GIAC
CERTIFICATIONS

# 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 permited 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

# The Welchia Worm

Gene Bransfield
GCIH Practical
V. 3
18 December 2003

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

**Abstract:**

The topic of this paper is the Welchia (aka Nachi) worm. This paper will discuss the vulnerabilities exploited by the Welchia worm, the methods of detection and prevention, and will recount the events of an incident handled by the author and his coworkers during Welchia worm infection.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

# Table of Contents


Introduction....1

Statement of Purpose....1

The Exploit....3
- Identification....3
- Exploit Description....3
- Protocols/Services/Applications Affected....4
- RPC....4
- DCOM....8
- RPC DCOM; Putting it All Together....11
- WebDAV....11
- Variants of Welchia....12
- Description....14
- The RPC DCOM Vulnerability....15
- The WebDAV Vulnerability....16
- Signatures of the Attack....17

The Platforms/Environments....24
- The Victim....24
- The Source....24
- The Network....24

Stages of the Attack....25
- Reconnaissance....25
- Scanning....26
- Exploiting the System....26
- Keeping Access....27
- Covering Tracks....27

The Incident Handling Process....28
- Prevention....28
- Identification....29
- Containment and Eradication....30
- Recovery....33
- Lessons Learned....33

Conclusion....35

Extras....36

References....40


© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

## Introduction

The debate between the pros and cons of "good" worms has been going on for years. The pro side of the argument states that a properly designed worm could discover a vulnerability, patch it, do no harm to the system, move on to the next system and delete itself when it is done. The pro side believes that this would be a quick and efficient way to make sure you and your neighbors are safe and fully patched against all vulnerabilities, and would also make patching the machines on your network much less of a hassle.

The con side, makes the argument of "Thou shalt not alter thy neighbor's system uninvited, lest ye be a hacker." Furthermore, the con side argues: Who will code these "fixes?" Who will distribute them? Who will ensure that these "fixes" won't break something else on "…my system that was working fine before you introduced your "friendly" worm?" Most importantly, the con side argues, how will a system administrator keep this code contained within the particular subnet for which it was designed? Well, score one for the con side of the debate. Let's talk about Welchia.

Shortly after the outbreak of the notorious MSBlaster worm, Bugtraq was pummeled with the age-old rants from frustrated administrators who were sick of people not patching their systems to known vulnerabilities such as the RPC DCOM vulnerability. The question was asked "Is it not possible to create another worm or modify this worm to actually patch the machines?" (Stuart) Perhaps that got someone thinking, because that's exactly what happened.

Unfortunately the worm's creator should have kept reading, because one of the first responses to that question was "You would be stepping on a lot of toes doing that. Not to mention breaking several laws." (Stickler) However, the worm creator paid the warnings no mind and went about his way creating the Welchia (aka Nachi) worm, much to the woe of many a network administrator.

Like every "good worm" before it, including Den_Zuko, Cheeze, CodeGreen and the Millenium worm, Welchia came in peace, but left the network in pieces. Whatever it's intentions, the end could not justify the means. The worm may have had the good intention of patching an infected system and removing any signs of the MSBlaster worm, but the execution of that mission was not carried out in a technically practical way. In the end, Welchia caused many more problems than it solved.

## Statement of Purpose

The purpose of this paper is to examine the Welchia worm and recount the actual events as experienced by myself and my colleagues while dealing with Welchia infection. Overviews will be given of the processes exploited, the vulnerabilities, and the worm itself. Further examination will describe the methods of prevention



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

and detection, outward symptoms of Welchia infection, methods of containment and methods of eradication. This paper will also delve slightly into the politics associated with incidents such as these, and the lessons learned in all aspects.

The organization involved in the incident is a government organization, and has requested to remain anonymous. Furthermore, the government organization has requested that any specific quotes from actual policy be summarized rather than quoted directly so as to further support anonymity. Therefore, while policy will be mentioned where appropriate, it will not be quoted. Also, the network design layout will be summarized to only represent network elements essential to the content of this paper.

© SANS Institute 2004, Author retains full rights.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

# The Exploit

## Identification

**Name of Exploit**: W32.Welchia.Worm [Symantec]

CVE Number CAN-2003-0352; CAN-2003-0109

CERT Number CA-2003-19; CA-2003-09

Bugtraq ID 8205; 7116

Also Known As W32/Welchia.worm10240 [AhnLab],
W32/Nachi.worm [McAfee],
WORM_MSBLAST.D [Trend],
Lovsan.D