based on what I see via the network traffic; hence this section will bleed directly into the network diagram. More information about what programming might go into a tool like this will be discussed in the section "*Source code / Pseudo code*." There I will dive into what a sample program does to send messages from one machine to another. The actual exploitation of Direct Advertiser (and any tool, really) is very simple. I could almost sum it up in saying "they're just taking advantage of Microsoft's lax security settings." Hopefully this will be useful to the Internet community, I've not seen anyone go into this much detail in regards to what is specifically happening with an RPC message when received.

From the moment the user clicks "Start" within Direct Advertiser, the application begins with sending a single ICMP echo request to the target. The total size for this ICMP echo request datagram is a total of 74 bytes (14 for the Ethernet header, 20 for the IP header, 8 for the ICMP header, leaving 32 bytes for ICMP payload). The data portion of the ICMP datagram appears to be random nonsense. Failing a ping response, Direct Advertiser returns the error "The computer you are trying to send the message to is either offline or behind a firewall!"

The second datagram sent is basically the exploit in action. This UDP datagram is the RPC request for the Messenger Service and contains the entire message payload. However, the user can still run into problems here, too. If for some reason the Messenger Service isn't running, or not registered with the Mapper, Direct Advertiser will abort with the message "Your message was NOT sent due to NETBIOS ERROR! NetBUI protocol is not configured on target computer." It would appear from the error that Direct Advertiser tries a couple of other different

methods of delivering its payload. I did not however notice any new attempts other than the failed RPC try. Please see “Appendix C” for a network trace of an RPC reject due to an unregistered interface.

The total datagram size is really entirely dependant on the size of the message from the sender. In this particular datagram, the actual text message is 194 bytes (we'll look at the actual datagram in the “*Diagram*” section). The total size for this datagram is 377 bytes; this includes the 14-byte Ethernet header, 20-byte IP header, 8-byte UDP header, 80 bytes for the DCE RPC header (which is a fixed length) and 255 bytes for the payload message. The payload, also known as the PDU body, will always be larger than the actual amount of user data sent. The reasoning for this is two-fold. First, because the data is prefixed with the "From" name, and the destination IP, and secondly, the PDU header fields have an alignment of "*0 MOD min(8,sizeof(field))*," meaning PDUs that have a body will be padded with zeros so that it will always end on an 8 byte boundary.

At this point, the message has been delivered to the destination machine (yes, the entire text message was contained in the second datagram sent to the server at port 135). However, what happens next is a bit of an interesting twist. I would think that the message would just pop up now, but it doesn't.

The destination machine responds with a request that is called conv_who_are_you. Keep in mind this is the **destination** that generates this traffic, and from a high port >1023. In my tests, the server connected from its high port, to the source port (1348) of the remote host. Most places I've seen will allow all outbound traffic from internal machines. This is a good reason to only allow what is needed from inside->out, as well as outside->in. When a server has no previous knowledge of a client, and it receives a non-idempotent (more on this in the next section) request from said client, it will initiate this operation in an effort to get a couple of questions answered about the requester. I interpret this as a "oh, I wasn't expecting you... and you are?" This function requests from the client its sequence number/Activity UUID (more on this in the next section). This prevents duplicate requests by comparing the sequence numbers. If the current sequence number is higher than the original request that caused the server to generate the conv_who_are_you request, then it is ignored as a duplicate. Some people have reported being able to block the popup Spam by not allowing these outbound connections. I have seen the same results in my tests. If Direct Advertiser does not get a response to its initial request due to outbound filters on server blocking UDP >1023, we will begin to see DCE RPC datagrams of the type "ping." Ping packets contain no data, and are merely a method of inquiring about the status of a pending request. If there are inbound filters on the server, blocking inbound UDP >1023, when the server responds with the conv_who_are_you, the client will not be able to return a response to the server. This again will cause "Ping" requests from the client - these go over port 135. The server will receive this and respond with a "working" packet type, which means that it is working on a request and eventually try again with the

conv_who_are_you packet. This loop continues until the client sends a cancel request, and an error message pops up with Direct Advertiser. For either situation, the error message suggests that it tries other methods of sending the message if RPC fails, "Your message was NOT sent due to NETBIOS ERROR! NetBUI protocol is not configured on the target computer." I did not see any NetBIOS related traffic though. Please see “Appendix D” for dumps of how this looks when firewalls get in the way.

Now, assuming that all datagrams aren't inhibited, the client at this point will respond to the server's request. In my tests, the client responded to the request with conv_who_are_you2. It is similar to the previous "who are you," with a minor difference. This difference is a unique CAS (Client Address Space) UUID (Unique UID) is also returned/requested.

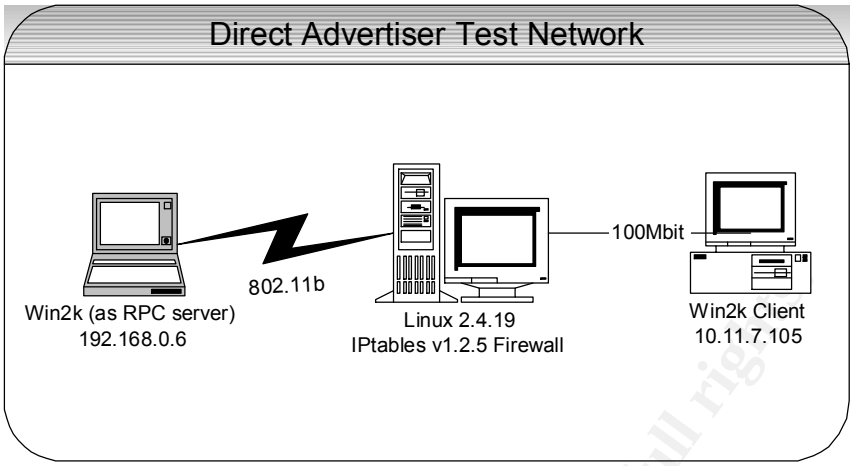
The remaining conversation consists of 3 packets that are acknowledgements of the previous "who are you" request, and the original request of the client.

Now we must ask ourselves one question though. Why is it that these programs (Direct Advertiser in specific) can exploit this vulnerability? Again, Microsoft doesn't provide their source code; hence it is unavailable to me so I can only speculate. However, it's not a real difficult subject as to what the real hole being exploited is and what should be changed. Let's get to the dialog! As noted earlier, this is a simple concept: Microsoft's lax control over the RPC service. To put it bluntly, it is an application design flaw that just anyone from anywhere can send a random, useless message to anyone else at any location. It seems silly to me that Microsoft hasn't thought about putting in some sort of Access Control into their RPC services. There is nothing that limits who (users) or from where (IP) a message can be received. Granted, within a UDP datagram, the user can be spoofed, as well as the source IP (the sources may already spoofed in some homegrown applications. I'd be surprised if that hasn’t at least been attempted somewhere). So, this would only offer marginal security. Hmm. However, what about authentication? One doesn't have sshd giving out free shells to anyone that connects to it. Authentication would be workable, RPC services could just require authenticating against the domain.

Of course, one could also blame administrators as well for not adequately protecting their machines, or having adequate firewall rules, but that is neither here nor there, as it has no bearing on what it is about the service that is exploitable.

Diagram

At this point, I'd like to go into more fine-grained detail of what a typical run from start to finish would look like from a network standpoint. The semantics of the various tools will be covered in the next section. The test network consists of the following:



I will refer to “client” as the RPC client where Direct Advertiser is installed. Packet captures were recorded from the client with Ethereal, version 0.9.9, and can be found at http://www.ethereal.com. Packet dissection for this section was performed via tcpdump version 3.7.1 with libpcap version 0.7 (both found at www.tcpdump.org). First off, let's define what a typical line of output from tcpdump (using */usr/sbin/tcpdump -enr da.tcpdump -X -s 1514 -vv*) looks like:

```
17:45:02.887815 0:50:8b:f0:b5:30 0:c0:f0:6d:8:73 ip 377: 10.11.7.105.1348 > 192.168.0.6.135:
[udp sum ok] udp 335 (ttl 128, id 10734, len 363)
```

17:45:02.887815 - Time packet was captured.
0:50:8b:f0:b5:30 - Source MAC address
0:c0:f0:6d:8:73 - Destination MAC addresses
ip 377 - The entire size of the packet (This includes the Ethernet header)
10.11.7.105.1348 - Source IP and source port
> - Direction
192.168.0.6.135 - Destination IP and destination port
[udp sum ok] - UDP checksum checked out ok.
udp - Protocol used
335 - Entire size of the UDP datagram payload
ttl 128 - The Time-To-Live remaining on this packet. Each router or gateway will decrement this by one. When the TTL reaches zero, it will be discarded to prevent a packet getting stuck in a loop and living forever.
id 10734 - Each packet has an ID associated with it. This is used primarily in fragmentation where each fragment will get the same ID so the other side is able to determine what fragments go together.
len 363 - The entire size of the packet, this includes the UDP payload, UDP header, and the IP header.

In the discussions below, I will also be including the payload in both HEX and

ASCII; this will immediately follow each line of output, as described above. So, on with the show!

```
17:45:02.876907 0:50:8b:f0:b5:30 0:c0:f0:6d:8:73 ip 74: 10.11.7.105 > 192.168.0.6: icmp: echo
request (ttl 128, id 10733, len 60)
0x0000   4500 003c 29ed 0000 8001 3eb2 0a0b 0769      E..<).....>....i
0x0010   c0a8 0006 0800 1681 0200 0200 0f00 0000      ................
0x0020   0000 0000 48c3 d700 003b 2a81 3011 db00      ....H....;*.0...
0x0030   70ff 5901 4f27 4000 48c3 d700                p.Y.O'@.H...
```

This is the first packet we'll see when using Direct Advertiser. It is a simple ICMP echo request (ping), used only to verify if the host is up. Notice that length of the entire packet is 73 bytes long, or 60 bytes if the Ethernet header is ignored. Subtracting the 20-byte IP header and 8-byte ICMP header, we are left with 32 bytes of payload for the remote host to return to us. We'll notice also that the TTL is 128, which is the default for windows 2000. The payload is highlighted above.

```
17:45:02.881092 0:c0:f0:6d:8:73 0:50:8b:f0:b5:30 ip 74: 192.168.0.6 > 10.11.7.105: icmp: echo
reply (ttl 127, id 18049, len 60)
0x0000   4500 003c 4681 0000 7f01 231e c0a8 0006      E..<F.....#.....
0x0010   0a0b 0769 0000 1e81 0200 0200 0f00 0000      ...i............
0x0020   0000 0000 48c3 d700 003b 2a81 3011 db00      ....H....;*.0...
0x0030   70ff 5901 4f27 4000 48c3 d700                p.Y.O'@.H...
```

Here we see the second packet, which is the target machine returning the initial ICMP echo request. This packet is pretty much identical to the previous packet, with the only real difference being that the ICMP type is now an "echo reply." We can also notice that the TTL is 127, and knowing that the destination host is a Windows 2000 machine, we can infer that this machine is located one hop away.

The remaining traces are really where it gets really interesting. Those ICMP traces are more or less common, and there's nothing terribly unique about it. The following datagram is really the meat-and-potatoes of the vulnerability, so I'll go into greater detail about what is contained in the datagram. This is what the third datagram sent looks like (second datagram sent from the client):

```
17:45:02.887815 0:50:8b:f0:b5:30 0:c0:f0:6d:8:73 ip 377: 10.11.7.105.1348 > 192.168.0.6.135:
[udp sum ok] udp 335 (ttl 128, id 10734, len 363)
0x0000   4500 016b 29ee 0000 8011 3d72 0a0b 0769      E..k).....=r...i
0x0010   c0a8 0006 0544 0087 0157 6aec 0400 0800      .....D...Wj.....
0x0020   1000 0000 0000 0000 0000 0000 0000 0000      ................
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0      ......{Z........
0x0040   4fb6 e6fc 58f4 c1aa 68a4 9047 a3a9 4ba6      O...X...h..G..K.
0x0050   bb36 c1f7 0000 0000 0100 0000 0000 0000      .6..............
0x0060   0000 ffff ffff ff00 0000 0000 0c00 0000      ................
0x0070   0000 0000 0c00 0000 4144 5645 5254 4953      ........ADVERTIS
0x0080   494e 4700 0c00 0000 0000 0000 0c00 0000      ING.............
0x0090   3139 322e 3136 382e 302e 3600 c300 0000      192.168.0.6.....
0x00a0   0000 0000 c300 0000 4669 6c6c 206f 7574      ........Fill.out
0x00b0   2074 6865 2054 6f49 5020 6669 656c 6420      .the.ToIP.field.
0x00c0   7769 7468 2074 6865 2049 5020 6164 6472      with.the.IP.addr
0x00d0   6573 7320 796f 7520 7761 6e74 2074 6f20      ess.you.want.to.
```

```
0x00e0   7365 6e64 200d 0a79 6f75 7220 6d65 7373      send...your.mess
0x00f0   6167 6520 746f 2c20 7468 656e 2065 6e74      age.to,.then.ent
0x0100   6572 2061 206d 6573 7361 6765 2068 6572      er.a.message.her
0x0110   652c 2061 6e64 2068 6974 2053 7461 7274      e,.and.hit.Start
0x0120   2e0d 0a54 6865 2072 6567 6973 7465 7265      ...The.registere
0x0130   6420 7665 7273 696f 6e20 7769 6c6c 2073      d.version.will.s
0x0140   656e 6420 6d65 7373 6167 6573 2074 6f20      end.messages.to.
0x0150   7468 6520 7365 6c65 6374 6564 2049 5020      the.selected.IP.
0x0160   0d0a 7261 6e67 652e 0d0a 00                  ..range....
```

And so, there it is, the message datagram. Now let's dissect it! This is the only time the DCE RPC client will connect to port 135, assuming all replies are received as expected (remember the RPC Pings). The first thing that is noticeable is the packet ID of 10734. As the previous packet ID was 10733, we can deduce that the sending machine is not sending any other traffic out. How does one know that? Well, the IDs are assigned incrementally, and as the increment from the first packet to the second packet sent has only incremented by one, we know the sending host isn't generating any other network traffic. I would suspect that in a registered version that can blast out messages as fast as they'll go, the IDs would jump wildly.

Now, let's get an idea of where our Internet protocol header boundaries are so that we can determine where are DCE RPC header and payload begins. We can see that the packet has a total length of 377 bytes. First, we must subtract 14 bytes for the Ethernet header (not shown in the above dumps), leaving us with 363 bytes. Second, let's remove the IP header. The only issue is that the IP header is a 4-bit field, which means our header can be up to 60 bytes, so we'll have to determine how big it actually is. We determine this by looking at the low order nibble of the first byte ((**5**) **bolded** and underlined). As it is a 32-bit word, we multiply that number by 4 to get the size in bytes, which gives us the number 20. Since 20 bytes is the minimum size of an IP header, we also know that no IP options are set (IP header is colored in red). We subtract 8 now, as the UDP header is always 8 bytes in length (colored in ugly green). Following this, we subtract 80 for the DCE RPC header (in blue), which is a fixed length header. This leaves us with 255 bytes for stub payload (which remains uncolored). As noted earlier, while we only sent 194 bytes worth of user data, the actual stub payload portion will be larger because it contains additional information along with the user's message.

What's in the DCE RPC headers? Below is a portion of the above trace, with only the DCE RPC header intact. There's quite a bit of useful information to glean out of this header, and why shouldn't there be? It's 80 bytes long! This part is likely to get a bit drab, but atleast the hex dump is all rainbow-rific ☺

```
0x0010                                  0400 0800      .....D...Wj.....
0x0020   1000 0000 0000 0000 0000 0000 0000 0000       ................
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0       ......{Z........
0x0040   4fb6 e6fc 58f4 c1aa 68a4 9047 a3a9 4ba6       O...X...h..G..K.
0x0050   bb36 c1f7 0000 0000 0100 0000 0000 0000       .6..............
```

0x0060 0000 ffff ffff ff00 0000 0000 ....

The easiest way to cover all the fields is probably to just list them out and go over them one at a time, highlighting along the way. Research for the following was done at http://www.opengroup.org/onlinepubs/9629399/chap12.htm. It's a fantastic wealth of information on the various details of DCE RPC.

- The first field is the RPC version. In this case we can see the version in use is four (04).
- The second field is the Packet Type, listed here as 00. There are many types of packets, however the kinds we will see are:
    1. **Response** (Indicated by a value of 02)
    2. **Request** (Indicated by a value of 00)
    3. **Ack** (Indicated by a value of 07. This is similar to a TCP ack, where it is just an acknowledgement of reception)

  A complete list of types can be found at http://www.opengroup.org/onlinepubs/9629399/chap12.htm.
- The third field is the first flags field, listed here as 08. This field has a number of options
    1. Reserved (value 01)
    2. Reserved (value 80)
    3. Lastfrag (value 02) – Signifies last fragment
    4. Frag (value 04) – Signifies a single fragment and there are more parts to come
    5. NoFack (value 08) – Signifies the receiving end does not need to send an ack for fragments (basically acknowledgement of fragment reception).
    6. Maybe (value 10) – A client sets this when it doesn't care for a response.
    7. Idempotent (value 20) – Idempotent is defined as multiple iterations being considered as one.
    8. Broadcast (value 40) – Set only by client. A client sets this when it desires the request to be sent to all hosts.
- The fourth field is the second flags field, represented by 00. This field also has numerous meanings
    1. Reserved (value 01)
    2. Reserved (value 04)
    3. Reserved (value 08)
    4. Reserved (value 10)
    5. Reserved (value 20)
    6. Reserved (value 40)
    7. Reserved (value 80)
    8. Cancel_Pending (value 02) – Used to cancel any pending operations

  All reserved bits MUST be set to zero, with the exception of Reserved 01. Reserved 01, like the other Reserved bits in first flags field, are available

for use by other implementations.

- The fifth field is Data Representation (100000). This field informs the receiver of what types of formats the sender adheres. This has three sub fields, they are:
    1. Byte Order (Little Endian is 10)
    2. Character (ASCII is (00)
    3. Floating Point (IEEE is (00)
- The sixth field is high order serial (00). This is set to zero for the initial request. When fragments are sent or resent, this field is incremented. It acts similar to the IP ID. It is important to note even though this is a 16-bit field, the high and low order bits are not immediately adjacent to each other in the header. The high order bit follows after the Data Representation, and the low order after the "Authentication Protocol Number."
- The seventh field is the Object UUID (00000000000000000000000000000000). Objects are identified by this UUID; this field will be zeros if there is no object.
- The eighth field is the Interface UUID (f8917b5a00ffd011a9b200c04fb6e6fc). Again, this is a unique ID. As noted earlier, the Messenger Service registers itself with the Mapper, and this is its unique ID. Below is an example what the Messenger Service looks like from rpcdump:

  (The above HEX translates into the below Messenger UUID of 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc. All UUIDs are sent in network byte order, which means that they are printed backwards in the HEX.)

  ```
  ProtSeq:ncadg_ip_udp
  Endpoint:1026
  NetOpt:
  Annotation:Messenger Service
  IsListening:NOT_PINGED
  StringBinding:ncadg_ip_udp:192.168.0.6[1026]
  UUID:5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
  ComTimeOutValue:RPC_C_BINDING_MIN_TIMEOUT
  VersMajor 1  VersMinor 0
  ```

  We can also notice from *ProtSeq:ncadg_ip_udp* and *Endpoint:1026* the Messenger Service is bound to UDP/1026.
- The ninth field is the Activity UUID (58f4c1aa68a49047a3a94ba6bb36c1f7). This basically performs the same action that a session ID would.
- The tenth field is the Server Boot Time (00000000). This is the time in seconds since January 1st, 1970, that the service has been running. This is not the same thing as when the server itself physically booted up. If the service should crash, the server will have a new Boot Time, which the client will notice. This prevents accidentally running operations multiple times.

- The eleventh field is the Interface Version (01000000). This number defines the version of an Interface a client is requesting. This enables servers to offer multiple versions of an Interface.
- The twelfth field is the Sequence Number (00000000). Sequence numbers are associated with an Activity when they initiate an RPC call, and will remain unchanged until a new call is initiated. Activity UUIDs and Sequence Numbers are used together to become a unique ID that identifies specific RPC calls.
- The thirteenth field is the Operation Number (0000). Distinguishes specific operations. When a call has no object, this field will have a value of zero.
- The fourteenth field is the Interface Hint (ffff). This field is used for server optimization when looking up Interface information, but can additionally be used by an implementation for whatever it needs.
- The fifteenth field is the Activity Hint (ffff). This field is used for server optimizations when looking up Activity information, but can additionally be used by an implementation for whatever it needs.
- The sixteenth field is the Fragment length (ff00). I have also seen this listed as "PDU Body Length." It is the size in bytes of the DCE RPC payload.
- The seventeenth field is the Fragment number (0000). If there are multiple parts to a RPC session, this indicates what fragment this particular part is.
- The eighteenth field is the Authentication protocol (00). This field indicates whether or not authentication is used. 0 implies none, and 1 indicates "OSF DCE Private Key Authentication."
- The nineteenth field is the low order serial (00). This is the other half of the serial number.
- The twentieth field is the Stub payload (Finally! This is the rest of the datagram, which contains the user's message. It is not shown in the above header snippet).

What does this mean in English? The client creates an RPC request that is delivered to port 135. Once arriving at the other side, the RPC server determines that the client is requesting the Messenger Service. The client also sets the "No Fack" option, signifying that the server does not need to acknowledge fragments, should a request's data exceed the maximum PDU body size. The sequence number and operation number are both set to zero. As this is an initial request, the serial is also zero. We also can see the client informing the server that the PDU body is 255 bytes. The body (the text message) is also included after the header. No authentication is present, and the Idempotent flag is not set.

The next datagram is the server's response:

```
17:45:02.894470 0:c0:f0:6d:8:73 0:50:8b:f0:b5:30 0800 142: 192.168.0.6.2697 >
10.11.7.105.1348: [udp sum ok] udp 100 (ttl 127, id 18051, len 128)
0x0000   4500 0080 4683 0000 7f11 22c8 c0a8 0006        E...F.....".....
0x0010   0a0b 0769 0a89 0544 006c ac8b 0400 2004        ...i...D.l......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
```

```
0x0030   0000 0000 7622 3a33 0000 0000 0d00 0080        ....v":3........
0x0040   9c00 0000 61ce be49 6c8b b444 bcb1 7939        ....a..Il..D..y9
0x0050   43b0 b260 0000 0000 0300 0000 0000 0000        C..`............
0x0060   0100 ffff ffff 1400 0000 0000 58f4 c1aa        ..............X...
0x0070   68a4 9047 a3a9 4ba6 bb36 c1f7 b194 2c3e        h..G..K..6....,>
```

Remember earlier when I mentioned the server responding to an unexpected client with conv_who_are_you (or conv_who_are_you2)? Well, this is it! Doesn't look like much off hand. Let's refresh what conv_who_are_you means before we dig into the packet. As stated previously, non-idempotent requests from clients that the server knows nothing about will illicit this response. Upon receiving this call, the client returns its sequence number for the current operation. If the sequence number is higher than the one in the previous request, the server considers the request a duplicate and will discard it.

The server will also send its Server Boot Time (as noted earlier) to the client. Sending the boot time to the client prevents a scenario where the server receives and executes a request, but crashes before being able to send a reply. After the service restarts, the client will try to send the request again which will illicit another conv_who_are_you, with a different boot time. The client will notice the times are different, and tell the server not to execute its request.

So, above we see the request for the current Activity UUID in orange, which in this case translates to aac1f458-a468-4790-a3a9-4ba6bb36c1f7 (notice that these are backwards in the dump). The date is in green, and translates to Jan 20th 2003 16:30:41. Note that the request was generated from the server on a high port, and sent to the client's originating port of 1348 from the initial DCE RPC request.

The server sets a new Activity UUID (session ID), as this is a new request. The Activity UUID for this conv_who_are_you conversation is 49bece61-8b6c-44b4-bcb1-793943b0b260 (marked in blue). This Activity UUID will also need to be returned to the server in the reply from the client.

The client's response looks like:

```
17:45:02.894581 0:50:8b:f0:b5:30 0:c0:f0:6d:8:73 0800 146: 10.11.7.105.1348 >
192.168.0.6.2697: [udp sum ok] udp 104 (ttl 128, id 10735, len 132)
0x0000   4500 0084 29ef 0000 8011 3e58 0a0b 0769        E...).....>X...i
0x0010   c0a8 0006 0544 0a89 0070 019a 0402 0804        .....D...p......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 7622 3a33 0000 0000 0d00 0080        ....v":3........
0x0040   9c00 0000 61ce be49 6c8b b444 bcb1 7939        ....a..Il..D..y9
0x0050   43b0 b260 0000 0000 0300 0000 0000 0000        C..`............
0x0060   0100 ffff ffff 1800 0000 0000 0000 0000        ................
0x0070   3bb8 171b 4b74 4948 a2e2 1102 d760 a9ee        ;...KtIH.....`..
0x0080   0000 0000                                      ....
```

Like the previous packet, this is just as simple. As seen in blue, the client returns

its unique client address space (CAS UUID), which translates to 1b17b83b-744b-4849-a2e2-1102d760a9ee. The returned sequence number of zero is colored in brown. Incidentally, a success (zero) is returned. Since the sequence number is not higher than it should be, this call is not a duplicate and can continue.

And the server's response:

```
17:45:02.899401 0:c0:f0:6d:8:73 0:50:8b:f0:b5:30 0800 122: 192.168.0.6.2697 >
10.11.7.105.1348: [udp sum ok] udp 80 (ttl 127, id 18052, len 108)
0x0000   4500 006c 4684 0000 7f11 22db c0a8 0006        E..lF.....".....
0x0010   0a0b 0769 0a89 0544 0058 1d8c 0407 2000        ...i...D.X......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 7622 3a33 0000 0000 0d00 0080        ....v":3........
0x0040   9c00 0000 61ce be49 6c8b b444 bcb1 7939        ....a..Il..D..y9
0x0050   43b0 b260 0000 0000 0300 0000 0000 0000        C..`............
0x0060   0100 ffff ffff 0000 0100 0001                  ..............
```

Basically, this is just a response to the client in regards to the conv_who_are_you conversation they've had. We can determine this is an "Ack" packet type by the HEX 07, **bolded** above. I can also relate the above conv_who_are_you datagrams with this one, as they all have the same Activity UUID, as colored above in pink. This translates to 49bece61-8b6c-44b4-bcb1-793943b0b260. This PDU has no payload. Also notice that the server's serial number is now incremented.

The next packet is a response to the original request, as sent by the client. The astute reader will also notice that the server's (target) source port has changed to 1026. This is the same port listed in the above rpcdump from the section "The eighth field," which is where the Messenger Service is listening.

```
17:45:02.902967 0:c0:f0:6d:8:73 0:50:8b:f0:b5:30 0800 126: 192.168.0.6.1026 >
10.11.7.105.1348: [udp sum ok] udp 84 (ttl 127, id 18054, len 112)
0x0000   4500 0070 4686 0000 7f11 22d5 c0a8 0006        E..pF.....".....
0x0010   0a0b 0769 0402 0544 005c 7f78 0402 0800        ...i...D.\.x....
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 d221 8e42 cfcd 66aa 50c1 0beb        .....!.B..f.P...
0x0040   4c12 6434 58f4 c1aa 68a4 9047 a3a9 4ba6        L.d4X...h..G..K.
0x0050   bb36 c1f7 b194 2c3e 1357 5888 0000 0000        .6....,>.WX.....
0x0060   a3c7 ffff 7300 0400 0000 0000 0000 0000        ....s...........
```

We can determine this, again, by the fact that this response (listed in **bold**) and the original request refer to the same Activity UUID, which is colored in Aqua. Once more for the curious, this is translated into aac1f458-a468-4790-a3a9-4ba6bb36c1f7 (we're back to the original Activity UUID). It is at this point the client's text message is actually delivered.

And last, but not least, the final packet! Drum roll? Anyone? Anyone?

```
17:45:02.903351 0:50:8b:f0:b5:30 0:c0:f0:6d:8:73 0800 122: 10.11.7.105.1348 >
192.168.0.6.1026: [udp sum ok] udp 80 (ttl 128, id 10736, len 108)
```

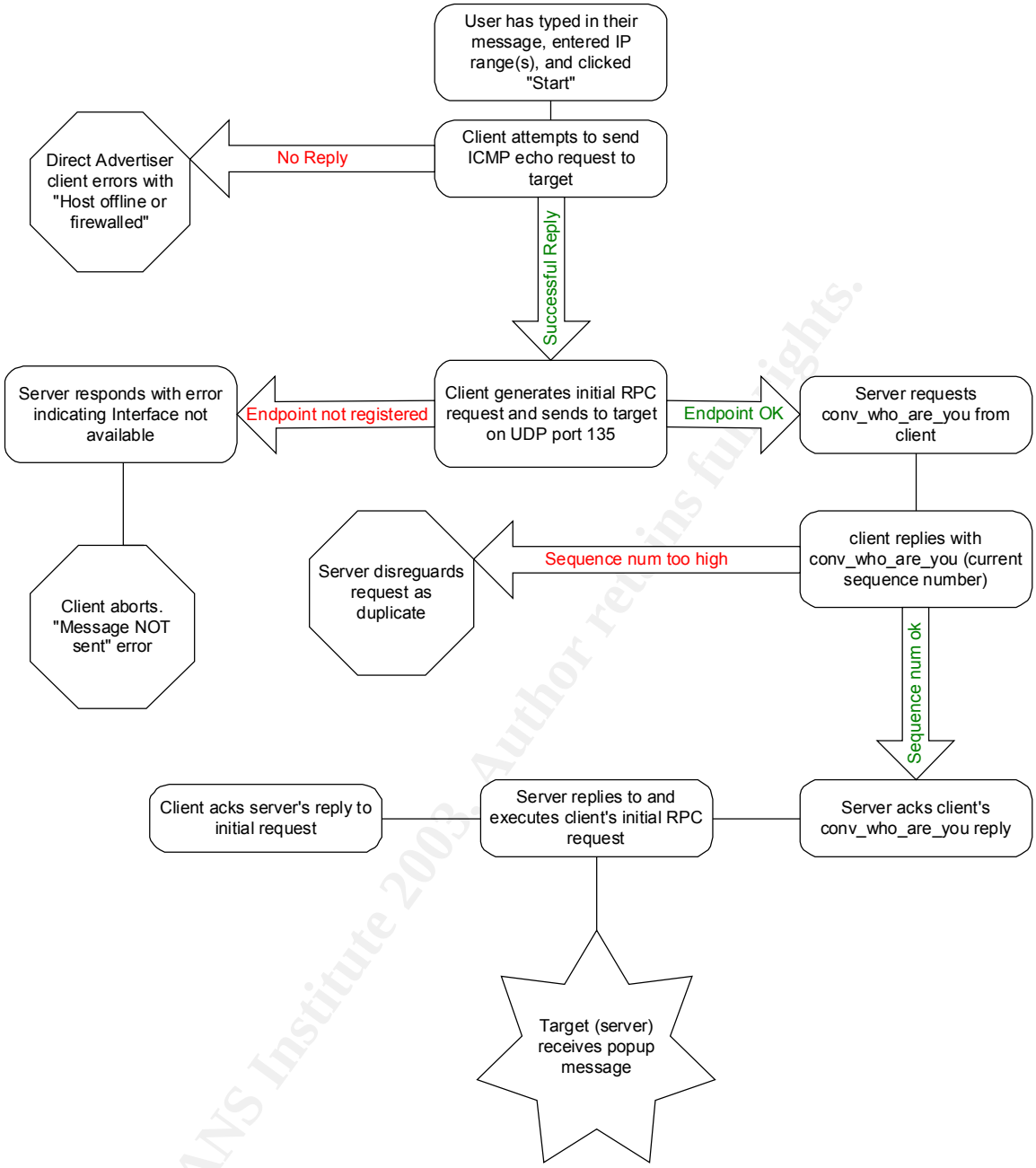
```
0x0000   4500 006c 29f0 0000 8011 3e6f 0a0b 0769        E..l).....>o...i
0x0010   c0a8 0006 0544 0402 0058 f4a0 0407 0000        .....D...X......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0        ......{Z........
0x0040   4fb6 e6fc 58f4 c1aa 68a4 9047 a3a9 4ba6        O...X...h..G..K.
0x0050   bb36 c1f7 0000 0000 0100 0000 0000 0000        .6..............
0x0060   0000 ffff 7300 0000 0100 0001                  ....s.......
```

It's an ack type! There's really nothing spectacular about this and it doesn't contain anything we didn't already cover. The purpose of this datagram is simply to tell the server that the client has received the previous response datagram. Though, it's worth noting that the client has now incremented its serial number. In my tests it was not necessary for the server to receive this ack. The popup message was delivered just fine having firewalled this packet.

That's the entire path the message takes to get from the sender to the receiver. Pretty simple, eh? Only eight packets total were sent or received, and in less than one second. In the end, the target machine will now have a popup message that looks like:

Ok, but what does it look like conceptually? Consider the following diagram:

NOTE: this diagram does not take into consideration firewalls, administrative actions, route failures, etc that will otherwise block the sequence of events.



Some readers may have noticed at this point that the freshly described RPC process does not exactly flow like the RPC protocol was discussed – specifically, the part that is missing here is where the client asks the Mapper on what port a specific dynamically allocated service is listening, and then connects directly to that port. In actuality, the above scenario also tells the client what port the service is on, and the client (stub) caches this information. Subsequent connections to the RPC server (as in if the client decided to send multiple message pop-ups) from the client would reuse this cached information, such as the same Activity UUID, **and** send the new request directly to whatever port the Messenger Service is bound to (1026 in this case). The Sequence Number will also

increment by one. The Activity UUID does have a limited lifetime however, and the cache will expire (seems to be about 40 seconds in my testing). After the expire time, any RPC requests to the server will repeat all the discovery steps over again, starting with port 135.

How to use the exploit

We've come a long way since we first started, but there are still many more areas left to cover! The next stop in the journey is how one would use the products covered in this paper. I will primarily focus on the specific aspects of using the programs, and less on general, overall discussions. I believe the overall discussion was explained in exhaustive detail in the "*Description of Variants*" section. As Direct Advertiser's product is the primary focus of this paper, I'll start with that.

Direct Advertiser:
As stated earlier, Direct Advertiser only offers a demo version of the product unless I shell out some major cash for a licensed copy. For this reason, some aspects of this program (really, all of the programs mentioned in this paper) are crippled, so how it works will be speculation. Though, the interfaces are intuitive, so I'd bet my speculation to be correct.

The demo version requires that the user enter "demo" at the Login prompt. After doing so, the user is presented with a notification that this is a demo product, and can only send messages to a single IP address. The user must click "OK" here in order to continue. The login prompt looks like this:

At this point, the user is now in the primary GUI. There are no other menus or screens, this is all there is to the program. There was a screen shot of this earlier in the "*Descriptions of Variants*" section. Everything we can do with the demo is on the right side. We'll focus our attention there, then move to the left and discuss what the user could do if this were a registered product.

The first thing the user should do is set the "Connection speed" to something reasonable. For me, I set it to "T1 or higher," as that is the fastest setting there is, and I'm on a 100Mbit LAN. Other settings are Cable/DSL, 56K and 28K dialup. In reality, this setting is of no use to anyone with the demo (I can only send to one IP address at a time). Its purpose is to have the program rate limit itself while

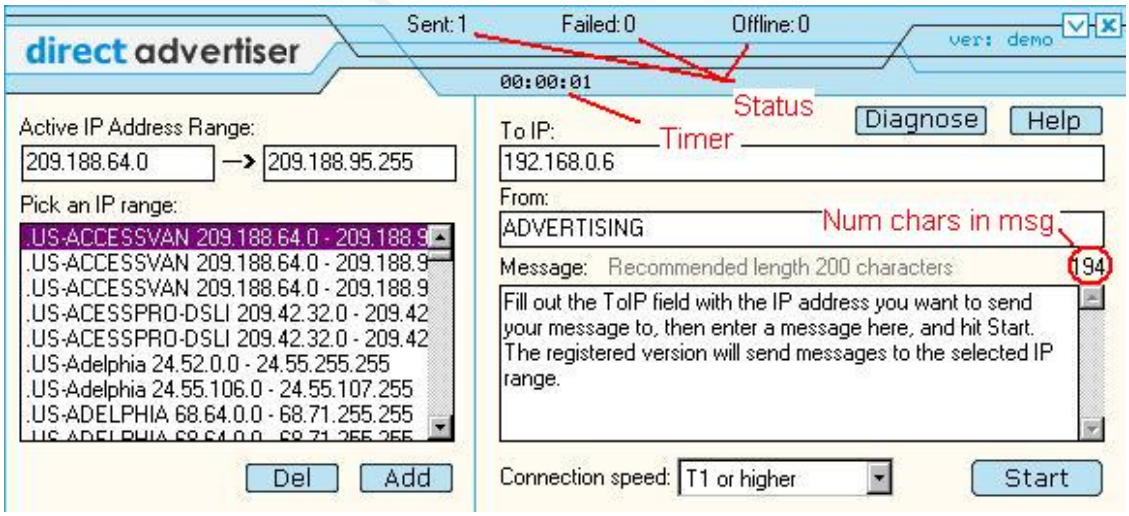
sending out messages to the Internet as fast as it can without causing the network to fall over.

Moving up a step, we're at the "Message" box. This is where the user types his or her exciting messages for everyone to read! I've left it at the default for testing. The recommended maximum message size is 200 characters, and really, anything over that gets to be a bit ugly anyway. I had setup a machine that was open to the Internet that I was watching for pop-ups, and really, some people write a whole book into those messages.

The next hop up is the "From:" box, which allows a user to specify a sender name. The default is ADVERTISING, but it could be anything the sender wishes. Though, it doesn't seem to like input if it's not word chars (alpha-numeric, underscore, etc). It aborts with "Your message was NOT sent due to unknown error!" Using no name at all will cause the pop-up to be sent using the actual name of the machine.

Finally, the last configurable part for the demo is the "To IP" box. Enter an IP address here. There doesn't seem to be any error checking here. Invalid IPs, hostnames, other characters, etc just seem to pass by without an error.

At this point, everything is all setup and ready to click "Start." Clicking will cause the message to be sent to the target IP, and a dialog box tells us that our message has been successfully sent (as in, the host is up, the endpoint is registered, and all acks and replies were received as expected). The "Sent" field will also increment by one. Should there be a failure or an offline machine, those status counters will also increment. I've included a screen shot so the reader doesn't have to scroll up to see this. I've also made a few notes on the graphic.



Another possibly useful bit of information provided by the application is a timer feature in the top center of the screen. This does a count-up second-by-second of how long it takes a message to be delivered. That's rather nice, I think.

Moving over the left side of the screen are the tid-bits that could be used if this were a registered product. This is where the product becomes really useful. This side is really only dedicated to one thing – IP ranges. The user can select from the MANY IP ranges that come default with the app. If the user doesn't find a range they like, it is possible to create a new one. Under the "Active IP Address Range" section about the IP ranges, a user can enter a starting IP in the first box, and a finishing IP in the second. The user can now click "Add," select the newly created range, and then configure as before. This is definitely easier than doing this one IP at a time. Notice the end user can also use "Del" to delete ranges that aren't necessary.

There are only two things left to do here. One is the "Help," which is just a typical Windows help box where one would point and click at topics. The other is what I've mentioned before and thought of it as minimal in its usefulness. I'll mention the "Diagnose" button again, as at this point I have a better understanding of the product. Clicking on "Diagnose" presents a dialog box that says this will diagnose the computer. Going ahead with the diagnosis, it just tells the user the stats of their machine (the output was listed earlier under "*Description of Variants*"). The two items to note which can't be determined from looking at the output are "what is it pinging, and how is it determining my upload speed?" As noted earlier, the ping attempts to talk to **w1.rc.vip.scd.yahoo.com**, **altavista.com**, and **www.google.com** - Nothing special there. What had me amused for a short while was how it checks the upload speed. It establishes an http connection with www.bestadv.net (an interesting site in and of itself. Bestadv is a website hit booster that almost sounds like a pyramid scheme), makes a POST to /test.cfm, and sends 51200 bytes to the server. I've included dumps as "Appendix E" for those that are interested.

That's all there is to do with this product, we've covered everything! It, like the others, is a focused product and doesn't do anything else fancy except what it was designed to do.

Broadcast Advertiser (v3.0.1):
After starting the application, the user is in the one, and only, screen. There are really only three areas of configuration here: the "IP range," the "From" and the "Message." There are two fields related to the IP range we want, and they are "Start IP Addr" and "Finish IP Addr." The IPs are entered as dotted-quads (the period notation, like 1.2.3.4). The IPs must be valid, as invalid IPs will refused to be added. One caveat to adding an IP is that the cursor automatically advances after 3 digits per octet, or when the user types a period. Those of us used to entering a period to advance the cursor to the next octet will be hung up in the interface and left with a silly-looking IP that is missing an octet or two. In the demo (maybe this is like this with the registered version?) the only way I'm able to get the newly created IP range into the box of ranges that the program uses is to click the "Only one IP range" box. Doing so adds the IP range as well as

automatically selects it for use. The product does have a “Help” button, but all it does is redirect to their website which displays their phone number and support email address.

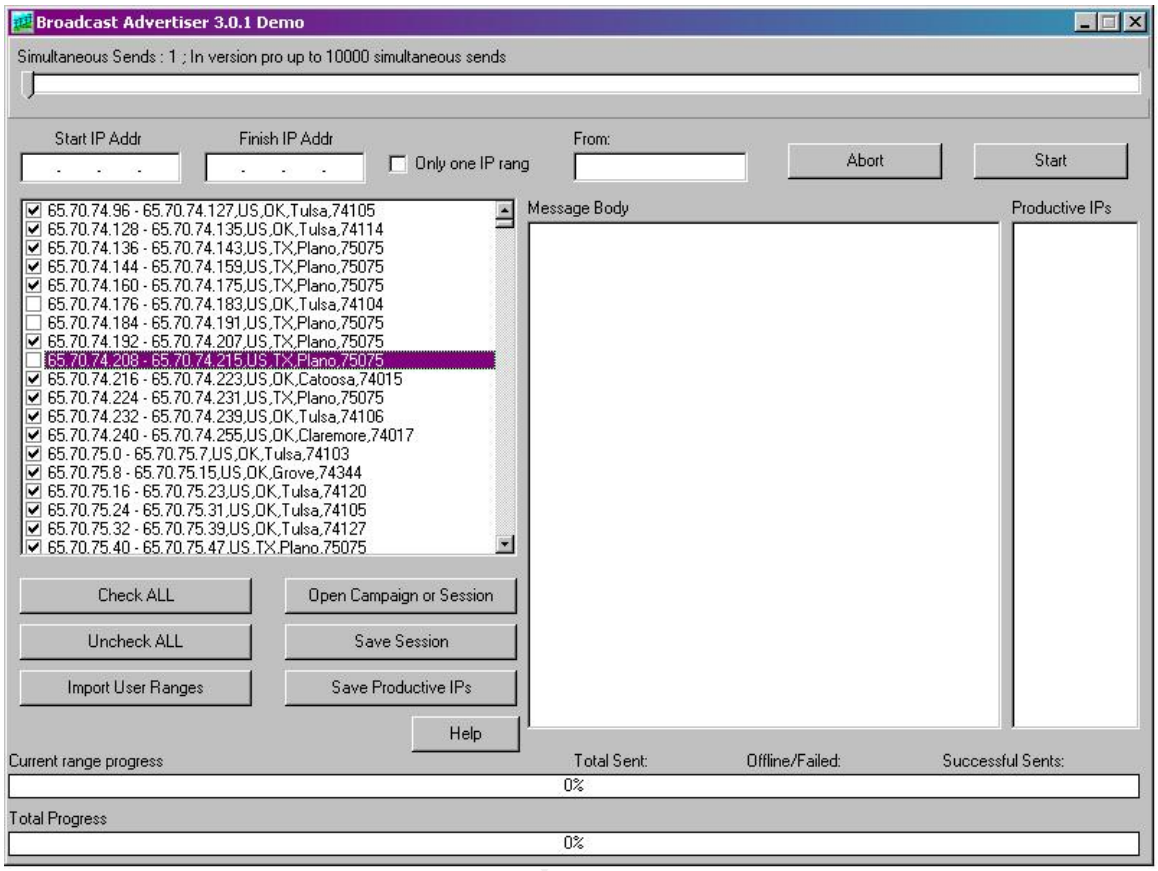
Next up is the “From” field. It’s really self-explanatory. Symbols do not work, and so word characters should be used here, just as with Direct Advertiser.

Lastly, the Message can be whatever one feels they want to send. Keeping to a smaller size makes the message look less ugly. At this point, “Start” can be clicked, and the Message will be sent to the range created (one IP at a time, since this isn’t licensed). The progress bar for the range will also slide across the screen as messages are sent, and the stats for “Total Sent,” “Offline/Failed,” and “Successful Sents” increase as appropriate.

The “Total Progress” bar never seemed to do anything for me.

The user should also be able to slide the “simultaneous sends” bar across the screen to increase the amount of sends done at one time. This isn’t something I can test since this is a demo.

We can also save our session with the “Save Session” button, should we decide later to use it again. However, I’ve noticed that when opening the session, all that is saved is the IP range. The “From” and Message are blank. Though, the most powerful part of this product is how granular it can be with being able to send to IPs as specific as a zip code. In order to make use of these newly added features, Broadcast Advertiser’s campaign database must be downloaded and loaded into the product (via the “Open Campaign” button). Otherwise, the process of sending is the same. All the campaign does is populate the IP range box, just like creating a new IP range does.
The databases can be found at http://www.broadcastadvertiser.com/demodb.html. I’ve included a screenshot of the application with the zip code ranges and also so the reader doesn’t need to scroll way up to view the application while reading how to use it.



Sly Sender (4.0)
Using Sly Sender is about the same as using Broadcast Advertiser, but with a few bells and whistles changed. Again, as before, the first screen presented after starting the application is the one and only. Per the usual, there are three critical areas to fill out. The first is the IP range, which will normalize bunk IPs, like converting 666 into 255 and ignoring anything other than numbers. After adding the ip, clicking “Add” will include the IP range in the box of selected ranges. Next to customize are our ol’ favorites, the “From” field, and the Message field. The last thing for the user to do is to select the network speed. The available speeds in the drop-down menu range from 56K to T3.

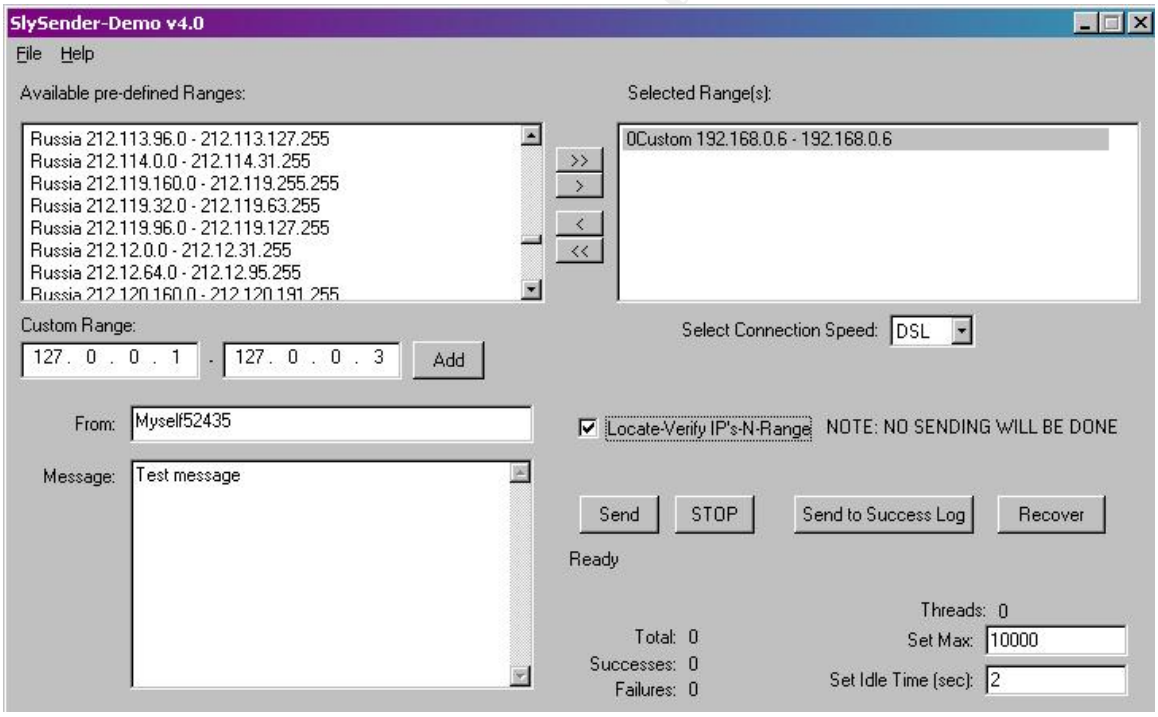
If this were a registered product, the user could also select IP ranges from the list that comes with the product (which appear to cover a large range of address from Albania to Zimbabwe). These ranges may be selected using the >, >>, <, and << buttons, which will move IP ranges to or from the selected ranges box. Finally to send the message, the user should click “Send.” Status bar items will increment accordingly for Total, Successes, or Failures.

I hear what the reader is thinking; these products are all almost the same! Yeah, pretty much. Sly Sender does have a couple of interesting features. Mostly notably is the Locate and Verify feature. The entire configuration is done basically the same, however, with this feature selected, no messages are sent.

The application simply verifies whether or not an IP is able to accept a message (which goes through all the motions of sending a message, just doesn't tack on a message payload). Simply clicking the checkbox labeled "Locate-Verify IP's-N-Range" will turn on this feature. This feature would likely be a good combo with the "Send to Success Log" button, which will store successful IPs to a file. I can't test the Success Log button myself; it is disabled in the demo.

The user can also further fine-tune how system resources are used, in addition to setting the connection speed, by setting the max threads and idle time options. The faster the connection, the more threads one can use and setting the idle time higher reduces the stress on the machine. Remember from earlier that the maximum number of threads is 99,999. These options again have no bearing for the demo product, as it can only send to one IP address anyway.

Lastly, in the event a of a system crash, it is still possible for the user to recover from where the application left off. This functionality is enabled with the "Recover" button, which will recover the last ~40k IPs that were previously in memory. Again, this is disabled in the demo. Below is a screenshot for reference.



Are there any other programs?

What would happen if there weren't any programs like the above? How could these messages be sent? That's the real beauty of the exposure/vulnerability – this functionality is built right into Windows itself via the "net send" command (which can likely be found in the system32 directory). Net send has just one job, and that is to take a user's message and direct it to the appropriate user(s) or computer(s) across a network. Even more, Direct Advertiser and Broadcast

Advertiser both claim on their websites they use “net send.” I don’t believe that these programs actually make use of it directly. I will make note of these during the discussion of “net send.” Let’s start out with some banter about how to use “net send.”

There are several different ways to use “net send,” each usage variant affects where a message is delivered. If the user were a part of a domain and wanted to send a message to everyone else associated with the local domain, “net send” would be invoked as:

net send * “a fancy message”

A message has now been delivered to everyone. This technique also has an interesting feature in that if the sending and receiving machines have the same multiple protocols bound, the receiving user will receive duplicate messages - one for each protocol in use.

Replacing the asterisk with the name of a Windows domain allows the user to send to domains outside the local domain.

net send /domain:<DOMAIN NAME> “another message here”

<DOMAIN NAME> is the name of any existing domain. This command should be used without the < > characters. This accomplishes the same results as above, except a domain has been specified.

Getting grittier, “net send” allows sending of message to all users who have established a session with the server.

net send /users “hello all you users!”

This command, like the above, will issue duplicate messages to users if they have multiple establishes sessions.

Grittier still, and the most useful for anyone trying to emulate how the above products works is:

net send <USER NAME or IP> “I’m a carrot!”

Using this command will send the message “I’m a carrot!” to the specific IP or username listed. Again, < > are not part of the command.

NOTE: This is my first hint that the programs discussed in this paper must not use “net send” directly from the command line. Within the syntax of “net send” there is nowhere to specify a “From” name. “Net send” uses the sending computer’s real NetBIOS name.

So how would one use “net send” to send a message? Let’s consult the test network. Using what we’ve learned above, I constructed the following command:

```
C:\WINNT\system32>net send 192.168.0.6 "test message"
```

Which returned the response (as a Messenger dialog box popped up on the target host):

```
The message was successfully sent to 192.168.0.6.
```

Now, as the reader will notice, it would appear that our message was delivered via NBT (NetBIOS over TCP/IP) protocol. This is unlike the Messenger Spam we’ve been discussing, as that uses port 135, MS RPC. Below I use tcpdump again to display the packet dump of the conversation. I’ve expanded the message payload packet so the reader can see the message text.

```
18:40:40.992682 10.11.7.105.137 > 192.168.0.6.137: NBT UDP PACKET(137): QUERY;
REQUEST; UNICAST
18:40:40.998417 192.168.0.6.137 > 10.11.7.105.137: NBT UDP PACKET(137): QUERY;
POSITIVE; RESPONSE; UNICAST
18:40:40.998488 10.11.7.105.1207 > 192.168.0.6.139: S 1778340959:1778340959(0) win 64240
<mss 1460,nop,nop,sackOK> (DF)
18:40:41.002353 192.168.0.6.139 > 10.11.7.105.1207: S 1162742068:1162742068(0) ack
1778340960 win 17520 <mss 1460,nop,nop,sackOK> (DF)
18:40:41.002369 10.11.7.105.1207 > 192.168.0.6.139: . ack 1 win 64240 (DF)
18:40:41.002394 10.11.7.105.1207 > 192.168.0.6.139: P 1:73(72) ack 1 win 64240NBT Packet
(DF)
18:40:41.007358 192.168.0.6.139 > 10.11.7.105.1207: P 1:5(4) ack 73 win 17448NBT Packet
(DF)

[Expanded packet begin]

18:40:41.007463 10.11.7.105.1207 > 192.168.0.6.139: P [tcp sum ok] 73:150(77) ack 5 win
64236
>>> NBT Packet
NBT Session Packet
Flags=0x0
Length=73 (0x49)

SMB PACKET: SMBsends (REQUEST)
SMB Command   =  0xD0
Error class   =  0x0
Error code    =  0 (0x0)
Flags1        =  0x0
Flags2        =  0x0
Tree ID       =  0 (0x0)
Proc ID       =  0 (0x0)
UID           =  0 (0x0)
MID           =  0 (0x0)
Word Count    =  0 (0x0)
smbbuf[]=
Source=PENELOPE
Dest=192.168.0.6
```

Data: (15 bytes)
[000] 01 0C 00 74 65 73 74 20 6D 65 73 73 61 67 65   ...test message

[Expaned packet end]

18:40:41.008354 192.168.0.6.139 > 10.11.7.105.1207: . ack 73 win 17448 (DF)
18:40:41.013710 192.168.0.6.139 > 10.11.7.105.1207: P 5:44(39) ack 150 win 17371NBT Packet (DF)
18:40:41.013820 10.11.7.105.1207 > 192.168.0.6.139: F 150:150(0) ack 44 win 64197 (DF)
18:40:41.018082 192.168.0.6.139 > 10.11.7.105.1207: F 44:44(0) ack 151 win 17371 (DF)
18:40:41.018096 10.11.7.105.1207 > 192.168.0.6.139: . ack 45 win 64197 (DF)

So what gives, this isn't RPC! Remember earlier when I mentioned that this type of Spam could be delivered over NetBIOS? Well, here it is!

NOTE: This is my second hint that the programs discussed don't use "net send" directly, and rather probably something along the lines of functions or DLLs (libraries) associated with it. Alas, there is no source code available, so I can only speculate.

The reader should also remember from earlier that "net send" also as the capability to use RPC as well. "Net send" will do this if the NetBIOS ports are not available for use. To wit, I've added the following firewall rules to the Linux Iptables firewall:

```
iptables -A FORWARD -o ipsec0 -p udp --dport 137 -j REJECT
iptables -A FORWARD -o ipsec0 -p tcp --dport 139 -j REJECT
iptables -A FORWARD -o ipsec0 -i eth1 -j ACCEPT
```

Let's go through the firewall ruleset to get a better understanding of what packets are being blocked.

iptables
The name of the firewall binary

-A
Appends a rule

FORWARD
Specifies which chain to apply the rule to. Chains can be INPUT, OUTPUT, or FORWARD and pertain to how a packet traverses an interface. I've used FORWARD here because the firewall/gateway is routing packets through its interface and not necessarily IN or OUT of one of its own interfaces.

-o ipsec0
Determines whether the rule is applied going out (-o) or in (-i) on the specified interface (ipsec0)

-p udp
Which protocol to use – UDP is listed here.

--dport 137
What destination port should the rule watch for?

-j REJECT
This determines what happens to the packet when it matches a rule. In this case, it's REJECT. Other options are ACCEPT and DROP.

In a nutshell, what these rules say are "REJECT (tell the remote host 'you can't do that') packets going out the Ipsec0 interface to any destination port UDP/137 or TCP/139 and allow everything else."

Trying our earlier command again with the firewall rules inplace yields the following dump:

19:17:52.051793 10.11.7.105.137 > 192.168.0.6.137: NBT UDP PACKET(137): QUERY; REQUEST; UNICAST
19:17:52.051966 10.11.7.77 > 10.11.7.105: icmp: 192.168.0.6 udp port 137 unreachable [tos 0xc0]

[more of the same snipped]

19:18:07.045993 10.11.7.105.137 > 192.168.0.6.137: NBT UDP PACKET(137): QUERY; REQUEST; UNICAST
19:18:07.046119 10.11.7.77 > 10.11.7.105: icmp: 192.168.0.6 udp port 137 unreachable [tos 0xc0]
19:18:08.545992 10.11.7.105.137 > 192.168.0.6.137: NBT UDP PACKET(137): QUERY; REQUEST; UNICAST
19:18:08.546117 10.11.7.77 > 10.11.7.105: icmp: 192.168.0.6 udp port 137 unreachable [tos 0xc0]

[Here we start the familiar RPC request]

19:18:10.046952 10.11.7.105.1214 > 192.168.0.6.**135**: udp 153
19:18:10.053261 192.168.0.6.1062 > 10.11.7.105.1214: udp 100
19:18:10.053370 10.11.7.105.1214 > 192.168.0.6.1062: udp 104
19:18:10.058115 192.168.0.6.1062 > 10.11.7.105.1214: udp 80
19:18:10.060594 192.168.0.6.**1026** > 10.11.7.105.1214: udp 84
19:18:10.060850 10.11.7.105.1214 > 192.168.0.6.**1026**: udp 80

So while it takes a bit longer to send the message (about 18 seconds), "net send" can deliver the message over RPC if NetBIOS is not available. Tim Mullen also noticed this type of behavior in his column, which can be found at http://online.securityfocus.com/columnists/117. "Net send" could easily be automated with some PERL code; something similar to the pseudo code below would suffice:

```
while("$startingip" <= "$endingip") {
  system("net send $startingip $message")
  $startingip++;
}
```

Now, what about the spoofing of computer names? I went looking around for something that was similar to "net send" and came up with

http://www.exploitresearch.org/faqs/netspoof2.zip. It works the same way as "net send," only it allows spoofing the "From." A typical command line might look like:

Netspoofmsg 192.168.0.6 "From name" "message text."

It would even appear that spoofing the "From" is very easy – all that is required is the use of the NetMessageBufferSend() function. There will be more on this later in the "*Source code / Pseudo code*" section.

NOTE: this is my third, and final hint that Direct Advertiser does not use "net send" directly. Using "strings" (a program on UNIX that shows printable characters within binaries) I come up with the following string:

[…]
Invalid SendFlags
Invalid internal SendFlags
NetMessageBufferSend
OnAboutToSend4fG
OnSendBegin
[…]

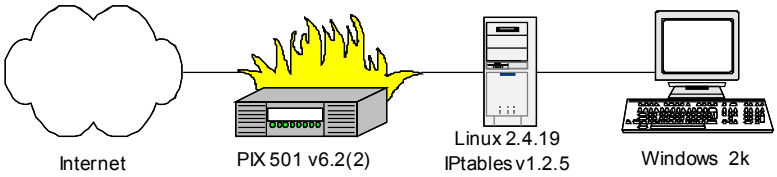
I checked Broadcast Advertiser and Sly Sender as well – they both display similar results.

Signature of the attack

During this research, I've read many articles on Messenger Spam being untraceable, and I have to disagree with that. While it is true that the protocol in use is UDP, which is easily spoofable, I do not believe this would be an easy spoof. I believe this because the client must return the Activity UUID generated by the server during the conv_who_are_you request. The UUID is a 128-bit field, which would be hard to guess. From a network perspective, as the datagram carrying the Spam moves from its source to its destination, it must interact with various network devices. Any device it goes though could record the packet.

Anyway, without further delay, let's start out with what rules must be added to the firewalls, and then discuss the rules, and the meaning of their messages. This will probably be boring, but it is a necessity in order to understand what the firewalls are doing and what the logs mean.

There are many areas where Messenger Spam leaves muddy footprints. As the first layer of defense is a firewall, I'll start here. Consider the following network setup for this discussion:



As a side note, connections coming into the PIX (pixfirewall) are statically NATed through the Linux firewall/gateway (ootganootga) to the Windows 2k box (10.11.7.105).

In order to allow Messenger Spam through the PIX, a few firewall rules must be added:

```
Access-list intout permit tcp any any eq 135
Access-list intout permit udp any any eq 135
Access-group intout in interface outside
```

The first two lines tell the firewall to allow any inbound connections from anywhere, using either tcp or udp, to port 135. TCP really isn't required here, but I added it anyway. Who knows what might show up on my doorstep? Now I must apply the rule group to an interface, which is what the third line does. Let's break it down specifically:

access-list
Command name

intout
Rule-group name

permit
Action to take on match

udp
Protocol to match

any
Source host

any
Destination host (I have a dynamic IP. Port 135 is sent to inside:10.11.7.105)

eq 135
Destination port (**eq**uals)

And finally:

access-group
Command name

intout
Rule group name

in
direction

interface outside
apply to outside interface

Next to setup is the Linux firewall (This will be just as drab. Lucky you!) This is the basic setup of the firewall where this Messenger Spam is concerned.

```
iptables -N forwin_eth1-outside
```

This command creates a rule group called “forwin_eth1-outside.” The following firewall rules will be attached to this rule group.

```
iptables -A forwin_eth1-outside -j LOG
```

Log everything that goes through here (Logging is added here so packets will get logged before any other matching).

```
iptables -A forwin_eth1-outside -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Any connections that were previously allowed, allow the return traffic too. This is also known as stateful.

```
iptables -A forwin_eth1-outside -s 0/0 -d 10.11.7.105 -p tcp --dport 135 -j ACCEPT
```

Append to the forwin_eth1-outside chain a rule that says “allow any source (0/0) going to destination 10.11.7.105 (this destination has been NATed to my local address from my internet (DSL) IP by the PIX at this point) using protocol TCP and going to port 135.

```
iptables -A forwin_eth1-outside -s 0/0 -d 10.11.7.105 -p udp --dport 135 -j ACCEPT
```

Same as above, only use UDP instead of TCP.

```
[… other rules here …]
```

```
iptables -A FORWARD -i eth1 -j forwin_eth1-outside
```

Apply the above rule group to the FORWARD chain.

Now that we know how to add logging rules for our firewalls, let’s discuss what their output is like. If we don’t understand the output, we can’t do anything to protect ourselves, can we? All of my firewall messages are centrally logged via

syslog, let's first examine what a typical syslog message looks like. The format of syslog is generally *Date/Time Hostname Message.* The process that sent the message via syslog often prefixes the message. An example:

Feb 17 06:28:20 ootganootga syslogd 1.4.1#10: restart (remote reception).

PIX logs also follow the format of
Date/Time Code Message. An example:

Feb 17 2003 04:33:14: %PIX-6-302015: Built inbound UDP connection 133471 for outside:xx.xxx.82.132/3475 (xx.xxx.82.132/3475) to inside:10.11.7.105/135 (xx.xx.218.241/135)

Now let's break down the PIX message:

*%PIX-**6-302015***
This has two parts. The first part, 6, is the message severity, which can range from 1 (alert) to 7 (debug). Six is "informational."

The second part, 302015, is a specific number given to each unique message.

*Built **inbound** UDP connection*
This is the really the gist of the message. "Inbound" may also be "outbound."

133471
This number represents the ID associated with this connection. These are incremented for each connection.

for outside:xx.xxx.82.132/3475
Reveals whom this connection was built for. This particular event was generated on the interface "outside" by xx.xxx.82.132 (a host out on the Internet) with source port 3475.

(xx.xxx.82.132/3475)
If this was a mapped address or port, this would show the mapped address. In this case, they are the same.

to inside:10.11.7.105/135 (xx.xx.218.241/135)
This part shows us the destination interface is "inside." The destination address is 10.11.7.105 at port 135. The real address this packet was destined for was xx.xx.218.241 (my DSL IP) at port 135.

Lastly, but not least, here is an example Linux Iptables log message and its definition.

IN=eth1 OUT=eth0 SRC=xx.xxx.82.132 DST=10.11.7.105 LEN=173 TOS=0x00 PREC=0x00 TTL=110 ID=997 PROTO=UDP SPT=3475 DPT=135 LEN=153

IN=eth1
Which direction is the packet headed, and on what interface. Here we have a packet headed IN on eth1. For the curious, this is the interface connected to the PIX. This can also be blank if the local machine created the matched packet.

OUT=eth0
This part is essentially the same as above, just that it describes the other direction. The packet came IN on eth1, and it's going OUT on eth0. This interface connects to the internal LAN. This can also be blank if the packet was sent to the local machine.

SRC=xx.xxx.82.132
This is the source address of the matched packet.

DST=10.11.7.105
This is the destination of the packet

LEN=173
Total length of the entire packet is 173 bytes.

TOS=0x00
Type of Service. Setting different bits in this field effect how the packet is prioritized, such as "minimize delay" or "maximize throughput." However, this field can also be used for ECN (explicit congestion notification). ECN is used for informing the remote host that data is being sent too fast, and it should back off.

PREC=0x00
This is the Type of Service precedence field.

TTL=110
This field represents the Time to Live value, or "how long may a packet ride the network without being discarded." The maximum is 255 hops.

ID=997
Look, another ID! This is the IP (Internet Protocol) ID, which is used in reassembling any IP packets that have been fragmented.

PROTO=UDP
The protocol is UDP.

SPT=3475
The source port is 3475.

DPT=135
The destination port is 135.

LEN=153
Total length of the IP payload is 153 bytes.

Whoohoo!! We're finally finished discussing firewall commands and logs! Now we can actually look at the logs generated from an RPC connection via the Internet. Remember that these logs went through syslog, so each line is prefaced with syslog information, as discussed earlier.

Feb 16 22:33:14 pixfirewall Feb 17 2003 04:33:14: %PIX-6-302015: Built inbound UDP connection 133471 for outside:xx.xxx.82.132/3475 (xx.xxx.82.132/3475) to inside:10.11.7.105/135 (xx.xx.218.241/135)

Feb 16 22:33:14 ootganootga kernel: IN=eth1 OUT=eth0 SRC=xx.xxx.82.132 DST=10.11.7.105 LEN=173 TOS=0x00 PREC=0x00 TTL=110 ID=997 PROTO=UDP SPT=3475 DPT=135 LEN=153

These first two log entries are from the initial DCE RPC connection to UDP port 135. This packet has the entire payload, and initiates the conv_who_are_you from the server.

Feb 16 22:33:14 pixfirewall Feb 17 2003 04:33:14: %PIX-6-302015: Built outbound UDP connection 133472 for outside:xx.xxx.82.132/3475 (xx.xxx.82.132/3475) to inside:10.11.7.105/1035 (xx.xx.218.241/34674)

Feb 16 22:33:14 ootganootga kernel: IN=eth0 OUT=eth1 SRC=10.11.7.105 DST=xx.xxx.82.132 LEN=128 TOS=0x00 PREC=0x00 TTL=127 ID=43433 PROTO=UDP SPT=1035 DPT=3475 LEN=108

Now we see the server (intended target) send out the conv_who_are_you request.

Feb 16 22:33:14 ootganootga kernel: IN=eth1 OUT=eth0 SRC=xx.xxx.82.132 DST=10.11.7.105 LEN=132 TOS=0x00 PREC=0x00 TTL=110 ID=998 PROTO=UDP SPT=3475 DPT=1035 LEN=112

Here is the response from the client. There will be no logging performed here by the PIX as the connection has already been built.

Feb 16 22:33:14 ootganootga kernel: IN=eth0 OUT=eth1 SRC=10.11.7.105 DST=xx.xxx.82.132 LEN=108 TOS=0x00 PREC=0x00 TTL=127 ID=43434 PROTO=UDP SPT=1035 DPT=3475 LEN=88

The above shows an acknowledgement of receipt from the server of the client's conv_who_are_you reply. Again, there is no logging done here by the PIX since the connection has been previously built.

Feb 16 22:33:14 pixfirewall Feb 17 2003 04:33:14: %PIX-6-302015: Built outbound UDP connection 133473 for outside:xx.xxx.82.132/3475 (xx.xxx.82.132/3475) to inside:10.11.7.105/1026 (xx.xx.218.241/34675)

Feb 16 22:33:14 ootganootga kernel: IN=eth0 OUT=eth1 SRC=10.11.7.105 DST=xx.xxx.82.132 LEN=112 TOS=0x00 PREC=0x00 TTL=127 ID=43435 PROTO=UDP SPT=1026 DPT=3475 LEN=92

After determining the request was not a duplicate, the server replies to the client's original request. This is a new connection originating from the port the Messenger Service is bound to. As this is a new connection, notice we also have a new PIX log message.

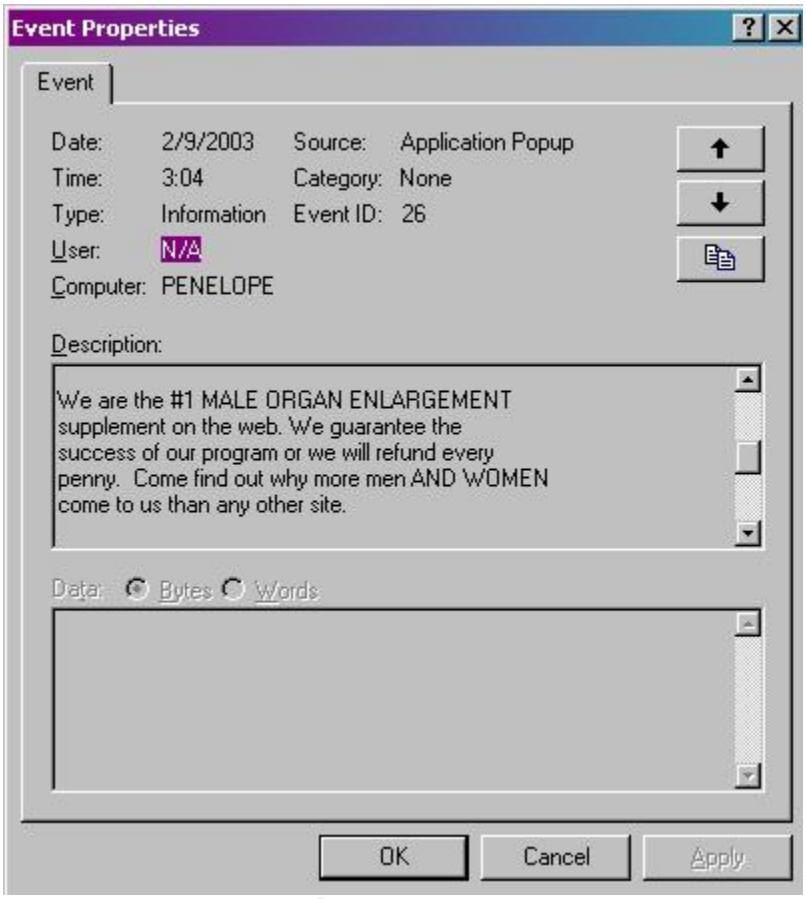
Feb 16 22:33:14 ootganootga kernel: IN=eth1 OUT=eth0 SRC=xx.xxx.82.132 DST=10.11.7.105 LEN=108 TOS=0x00 PREC=0x00 TTL=110 ID=999 PROTO=UDP SPT=3475 DPT=1026 LEN=88

…and the client acknowledges the server.

As the reader can see (and hopefully the **long** explanations were understandable) sending messages via RPC can be quite noisy in firewall logs (at least where Iptables and PIX are concerned). The end result, of course, will be that an advertisement of some sort has been displayed on the user's desktop. Aside from an annoying, focus stealing popup box, there are other clues left around for an end user to know what has happened. If we look under

Settings → Control Panel → Administrative Tools → Event Viewer → System Log

we'll find a little present left for us by the Windows Event Log. When popup messages are delivered by the system, it will create an entry in the "System Log" with a "Source" of "Application Popup." Admittedly, if the particular machine or user in question regularly receives legitimate messages, there is no way to distinguish between legitimate and non-legitimate Popups without actually viewing the specific log entry. Below is a screen cap of an actual Spam having been recorded in the Event Viewer:



We've now covered the signatures this type of activity leaves behind at network entry points (firewalls) and what it leaves behind on the end user side of things. All we're missing now is the stuff in-between, and this is where it gets real fun! Below I'll go into detail on how to detect the various programs and what they're doing. For this project, I'll need something a bit more granular than what we've been working with. Typically, this area is best suited for an Intrusion Detection System (IDS) because it can break open packets and inspect the innards. This is what we want, and for this project I have chosen Snort v1.9.0. Snort is an Open-Source (meaning free to use and redistribute) IDS that can be found at http://www.snort.org.

The first thing that needs to be done in order to pinpoint the various programs is to find something that is unique among them. I've got an idea that their initial ICMP pings will do the trick here. Unfortunately, of the three programs being tested, Sly Sender does not first try to ping the target. We'll have to focus this part of the paper solely on Direct Advertiser and Broadcast Advertiser. The only other question that needs answering is whether or not we want to detect the ICMP echo request or the ICMP echo reply. It is my opinion that replies (attack responses, really) are more important because they imply that the attacker has actually made at least SOME contact with the intended target. Writing signatures

that look for attacks going to the target are ok, but they often can be blocked by firewalls after the packet has traversed the sniffer.

Let's look at a typical ICMP echo request and reply from Direct Advertiser:

```
23:07:08.639330 10.11.7.105 > 192.168.0.6: icmp: echo request
0x0000   4500 003c 02e8 0000 8001 65b7 0a0b 0769        E..<.....e....i
0x0010   c0a8 0006 0800 738e 0200 0500 0f00 0000        ......s.........
0x0020   0000 0000 48c3 d700 a02d 2b81 3011 db00        ....H....-+.0...
0x0030   70ff 5801 4f27 4000 48c3 d700                  p.X.O'@.H...

23:07:08.643776 192.168.0.6 > 10.11.7.105: icmp: echo reply
0x0000   4500 003c a56d 0000 7f01 c431 c0a8 0006        E..<.m.....1....
0x0010   0a0b 0769 0000 7b8e 0200 0500 0f00 0000        ...i..{.........
0x0020   0000 0000 48c3 d700 a02d 2b81 3011 db00        ....H....-+.0...
0x0030   70ff 5801 4f27 4000 48c3 d700                  p.X.O'@.H...
```

As the reader will remember, we have a 20-byte IP header and an 8-byte ICMP header. This leaves us with an ICMP payload that I've colored in blue. I've taken the liberty of looking through various ICMP echo requests generated by Direct Advertiser (the reader can compare the above sample with the earlier sample in "*Diagram*" section) looking for static patterns that could be used to tag the program. The parts of the payload that are bolded represent a pattern that does not appear to change. It is always the same pattern, and always within the first 8-bytes of the payload. I've also highlighted another portion with italics. These could also be used as a signature, though beware; there are two bytes in between these static patterns that will change across reboots. I opted to go with matching on the first 8-bytes, this should be sufficient. The rule necessary to do this is written as:

```
alert icmp $HOME_NET any -> $EXTERNAL_NET any (msg:"POSSIBLE DA ICMP PING
REPLY"; content:"|0f00 0000 0000 0000|";itype:0;depth:8;)
```

The basic format of a snort rule is (ICMP doesn't use ports so these are effectively ignored):

<action> <protocol> <home network> <port> <directional notation> <external network> <port> <rule options>.

The rule options above are:

Msg →	Name of the rule when triggered (shown in the logs)
Content →	What to look for. The pipes (| symbol) tell snort what is between them should be interpreted as HEX. If the pipes are not there, it is interpreted as ASCII. Spaces aren't required, it's only spaced for readability.
Itype →	The ICMP type. 0 is echo reply (8 is echo request). A complete list of ICMP types can be found at http://www.spirit.com/Resources/icmp.html.

Depth → Signifies how far into the packet to look for the pattern. We are only concerned about the first 8-bytes.

This rule is saying "look for an ICMP echo reply coming from my HOME_NET to the EXTERNAL_NET that has the pattern 0f00 0000 0000 0000. For me, HOME_NET is defined as 192.168.0.0/16 and EXTERNAL_NET is defined as 10.11.7.0/24. I am running Snort on the Linux firewall, and for me, my logs are in /var/log/snort/<IP> (<IP> is a directory of the same name of the IP that triggered the alert. The text files contained within are the alert and packet dumps). The resulting output from our above rule looks like:

```
[**] POSSIBLE DA ICMP PING REPLY [**]
02/18-22:58:53.390077 192.168.0.6 -> 10.11.7.105
ICMP TTL:127 TOS:0x0 ID:42323 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:512  ECHO REPLY
0F 00 00 00 00 00 00 00 48 C3 D7 00 20 C0 43 81  ........H... .C.
30 11 DB 00 70 FF 58 01 4F 27 40 00 48 C3 D7 00  0...p.X.O'@.H...
```

This alert generally just gives us a little bit of information about what was captured. It's really the same stuff we've been looking at, so some of it may be familiar. Before we move on, let's dissect the alert to clear up any ambiguities the reader might have.

[**] POSSIBLE DA ICMP PING REPLY [**]
The text between the decorative end markers is the name of the signature that we're using.

02/18-22:58:53.390077 192.168.0.6 -> 10.11.7.105
Following that we have the date and time the packet was captured, along with the source (192.168.0.6) and the destination (10.11.7.105).

ICMP TTL:127 TOS:0x0 ID:42323 IpLen:20 DgmLen:60
We can learn from this line the protocol is ICMP, the Time to Live is 127, there is no Type of Service, the ID of the packet is 42323, the IP header length is 20 bytes, and the total size of the packet is 60 bytes.

Type:0 Code:0 ID:512 Seq:512 ECHO REPLY
The ICMP type is zero, the code is zero, the ICMP ID is 512, the ICMP sequence number is 512, and the ICMP type is ECHO REPLY (but we already know this from the type/code values).
ICMP ID numbers are usually the same as the ID of the current process and used for differentiating between different replies if multiple (for example, ping) programs are running.

ICMP SEQ numbers are used to determine if a response from (for example, ping) a program was lost. If we receive sequences 1,2,3,5, we know that 4 was lost.

Everything else is the payload. Notice what is in blue is the same sequence of HEX that we were looking for in our signature rule.

All right! Now we're finished explaining what things are, let's look into creating some other signatures. Here's what an ICMP request looks like with Broadcast Advertiser:

```
21:11:54.702983 10.11.7.105 > 192.168.0.6: icmp: echo request (ttl 255, id 3672, len 92)
0x0000   4500 005c 0e58 0000 ff01 db26 0a0b 0769        E..\.X.....&...i
0x0010   c0a8 0006 0800 ec0a 0200 1500 a7a7 a7a7        ..............
0x0020   a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7        ..............
0x0030   a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7        ..............
0x0040   a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7 a7a7        ..............
0x0050   a7a7 a7a7 a7a7 a7a7 a7a7 a7a7                  ............
```

I believe the pattern here is rather obvious. ☺ Let's change the direction of the signature, and instead of alerting on the responses, let's just alert on the initial request:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"POSSIBLE BA ICMP PING
REQUEST"; content:"| a7a7 a7a7 a7a7 a7a7 a7a7 a7a7|";itype:8;depth:12;)
```

I've also changed the depth to be 12 for no real reason other than to show that now we're looking at the first 12 bytes. Our alert would look like:

```
[**] POSSIBLE BA ICMP PING REQUEST [**]
02/19-21:17:04.834999 10.11.7.105 -> 192.168.0.6
ICMP TTL:255 TOS:0x0 ID:3934 IpLen:20 DgmLen:92
Type:8  Code:0  ID:512   Seq:5888  ECHO
A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7  ................
A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7  ................
A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7  ................
A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7 A7  ................
```

As said, we can't detect Sly Sender with ping rules, so let's move on to detecting RPC requests (which is more useful anyway. It's not as important to be able to identify a program as much as it is to identify what it's doing). The following is independent of any of the programs mentioned in this paper. DCE RPC works the same no matter what program is being used.

A typical request looks like:

```
21:23:35.447883 10.11.7.105.1128 > 192.168.0.6.135:  [udp sum ok] udp 170 (ttl 128, id 4351,
len 198)
0x0000   4500 00c6 10ff 0000 8011 5706 0a0b 0769        E.........W....i
0x0010   c0a8 0006 0468 0087 00b2 d8de 0400 0800        .....h..........
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0        ......{Z........
0x0040   4fb6 e6fc 839b fd7c 2fd2 d147 afcf 71a0        O......|/..G..q.
0x0050   0aef 6642 0000 0000 0100 0000 0000 0000        ..fB............
0x0060   0000 ffff ffff 5a00 0000 0000 0900 0000        ......Z.........
```

```
0x0070   0000 0000 0900 0000 4154 4553 544d 5347        ........ATESTMSG
0x0080   0000 0000 0c00 0000 0000 0000 0c00 0000        ................
0x0090   3139 322e 3136 382e 302e 3600 1e00 0000        192.168.0.6.....
0x00a0   0000 0000 1e00 0000 5448 4953 2049 5320        ........THIS.IS.
0x00b0   4120 5445 5354 2e20 4845 4c4c 4f4f 4f4f        A.TEST..HELLOOOO
0x00c0   4f4f 210d 0a00                                  OO!...
```

In this detect, we're going to get a bit more sophisticated. I've taken the liberty of coloring the "request" type (Remember, it's 0x00) in blue, and the Interface (for Messenger Service) UUID in red. This is what we want to match: A request for the Messenger Service. The following rule accomplishes this:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"MS RPC REQUEST (Messenger)";
content:"|00|";offset:1;depth:1; content:"|f891 7b5a 00ff d011 a9b2 00c0 4fb6 e6fc|"; offset:24;
depth:16;)
```

This rule is different in two ways.

1. We are now using two sets of "content" rules. Both patterns must be matched.
2. We've added a "depth" directive. This rule goes to the mark specified by "offset" and only reads in the amount of bytes specified in the "depth" directive.

Basically what we're doing here is going to offset one (from start of UDP payload, which is the DCE RPC header) and checking the next byte for 00 (a request). Next, look for f891 7b5a 00ff d011 a9b2 00c0 4fb6 e6fc starting at offset 24 and only read the next 16 bytes. Our output looks like:

```
[**] MS RPC REQUEST (Messenger) [**]
02/19-21:48:58.499211 10.11.7.105:1142 -> 192.168.0.6:135
UDP TTL:128 TOS:0x0 ID:6953 IpLen:20 DgmLen:198 Len: 178
04 00 08 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 F8 91 7B 5A 00 FF D0 11  ..........{Z....
A9 B2 00 C0 4F B6 E6 FC 32 F8 7C 00 0F E5 5A 40  ....O...2.|...Z@
86 58 A7 C8 81 B4 EF 85 00 00 00 00 01 00 00 00  .X..............
00 00 00 00 00 00 FF FF FF FF 5A 00 00 00 00 00  ..........Z.....
09 00 00 00 00 00 00 00 09 00 00 00 41 54 45 53  ............ATES
54 4D 53 47 00 00 45 00 0C 00 00 00 00 00 00 00  TMSG..E.........
0C 00 00 00 31 39 32 2E 31 36 38 2E 30 2E 36 00  ....192.168.0.6.
1E 00 00 00 00 00 00 00 1E 00 00 00 54 48 49 53  ............THIS
20 49 53 20 41 20 54 45 53 54 2E 20 48 45 4C 4C   IS A TEST. HELL
4F 4F 4F 4F 4F 4F 21 0D 0A 00                    OOOOOO!...
```

What if we wanted to get a lil' bit crazy with snort? There's an option that I find most useful, and I rarely consider an IDS that doesn't have this feature. It is the ability to alert on a packet, and then capture a pre-defined amount of packets after the initial match. This is good for watching what goes on after a successful signature trip. In snort, this is called "tagging." A tagged rule looks much the same as a normal rule, except with a "tag" directive:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"MS RPC REQUEST (Messenger)";
content:"|00|";offset:1;depth:1; content:"|f891 7b5a 00ff d011 a9b2 00c0 4fb6 e6fc|"; offset:24;
depth:16;tag:host,5,packets;)
```

The general format is tag: <session|host> <count> <packets|seconds>.

What we're saying above is after the initial match, tag the next 5 packets from that host. The output of this is placed into their corresponding <IP> directories. To wit:

```
ootganootga:/var/log/snort# ls -alR 10.11.7.105/ 192.168.0.6/
10.11.7.105/:
-rw-------   1 root     root        1099 Feb 19 22:25 UDP:1142-1026
-rw-------   1 root     root        3228 Feb 19 22:25 UDP:1142-135

192.168.0.6/:
-rw-------   1 root     root        1814 Feb 19 22:25 UDP:1760-1142
```

Within these files we have the next 5 packets of the conversation (I won't paste them out here in the interest of conserving space with things we've seen many times already).

I believe that about covers this section. We can spot it from the network entry points, the end user's station, and the network in-between. Now that we know what it looks like, how do we stop it you ask? I'm glad you asked!

How to protect against it

There are really only three ways to prevent this type of exposure. We'll discuss them each individually.

- **Disabling Messenger Service**

  This really is as easy as it sounds; it's not a daunting task! The following are the steps for Windows 2k and NT:

  1. Navigate to Start → Settings → Control Panel → Administrative Tools
  2. Click on "Services"
  3. Scroll down to "Messenger" in the list of services
  4. Right click → Properties
  5. In the drop-down menu, Startup type, change the type from "Automatic" to "Disabled."
  6. Click Apply/Ok
  7. Right click again and then select "Stop"

  The steps for XP are about the same:

  1. Navigate to Start → Settings (if present) → Control Panel
  2. Click "Performance and Maintenance" (if using category view)

3. Administrative Tools → Services
4. Scroll down to "Messenger" in the list of services
5. Right click → Properties
6. In the drop-down menu, Startup type, change the type from "Automatic" to "Disabled"
7. Click Apply/OK
8. Right click again and then select "Stop"

The Messenger Service is now stopped, and will not restart.

It is also possible to stop the Messenger Service via the command shell by typing "*net stop messenger*." This will stop the Messenger Service, but not permanently. On every reboot it will restart until the Startup Type has been changed to "Disabled."

Since the Messenger Service is stopped, this also means that legitimate Pop-ups will also be disabled. Also, any services that require the Messenger Service will not start. So, having said that, this option won't work for everybody - especially those that use "net send" or rely on anti-virus, UPS, or print spool messages. However, for most end users with a simple setup, this option will do well. The Messenger Service can of course be turned on again by changing the type back to "Automatic" and clicking "Start."

Alternatively, we can also just partially disable the Messenger Service by hacking the registry and adding the key *HKEY_LOCAL_MACHINE/Software/Microsoft/RPC/Internet*. Within this key, create a new String Value of "PortsInternetAvailable" of the type REG_SZ and set the value to N. After rebooting, this will only allow pop-ups from the local machine. This is in my opinion the best way of handling the pop-ups, as it still allows local functions to work properly. Of course, the next section, "Using a firewall" is also good advice and should be considered in addition to disabling the service.

- **Using a firewall**

  If the above suggestions are not workable, due to the fact that the Messenger Service is needed, this will be the next best solution. However, from a security point of view, even if the above **does** work, a firewall is an excellent added layer of protection, and should also be seriously considered. There are really two types of firewall software considered here – 3rd party, and the firewalling software included with XP.

  Whichever are used, the critical port to block is UDP port 135. This is where the initial DCE RPC request goes first. However, since Microsoft RPC also listens on TCP port 135, this should also be blocked. With this

port blocked, the end user should be safe, right? Well, sorta. As the reader will remember, the RPC Mapper was created for a reason, and that reason is to tell requesting programs on what port a particular **dynamically assigned** service is listening. Just because someone can't talk to the Mapper to determine on what port a service is listening, doesn't mean they can't scan for it and then muck about with it directly. The best all around solution here (and anywhere for that matter) is to block all in, and then allow what is needed. If that isn't feasible, it is still possible to define a range of ports that RPC services will use. Within the registry key, *HKEY_LOCAL_MACHINE/Software/Microsoft/RPC/Internet*, we need to add two more values in addition to "PortsInternetAvailable." The first value is "Ports," of type REG_MULTI_SZ. This value is the port range, such as 60000-62000, that any RPC services will be restricted to. A registry-editing tool that supports REG_MULTI_SZ will need to be used, such as regedt32. The second key is called "UseInternetPorts." The values for this are either Y (RPC services will use the ports defined in "Ports") or N (RPC services will use ports other than the ones defined in "Ports").

Other ports to block that can be related to Messenger Spam (though, not over RPC, using NetBIOS instead) are UDP ports 137, 138, and 445, TCP ports 139 and 445. Again, as above, it's just better to block everything and allow only what is necessary.

Next I'm going to briefly cover a few of the 3rd party firewalls and the XP firewall.

XP's software firewall:
Under Windows XP (both Home and Pro) users have the option of using the Internet Connection Firewall, or ICF for short. ICF is the same as any other end user firewall and can be configured to work on a dialup, DSL, Cable, LAN, etc.

To configure ICF, navigate "Start → Settings (if present) → Control Panel → Networking and Internet Connections (if using category view) → Network Connections." Once here, find the connection that should have ICF applied to it, right click on it, and then click properties. Under the "Advanced" tab, click the check box (unchecking this box disables ICF) labeled "Protect my computer and network by limiting or preventing access to this computer from the Internet."

The types of services and protocols can also be configured here under the "Settings" button. This is basically a point-and-click area. If an FTP server is desired, click it. Logging can also be configured from the "Security Logging" tab. Once finished, continue clicking OK to return to the "Network Connections" screen.

If ICF is not available or desired, there are other types of firewalls around. A couple other examples of software firewalls are:

ZoneAlarm: http://www.zonelabs.com/store/content/home.jsp:
The default setup of ZoneAlarm blocks all inbound traffic. To configure the firewall, go to the "Firewall" tab, then "Zones." From here other computers can be added to the Trusted zone, or the Internet zone (untrusted). The free version doesn't have as many configuration options. The free version can be downloaded from their web site, which also comes with a basic email attachment scrubber.

Norton Personal Firewall 2003: http://www.symantec.com/sabu/nis/npf/:
The default setup of Norton will also block all inbound access to the local machine. Firewall settings can be changed via "Status & Settings" → "Personal Firewall" → "Configure" → "Advanced" → "General Rules" → "Add." At this point there's a wizard that will allow the user to decide what actions to take on what ports and protocols, and from/to who.

Norton also has some other nice features like WWW ad blocking, privacy control, and Intrusion Detection rules among other things. A demo 15-day demo version is available at their site.

There are also firewall appliances such as the following low-end devices:

NetScreen-5XP:
http://www.netscreen.com/products/appliances.html#ns208ns204

CiscoPIX 501:
http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/ps2031/index.html

As with any firewall, whether it is software or hardware, adding rules may break Internet connectivity. Great care should be taken to ensure that required network functions have not been impaired by the addition of a firewall.

- **Installing Pop-up killers**

I've seen multiple websites, which mention installing these applications that will prevent pop-up Spam from reaching the user. Basically they run either in the task tray or the task bar and give the user an option of enabling or disabling messenger Spam. In reality though, after having looked at a couple of the products, all they do is the equivalent of "net stop messenger." A few products below are:

http://www.messenger-stopper.com: No demo available, the cost is $24.95

http://www.ipmessageblocker.com/: A 10-use demo is available. Demo can be downloaded off the their main page, and the cost for the full version is $9.95.
http://www.stopmessengerspam.com/stop_messenger_spam/stop_messenger_spam.html: A free demo is available from stopmessengerspam.com at the link listed.

Fixing the Issue

As mentioned earlier, the real problem is that anyone can access the RPC service (and ultimately the Messenger Service) via the Internet (or Localnet too). There are a few options that I can think of that Microsoft could do to alleviate the problem. Microsoft could simply just limit the exposure of the Messenger Service by having it not register itself with RPC. This has limited usefulness, however. Sure, it would effectively stop RPC Spam, but yet this type of exposure is also available over the NetBIOS ports. Removing it from NetBIOS could also work, but that would also kill the usefulness of the Messenger.

Microsoft could provide an easily accessible configuration tool that would allow the end-user or administrators to configure what IPs are allowed to interact with the Messenger Service. This is also mildly effective. As stated many times, the protocol that we've been dealing with in this paper is UDP, which is a connectionless protocol. Since connectionless protocols are easily spoofable, any IP that is allowed in could easily be spoofed. I say this is mildly effective because during the conv_who_are_you request the server generates a new Activity UUID, which would need to be returned in the client's response. If this were spoofed, the server's request would go somewhere else, and I feel it is unlikely that the client could guess the Activity UUID. However, having said all of that, spoofing and sending data (whether random, malicious, etc) would still allow anyone to muck with the Messenger Service.

I believe that the real problem is that in the current setup, RPC is wide open to anyone, lacking any sort of authentication. I believe that because the Messenger Service is an important part of the machine (virus alerts, critical system notices, administrator notices, etc), access to the Messenger Service should be authenticated/authorized. In fact, DCE RPC already has authentication/authorization support already built into it! This type of fix is more suited to an enterprise environment where there is a Windows domain available. In single user scenarios, I do believe that the "PortsInternetAvailable" registry hack from above is the best solution. Anyway, let's take a look at how it could possibly work.

Generally speaking, the client usually specifies that authentication is desired by making use of the auth_proto field (encompasses one byte) in the DCE header. The DCE spec talks primarily about Secret Key/Kerberos5, but other protocols can be implemented, such as NTLMSSP (NTLM Security Support Provider).

Requests to authenticate via NTLM are handled by NTLMSSP, and this would allow us to authenticate against the Windows Domain.

Since we have a service that requires authorization, any unauthorized attempt to access the restricted service would cause the server to return a reject to the client. The second time around (or even the first time if so configured to do so) the client would request authentication via NTLM. Upon receiving the request, the server would first have to verify whether or not the client and server have previously established a security context, and whether that context is still valid (timeout, expired, new session, etc). If the security context is still valid, communications can continue, otherwise the server will need to request credentials from the domain so that it can include those into its initial challenge/response sent to the client.

In order to exchange authorization credentials, the server will make use of the conversation functions again, only this time it will use conv_who_are_you_**auth** (in lieu of conv_who_are_you). This request includes the credentials the server acquired from the "security server," or in this case, the Windows domain server. If the client's response is too large to fit into one PDU (datagram, basically), the client will return a status of rpc_s_partial_credentials. This causes the server to loop a response of conv_who_are_you_auth_more that will continue until the entire client credentials have been received. If the credentials check out ok, a new security context is created, otherwise a reject is returned to the client.

The request for the Messenger Service could now continue as it normally would.

As an added bonus when using auth_proto, the PDU body will now contain an "authentication verifier" which contains fields such as "protection_level" that can be used to request certain levels of client protection. The types of protection range from none up to stub encryption and stub integrity verification. A complete list can be found at http://www.opengroup.org/onlinepubs/9629399/chap13.htm#tagcjh_18_01.

Source code / Pseudo code

This next section is where we'll really get into how the program works internally. Unfortunately, none of the programs that this paper has discussed have any source code available. As these commercial programs are likely the bread-and-butter of the companies selling them, this is not surprising to me at all. Fortunately the programs themselves are very simple in what they do to exploit the vulnerability, and as mentioned earlier, the real meat of the exploitation is in the NetMessageBufferSend() function. This function's sole responsibility is to take a message (aka buffer) and send it to the destination user, host, etc. According to Microsoft Developer Network, the layout of the function is as follows:

```
NET_API_STATUS NetMessageBufferSend(
  LPCWSTR servername,
  LPCWSTR msgname,
  LPCWSTR fromname,
  LPBYTE buf,
  DWORD buflen
);
```[10]

servername – is defined as the server where this function will execute. In most cases (especially for the ones in this paper) this will be NULL. A NULL servername indicates this function will be run locally. Special privileges are required to run this function on a remote server, such as being in a print, operator, or admin group.

msgname – is defined as the intended target of the message.

fromname – is defined as the name of where the message appears to come from (this is where someone could pretend to be the Administrator). A NULL value will use the name of the local machine.

buf – This is a buffer where the message to send is located

buflen – This is the length in bytes of the "*Buf*" pointer.

This function returns NERR_Success upon successful send, otherwise an error of any of the following may occur (these are mostly self-explanatory):

ERROR_ACCESS_DENIED – The user has insufficient access.
ERROR_INVALID_PARAMETER – The user has entered a bad parameter
ERROR_NOT_SUPPORTED – An unsupported request was issued.
NERR_NameNotFound – The target name was not located.
NERR_NetworkError – Some sort of network error occurred.

This function is located within the NetAPI32.dll library. So how does this function work in action? Well, since there isn't any source code available, I took the liberty of looking around this grand ol' Internet for a program whose requirements are to send a spoofed message using NetMessageBufferSend. I believe this to be an acceptable substitute, as these programs (including the ones mentioned throughout this paper) are simple and work nearly identical. In order to make use of the protocol, they all must follow by the same rules. The only differences that factor in are coding language choice (Direct Advertiser and Broadcast Advertiser seem to have been written in Delphi, where as Sly Sender was written in Microsoft Visual C++), and any other bells and whistles, such as an ICMP ping before sending the message.

[10] "NetMessageBufferSend." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/netmessagebuffersend.asp (23 February 2003)

I've settled on some code written by "jkr" on a message board hosted at www.experts-exchange.com. This c++ code can be found in its entirety at http://www.experts-exchange.com/Programming/Programming_Languages/Cplusplus/Q_10827401.html.

The pseudo code can really be summed up as saying "get the user input for the target address, the from name, and the message." The NetMessageBufferSend() function is all that is required, and it is truly simple. However, having said that, let's walk through how the pseudo code would work. This will be based on the above source code.

```
mbstowcs(<string>, <argument>)
<string>[MAXLEN-1] =L'\0'
```

The first thing that must be done in preparation to send the message is to convert each of the strings (server, name, from, msg) into UNICODE format. This is required by NetMessageBufferSend, and accomplished with the mbstowcs() function. Each string must be properly NULL terminated, and this is what the second line accomplishes.

```
NetMessageBufferSend(server,name,from,buf,buflen);
```

Exploiting the vulnerability is now as simple as calling the function NetMessageBufferSend() with the user arguments.

```
if return_value != NERR_Success
  Print_error
```

Checking the return value of NetMessageBufferSend() for a value of something other than NERR_Success will alert the user to whether or not the send was successful.

Additional Information

Well, we've come to the end of our journey into Messenger Spam, and its good friend, MS RPC. I've included below some additional resources that haven't been included in this paper, but may be of interest to the reader.

News articles:

- http://www.enn.ie/news.html?code=8843273 (starts about half-way down)
- http://www.vnunet.com/News/1136143
- http://www.cnet.com/internet/0-3762-8-20694175-1.html

Commentary:

- http://www.gorge.org/opinion/popup.shtml
- http://www.trageser.com/computers/web/2046.html
- http://www.badads.org/october02.shtml

News Groups (sorry for the HUGE URLs, I couldn't find anything smaller):

- http://groups.google.com/groups?hl=en&lr=lang_en&ie=UTF-8&oe=UTF-8&threadm=50a101c27f55%24cbae7130%2435ef2ecf%40TKMSFTNGXA11&rnum=20&prev=/groups%3Fq%3D%2522direct%2Badvertiser%2522%26hl%3Den%26lr%3Dlang_en%26ie%3DUTF-8%26oe%3DUTF-8%26start%3D10%26sa%3DN
- http://groups.google.com/groups?hl=en&lr=lang_en&ie=UTF-8&threadm=anov95%24fr7ps%242%40ID-146822.news.dfncis.de&rnum=7&prev=/groups%3Fhl%3Den%26lr%3Dlang_en%26ie%3DISO-8859-1%26q%3D%2522directadvertiser%2522%2B%26sa%3DN%26tab%3Dwg
- http://marc.theaimsgroup.com/?l=incidents&m=103481230405447&w=2
- http://marc.theaimsgroup.com/?l=incidents&w=2&r=1&s=port+135&q=b
- http://marc.theaimsgroup.com/?l=incidents&w=2&r=1&s=popup+spam&q=b

The following have been used in this paper; however, they are excellent references for Direct Advertiser, as well as MS RPC. I feel that they should also be included here:

- http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm
- http://www.mynetwatchman.com/kb/security/articles/popupspam/index.htm

As mentioned earlier, a demo version can be downloaded from the following URLs (there is no source):

- http://www.directadvertiser.com/demo.html
- http://asia.cnet.com/downloads/pc/swinfo/0,39000588,39010592s,00.htm (which is just a link back to www.directadvertiser.com).

Yeah, this URL was also mentioned previously, but it's the closest thing there is to a CERT-like advisory so I wanted to include it again):

- http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q330904&

All the information anyone could ever possibly want on RPC can be researched at the following:

- http://www.opengroup.org/onlinepubs/9629399/toc.htm
- http://www.hsc.fr/ressources/breves/min_srv_res_win.en.html

Information on some useful tools used to manipulate the RPC service can be found at the following location:

- http://razor.bindview.com/tools/desc/rpctools1.0-readme.html

References

"Top Ten Ports." 11 January 2003. URL: http://isc.incidents.org/top10.html (11 January 2003).

Riley, Steve. "Active Directory Replication over Firewalls." March 2001. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/ittasks/tasks/adrepfir.asp (1 March 2003).

"eDonkey2000." 28 January 2003. URL: http://www.edonkey2000.com (28 January 2003).

"Neohapsis Ports List – Welcome." 31 January 2003. URL: http://www.neohapsis.com/neolabs/neo-ports/ (13 February 2003).

"PORT NUMBERS." 11 February 2003. URL: http://www.iana.org/assignments/port-numbers (13 February 2003).

"Port 135 loc-srv/epmap." URL: http://www.iss.net/security_center/advice/Exploits/Ports/135/default.htm (1 February 2003).

"Port Microsoft." URL: http://www.iss.net/security_center/advice/Exploits/Ports/groups/Microsoft/default.htm (13 February 2003).

"Windows NT, Terminal Server, and Microsoft Exchange Services Use TCP/IP Ports." 14 October 2002. URL: http://support.microsoft.com/default.aspx?scid=KB;en-us;q150543 (13 February 2003).

"XGEN: TCP Ports and Microsoft Exchange: In-depth Discussion." 22 February 2002. URL: http://support.microsoft.com/default.aspx?scid=KB;en-us;q176466 (13 February 2003).

"Microsoft RPC." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/comext_3aw3.asp (4 February 2003).

"The RPC Model." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/microsoft_rpc_model.asp (4 February 2003).

"Malformed Request to RPC Endpoint Mapper can Cause RPC Service to Fail." 16 October 2002. URL:

http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B305399 (1 February 2003).

“Patch Available for 'Malformed RPC Packet' Vulnerability.” 11 September 2000. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-066.asp (4 February 2003).

“CAN-2001-0509.” URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0509 (4 February 2003).

“Telnet to Port 135 Causes 100 Percent CPU Usage.” 9 August 2001. http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B162567 (4 February 2003).

NetWatchMan. “Source of Win PopUP SPAM – prevention steps.” URL: http://www.dslreports.com/forum/remark,4699774~root=security,1~mode=flat (13 February 2003).

Primrose, Brenna. “RE: Source of Windows PopUp SPAM” 15 October 2002. URL: http://lists.jammed.com/incidents/2002/10/0120.html (13 February 2003).

root@hd.mil.tw. “FFUUCCKK out Webpopup” 12 January 2003 URL: http://groups.google.com/groups?selm=avrcld%24o8o%40netnews.hinet.net&oe=UTF-8&output=gplain (13 February 2003)

McWilliams, Brian. “Spam Masquerades as Admin Alerts” 15 October 2003. URL: http://www.wired.com/news/technology/0,1282,55795,00.html (13 February 2003).

C, H. “RE: Source of Windows PopUp SPAM.” 17 October 2002. http://marc.theaimsgroup.com/?l=incidents&m=103487466228680&w=2 (13 February 2003).

“SANS NewsBites December 18, 2002 Vol. 4, Num. 51.” 18 December 2002. URL: http://www.sans.org/newsletters/newsbites/vol4_51.php (13 February 2003).

“Messenger Service Window That Contains an Internet Advertisement Appears.” 31 January 2003. URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q330904 (13 February 2003).

“Get Started with Windows Messenger.” URL: http://www.microsoft.com/windowsxp/windowsmessenger/getstarted/default.asp (13 February 2003).

Mullen, Tim. “Busting Pop-up Spam.” 20 October 2002. URL: http://online.securityfocus.com/columnists/117 (13 February 2003).

“Direct Advertiser: Instant messages, instant leads.” URL: http://www.directadvertiser.com (11 February 2003).

“AOL, Spammers Duke It Out.” 3 December 2002. URL: http://www.badads.org/december02.shtml (13 February 2003).

“A New Marketing Revolution.” URL: http://www.broadcastadvertiser.com/index.htm (13 February 2003).

“Marketing that’s one step ahead of the competition.” URL: http://www.broadcastmarketer.com/ (13 February 2003).

URL: http://www.sharewarejunction.com/info.asp?ProductID=8465 (13 February 2003).

“Mass Mailing Software page 1.” URL: http://www.softdeko.8m.com/mass_mailing_software.htm (13 February 2003).

“Sly Sender – bulk MESSENGER SERVICE client.” URL: http://www.slysender.com/ (13 February 2003).

C Greene, Thomas. “Windows Messenger is new spam vector.” 16 October 2002. URL: http://www.theregister.co.uk/content/archive/27634.html (13 February 2003).

Lemos, Robert. “AOL blocks instant spam.” 26 November 2002. URL: http://news.zdnet.co.uk/story/0,,t269-s2126516,00.html (13 February 2003).

Best, Jay “Non email popup spam” 23 October 2002. URL: http://archive.nznog.org/2002-10/msg00092.html (15 February 2003).

“NetBios Popups.” URL: http://www.gdargaud.net/Hack/NoSpam.html (15 February 2003).

“Messenger Service of Windows” 10 October 2002. URL: http://support.microsoft.com/default.aspx?scid=kb%3ben-us%3b168893 (15 February 2003).

“Messenger Service (Net Send).” 11 December 2002. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/westudio/htm/_component_Messenger_Service.asp (15 February 2003).

"Alerter Service." 11 December 2002. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/westudio/htm/_component_Alerter_Service.asp (15 February 2003).

"Message Functions." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/message_functions.asp (15 February 2003).

"NetBIOS Suffixes (16 Character of the NetBIOS Name)." 10 January 2003. URL: http://support.microsoft.com/default.aspx?scid=kb%3ben-us%3b163409 (15 February 2003).

"Background Information." URL: http://www.stopmessengerspam.com/background_info/background_info.html (17 February 2003).

Stevens, W. Richard. TCP/IP Illustrated, Volume1. Reading: Addison Wesley Longman, Inc, 1994. 143-145,159,462.

Sakellariadis, Spyros. "Protecting Windows RPC Traffic." August 2002. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/isa/maintain/RPCwISA.asp (18 February 2003).

"How RPC Works." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp (18 February 2003).

De Clercq, Jan. "RPC Dynamic Port Allocation." URL: http://www.ntsecurity.net/Articles/Index.cfm?ArticleID=22206 (18 February 2003).

Akerman, Richard. "NET SEND On Windows." URL: http://www.chebucto.ns.ca/~rakerman/trojan-port-table.html#netsend (18 February 2003).

"Rpcdump.exe: RPC Dump." URL: http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/rpcdump-o.asp (18 February 2003).

"How to Use Portqry.exe to Troubleshoot Microsoft Exchange Server Connectivity Issues." 10 November 2002. URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;q310298& (18 February 2003).

Baldwin, Lawrence. "Windows Messenger Delivery options: SMB vs. MS RPC." URL: http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm (13 February 2003).

Baldwin, Lawrence. “myNetWatchman Alert – Windows PopUP SPAM.” 9 December 2002. URL: http://www.mynetwatchman.com/kb/security/articles/popupspam/ (13 February 2003).

Flynn, Gary. “…continuing saga of Windows Messenger SPAM, was re: (blank).” 16 October 2002. URL: http://online.securityfocus.com/archive/75/295561 (13 February 2003).

Open Group, The. “RPC PDU Encodings.” http://www.opengroup.org/onlinepubs/9629399/chap12.htm (13 February 2003).

“idempotent.” URL: http://dictionary.reference.com/search?q=idempotent (13 February 2003).

Open Group, The. “Conversation Manager Interface Definition.” URL: http://www.opengroup.org/onlinepubs/9629399/apdxp.htm (13 February 2003).

“Net Send – Microsoft Messenger Explained.” URL: http://www.exploitresearch.org/faqs/net_send.html (16 February 2003).

Rose, Kevin. “Spam Takes New Form.” 4 March 2002. URL: http://www.techtv.com/screensavers/answerstips/story/0,24330,3374542,00.html (16 February 2003).

Leach, Salz. “UUIDs and GUIDs (DRAFT).” 4 February 1998. URL: http://www.opengroup.org/dce/info/draft-leach-uuids-guids-01.txt (16 February 2003).

“System Log Messages” 6 February 2003. URL: http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/syslog/pixemsgs.htm (18 February 2003).

Unknown. “Netfilter Log Format.” URL: http://logi.cc/linux/netfilter-log-format.php3 (18 February 2003).

“ICMP Types and Codes.” URL: http://www.spirit.com/Resources/icmp.html (19 February 2003).

Green, Chris. “Writing Snort Rules.” URL: http://www.snort.org/docs/writing_rules/chap2.html#tth_chAp2 (19 February 2003).

Spring, Tom. “Sneaky New Form of Online Ads Pops Up.” 6 December 2002. URL: http://pcworld.shopping.yahoo.com/yahoo/article/0,aid,107754,00.asp (20 February 2003).

Sabin, Todd. “Win2ksecadvice: Multiple Remote DoS vulnerabilities in Microsoft DCE/RPC deamons.” 30 July 2001. URL: http://lists.insecure.org/lists/win2ksecadvice/2001/Jul/0023.html (20 February 2003).

Panders. “Managing Your NT Services.” URL: http://www.arstechnica.com/tweak/nt/ntservices-2.html (20 February 2003).

“How to stop Messenger Service pop up spam!” URL: http://www.bysoft.se/sureshot/articles/stopmessenger.html (22 February 2003).

“Stop Messenger Spam for Free.” URL: http://www.stopmessengerspam.com/ (22 February 2003).

“How to stop Messenger Spam: Disable Windows Messenger Service.” 3 December 2002. URL: http://www.auburn.edu/oit/security/messengerService.html (22 February 2003).

“Messenger Stopper.” URL: http://www.messenger-stopper.com/ (22 February 2003).

“IP Messageblocker. Stop unwanted IP messages!” URL: http://www.ipmessageblocker.com/ (22 February 2003).

J. Vaughan-Nichols, Steven. “Beating Messenger Spam.” 4 November 2002. URL: http://www.practical-tech.com/infrastructure/i11042002.htm (22 February 2003).

H. Corbett, Jr., Larry. “How to Block Messenger Spam.” 20 September 2002. URL: http://www.re-quest.net/computers/messenger-spam/ (22 February 2003).

Bair, Tom. “Get a free update CD.” February 2003. URL: http://www.computerbits.com/archive/2003/0200/bair0302.html (22 February 2003).

“HOW TO: Enable or Disable Internet Connection Firewall in Windows XP.” 15 November 2002. URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;283673 (22 February 2003).

Nelson, Michael. “Using Distributed COM with Firewalls.” 19 March 1999. URL: http://www.microsoft.com/com/wpaper/dcomfw.asp (22 February 2003).

"Malformed NTLMSSP Request Can Enable Code to Run with System Privileges." 7 February 2001. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-008.asp (22 February 2003).

Open Group, The. "RPC Protocol Definitions." URL: http://www.opengroup.org/onlinepubs/9629399/chap9.htm (22 February 2003).

"NetMessageBufferSend." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/netmessagebuffersend.asp (23 February 2003).

"NetMessageBufferSend." URL: http://www.allapi.net/apilist/NetMessageBufferSend.shtml (23 February 2003).

jkr. "net send through c++." 27 July 2000. URL: http://www.experts-exchange.com/Programming/Programming_Languages/Cplusplus/Q_10827401.html (23 February 2003).

Appendices

Appendix A

Below are the network traces showing various differences between the products making use of the Messenger RPC service. These traces were captured with tcpdump.

Direct Advertiser v1.0
21:54:19.525675 10.11.7.105 > 192.168.0.6: icmp: echo request (ttl 128, id 4856, len 60)
21:54:19.530290 192.168.0.6 > 10.11.7.105: icmp: echo reply (ttl 127, id 50862, len 60)
21:54:19.532813 10.11.7.105.2015 > 192.168.0.6.135: [udp sum ok] udp 336 (ttl 128, id 4857, len 364)
21:54:19.621735 192.168.0.6.2697 > 10.11.7.105.2015: [udp sum ok] udp 100 (ttl 127, id 50864, len 128)
21:54:19.621997 10.11.7.105.2015 > 192.168.0.6.2697: [udp sum ok] udp 104 (ttl 128, id 4858, len 132)
21:54:19.626806 192.168.0.6.2697 > 10.11.7.105.2015: [udp sum ok] udp 80 (ttl 127, id 50865, len 108)
21:54:19.734821 192.168.0.6.1026 > 10.11.7.105.2015: [udp sum ok] udp 84 (ttl 127, id 50866, len 112)
21:54:19.735246 10.11.7.105.2015 > 192.168.0.6.1026: [udp sum ok] udp 80 (ttl 128, id 4859, len 108)

Broadcast Advertiser v3.0.1
22:01:17.386806 10.11.7.105 > 192.168.0.6: icmp: echo request (ttl 255, id 4878, len 92)
22:01:17.391711 192.168.0.6 > 10.11.7.105: icmp: echo reply (ttl 127, id 50923, len 92)
22:01:17.393793 10.11.7.105.2018 > 192.168.0.6.135: [udp sum ok] udp 254 (ttl 128, id 4880, len 282)
22:01:17.400182 192.168.0.6.2697 > 10.11.7.105.2018: [udp sum ok] udp 100 (ttl 127, id 50925, len 128)
22:01:17.400370 10.11.7.105.2018 > 192.168.0.6.2697: [udp sum ok] udp 104 (ttl 128, id 4881, len 132)
22:01:17.405145 192.168.0.6.2697 > 10.11.7.105.2018: [udp sum ok] udp 80 (ttl 127, id 50926, len 108)
22:01:17.407213 192.168.0.6.1026 > 10.11.7.105.2018: [udp sum ok] udp 84 (ttl 127, id 50927, len 112)
22:01:17.407367 10.11.7.105.2018 > 192.168.0.6.1026: [udp sum ok] udp 80 (ttl 128, id 4882, len 108)

Sly Sender v4.0
22:49:02.566031 10.11.7.105.2023 > 192.168.0.6.135: [udp sum ok] udp 170 (ttl 128, id 4972, len 198)
22:49:02.572135 192.168.0.6.2697 > 10.11.7.105.2023: [udp sum ok] udp 100 (ttl 127, id 51328, len 128)
22:49:02.572352 10.11.7.105.2023 > 192.168.0.6.2697: [udp sum ok] udp 104 (ttl 128, id 4973, len 132)
22:49:02.577239 192.168.0.6.2697 > 10.11.7.105.2023: [udp sum ok] udp 80 (ttl 127, id 51329, len 108)
22:49:02.579613 192.168.0.6.1026 > 10.11.7.105.2023: [udp sum ok] udp 84 (ttl 127, id 51330, len 112)
22:49:02.579796 10.11.7.105.2023 > 192.168.0.6.1026: [udp sum ok] udp 80 (ttl 128, id 4974, len 108)

The following is a list of what processes were bound to what ports on the remote end session. This list was gathered using "fport," from Foundstone. This tool can be downloaded from www.foundstone.com.

Pid	Process		Port	Proto	Path
432	svchost	->	135	UDP	C:\WINNT\system32\svchost.exe
228	services	->	1026	UDP	C:\WINNT\system32\services.exe
228	services	->	2697	UDP	C:\WINNT\system32\services.exe

- Messenger service is provided via services.exe
- The Mapper is provided via svchost.exe

Appendix B

This appendix lists protocol types for “rpcdump” as provided by the usage help.

Protocol:(default ncacn_ip_tcp)
ncacn_np (Connection-oriented named pipes)
ncacn_mq (Datagram (connectionless) over the Microsoft Message Queue Server)
ncadg_ipx (Datagram (connectionless) IPX)
ncacn_spx (Connection-oriented SPX)
ncacn_http (Connection-oriented TCP/IP using Microsoft Internet Information Server as HTTP proxy.)
ncacn_nb_nb (Connection-oriented NetBEUI)
ncacn_nb_tcp (Connection-oriented NetBIOS over TCP)
ncacn_nb_ipx (Connection-oriented NetBIOS over IPX)
ncacn_ip_tcp (Connection-oriented TCP/IP)
ncacn_at_dsp (AppleTalk DSP)
ncadg_ip_udp (Datagram (connectionless) UDP/IP)
ncacn_vns_spp (Connection-oriented Vines SPP transport
ncacn_dnet_nsp (Connection-oriented DECnet transport)
ncacn_nb_xns (Connection-oriented XNS)

Appendix C

Traces of Interface not registered error.

```
19:44:50.726098 10.11.7.105 > 192.168.0.6: icmp: echo request
0x0000   4500 003c 0bfe 0000 8001 5ca1 0a0b 0769        E..<......\....i
0x0010   c0a8 0006 0800 2ed9 0200 1100 0f00 0000        ..........
0x0020   0000 0000 98c4 d700 20e0 4381 3011 db00        ..........C.0...
0x0030   70ff 5801 4f27 4000 98c4 d700                  p.X.O'@.....

19:44:50.730656 192.168.0.6 > 10.11.7.105: icmp: echo reply
0x0000   4500 003c 1abd 0000 7f01 4ee2 c0a8 0006        E..<......N.....
0x0010   0a0b 0769 0000 36d9 0200 1100 0f00 0000        ...i..6.....
0x0020   0000 0000 98c4 d700 20e0 4381 3011 db00        ..........C.0...
0x0030   70ff 5801 4f27 4000 98c4 d700                  p.X.O'@.....

19:44:50.737067 10.11.7.105.1090 > 192.168.0.6.135:  udp 335
0x0000   4500 016b 0bff 0000 8011 5b61 0a0b 0769        E..k......[a...i
0x0010   c0a8 0006 0442 0087 0157 86f6 0400 0800        .....B...W......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ..........
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0        ......{Z........
0x0040   4fb6 e6fc 9f5d db90 16d6 e748 bd1a 63af        O....]....H..c.
0x0050   2880 a2a9 0000 0000 0100 0000 0000 0000                (..........
0x0060   0000 ffff ffff ff00 0000 0000 0c00 0000        ..........
0x0070   0000 0000 0c00 0000 4144 5645 5254 4953        ........ADVERTIS
0x0080   494e 4700 0c00 0000 0000 0000 0c00 0000        ING.............
0x0090   3139 322e 3136 382e 302e 3600 c300 0000        192.168.0.6.....
0x00a0   0000 0000 c300 0000 4669 6c6c 206f 7574        ........Fill.out
0x00b0   2074 6865 2054 6f49 5020 6669 656c 6420        .the.ToIP.field.
0x00c0   7769 7468 2074 6865 2049 5020 6164 6472        with.the.IP.addr
0x00d0   6573 7320 796f 7520 7761 6e74 2074 6f20        ess.you.want.to.
0x00e0   7365 6e64 200d 0a79 6f75 7220 6d65 7373        send...your.mess
0x00f0   6167 6520 746f 2c20 7468 656e 2065 6e74        age.to,.then.ent
0x0100   6572 2061 206d 6573 7361 6765 2068 6572        er.a.message.her
0x0110   652c 2061 6e64 2068 6974 2053 7461 7274        e,.and.hit.Start
0x0120   2e0d 0a54 6865 2072 6567 6973 7465 7265                ...The.registere
0x0130   6420 7665 7273 696f 6e20 7769 6c6c 2073        d.version.will.s
0x0140   656e 6420 6d65 7373 6167 6573 2074 6f20        end.messages.to.
0x0150   7468 6520 7365 6c65 6374 6564 2049 5020        the.selected.IP.
0x0160   0d0a 7261 6e67 652e 0d0a 00                    ..range....

19:44:50.742943 192.168.0.6.135 > 10.11.7.105.1090:  udp 84
0x0000   4500 0070 1abe 0000 7f11 4e9d c0a8 0006        E..p......N.....
0x0010   0a0b 0769 0087 0442 005c 8004 04 06 0000       ...i...B.\......
0x0020   1000 0000 0000 0000 0000 0000 0000 0000        ..........
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0        ......{Z........
0x0040   4fb6 e6fc 9f5d db90 16d6 e748 bd1a 63af        O....]....H..c.
0x0050   2880 a2a9 0000 0000 0100 0000 0000 0000                (..........
0x0060   0000 ffff ffff 0400 0000 0000 0300 011c        ..........
```

We can determine that the last datagram is a “reject” packet, as made known by the **06**. The reject code (0x1c010003) is underlined. All of the different reject codes can be found at http://www.opengroup.org/onlinepubs/9629399/apdxe.htm#tagtcjh_43. This particular code is nca_unk_if, which means that Interface isn’t registered.

Appendix D

These traces were captured with Ethereal, and displayed with Tethereal. Tethereal can be found in the Ethereal distribution.

Firewall on the server blocking inbound UDP >1023.

```
 1   0.000000 00:50:8b:f0:b5:30 -> ff:ff:ff:ff:ff:ff ARP Who has 10.11.7.77?  Tell 10.11.7.105
 2   0.000130 00:c0:f0:6d:08:73 -> 00:50:8b:f0:b5:30 ARP 10.11.7.77 is at 00:c0:f0:6d:08:73
 3   0.000143  10.11.7.105 -> 192.168.0.6  ICMP Echo (ping) request
 4   0.008109  192.168.0.6 -> 10.11.7.105  ICMP Echo (ping) reply
 5   0.014838  10.11.7.105 -> 192.168.0.6  DCERPC Request: seq_num: 0 opnum: 0 (initial DCERPC
request always uses UDP port 135)
 6   0.021951  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
 7   0.022069  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply (this would be blocked by
firewall – UDP 1024+)
 8   1.010650  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0 (uses port 135)
 9   1.016017  192.168.0.6 -> 10.11.7.105  DCERPC Working: seq_num: 0
10   1.022403  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
11   1.022534  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply
12   3.010682  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
13   3.025420  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
14   3.025603  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply
15   4.026649  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
16   4.999540 00:c0:f0:6d:08:73 -> 00:50:8b:f0:b5:30 ARP Who has 10.11.7.105?  Tell 10.11.7.77
17   4.999574 00:50:8b:f0:b5:30 -> 00:c0:f0:6d:08:73 ARP 10.11.7.105 is at 00:50:8b:f0:b5:30
18   6.027437  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
19   7.031320  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
20   7.032579  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply
21  11.026578  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
22  15.042972  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
23  15.043189  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply
24  23.043110  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
25  31.066337  192.168.0.6 -> 10.11.7.105  CONV conv_who_are_you2 request
26  31.066550  10.11.7.105 -> 192.168.0.6  CONV conv_who_are_you2 reply
27  41.042447  10.11.7.105 -> 192.168.0.6  DCERPC Cl_cancel: seq_num: 0[Malformed Packet]
```

Firewall on server blocking outbound UDP >1023 errors

```
 1   0.000000  10.11.7.105 -> 192.168.0.6  ICMP Echo (ping) request
 2   0.004581  192.168.0.6 -> 10.11.7.105  ICMP Echo (ping) reply
 3   0.010363  10.11.7.105 -> 192.168.0.6  DCERPC Request: seq_num: 0 opnum: 0 (client won't see
anything else from server after this point)
 4   1.003980  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
 5   2.003577  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
 6   4.004313  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
 7   8.004374  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
 8  16.003695  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
 9  32.004608  10.11.7.105 -> 192.168.0.6  DCERPC Ping: seq_num: 0
10  32.019735  10.11.7.105 -> 192.168.0.6  DCERPC Cl_cancel: seq_num: 0[Malformed Packet]
```

Appendix E

Below is a Tethereal dump of how Direct Advertiser determines a machine's speed. I've also included the hosts that it pings when checking network connectivity.

```
 1   0.000000  10.11.7.105 -> 66.218.71.198 ICMP Echo (ping) request
 2   2.281023  10.11.7.105 -> 209.73.164.91 ICMP Echo (ping) request
 3   4.780798  10.11.7.105 -> 216.239.35.100 ICMP Echo (ping) request
 4   7.358172  10.11.7.105 -> 63.118.174.3 TCP 1122 > 80 [SYN] Seq=589652973 Ack=0 Win=64240
Len=0
 5   7.358325 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [SYN, ACK] Seq=2529955355 Ack=589652974
Win=5840 Len=0
 6   7.358350  10.11.7.105 -> 63.118.174.3 TCP 1122 > 80 [ACK] Seq=589652974 Ack=2529955356
Win=64240 Len=0
 7   7.358624  10.11.7.105 -> 63.118.174.3 HTTP POST /test.cfm HTTP/1.0
 8   7.358780 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589653196
Win=6432 Len=0
 9   7.358832  10.11.7.105 -> 63.118.174.3 HTTP Continuation
10   7.358884  10.11.7.105 -> 63.118.174.3 HTTP Continuation
11   7.358925  10.11.7.105 -> 63.118.174.3 HTTP Continuation
12   7.359186 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589654656
Win=8760 Len=0
13   7.359228  10.11.7.105 -> 63.118.174.3 HTTP Continuation
14   7.359259  10.11.7.105 -> 63.118.174.3 HTTP Continuation
15   7.359304 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589656116
Win=11680 Len=0
16   7.359326  10.11.7.105 -> 63.118.174.3 HTTP Continuation
17   7.359401  10.11.7.105 -> 63.118.174.3 HTTP Continuation
18   7.359415 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589657576
Win=14600 Len=0
19   7.359460  10.11.7.105 -> 63.118.174.3 HTTP Continuation
20   7.359504  10.11.7.105 -> 63.118.174.3 HTTP Continuation
21   7.359570 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589659036
Win=17520 Len=0
22   7.359647  10.11.7.105 -> 63.118.174.3 HTTP Continuation
23   7.359709 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589660496
Win=20440 Len=0
24   7.359766 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589661389
Win=23360 Len=0
25   7.359806  10.11.7.105 -> 63.118.174.3 HTTP Continuation
26   7.359905 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589662849
Win=26280 Len=0
27   7.359926  10.11.7.105 -> 63.118.174.3 HTTP Continuation
28   7.359997  10.11.7.105 -> 63.118.174.3 HTTP Continuation
29   7.360028 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589664309
Win=29200 Len=0
30   7.360133  10.11.7.105 -> 63.118.174.3 HTTP Continuation
31   7.360147 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589665769
Win=32120 Len=0
32   7.360195  10.11.7.105 -> 63.118.174.3 HTTP Continuation
33   7.360278 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589667229
Win=35040 Len=0
34   7.360406 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589668689
Win=37960 Len=0
35   7.360458 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589669582
Win=37960 Len=0
36   7.360588 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589671042
Win=40880 Len=0
```

```
 37   7.360724 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589672502
Win=43800 Len=0
 38   7.360838 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589673962
Win=45260 Len=0
 39   7.362058  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 40   7.362165  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 41   7.362340  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 42   7.362369  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 43   7.362420 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589675422
Win=43800 Len=0
 44   7.362473  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 45   7.362514  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 46   7.362537 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589676882
Win=42340 Len=0
 47   7.362584  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 48   7.362632  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 49   7.394378 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589685075
Win=35040 Len=0
 50   7.394518  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 51   7.394662  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 52   7.394706  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 53   7.394747  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 54   7.394798  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 55   7.394842  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 56   7.428105 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589693268
Win=46720 Len=0
 57   7.428315  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 58   7.428504  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 59   7.428546  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 60   7.428594  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 61   7.428606 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589694161
Win=46720 Len=0
 62   7.428659  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 63   7.428713  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 64   7.428844 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589695621
Win=49640 Len=0
 65   7.428872  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 66   7.428960  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 67   7.428974 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589697081
Win=52560 Len=0
 68   7.429085 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589698541
Win=55480 Len=0
 69   7.429105  10.11.7.105 -> 63.118.174.3 HTTP Continuation
 70   7.429208 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589700001
Win=58400 Len=0
 71   7.464386 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955356 Ack=589704396
Win=54020 Len=0
 72  10.704527 63.118.174.3 -> 10.11.7.105  HTTP HTTP/1.0 200 OK
 73  10.704678 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [FIN, ACK] Seq=2529955762 Ack=589704396
Win=58400 Len=0
 74  10.704729  10.11.7.105 -> 63.118.174.3 TCP 1122 > 80 [ACK] Seq=589704396 Ack=2529955763
Win=63834 Len=0
 75  10.705128  10.11.7.105 -> 63.118.174.3 TCP 1122 > 80 [FIN, ACK] Seq=589704396 Ack=2529955763
Win=63834 Len=0
 76  10.705233 63.118.174.3 -> 10.11.7.105  TCP 80 > 1122 [ACK] Seq=2529955763 Ack=589704397
Win=58400 Len=0
```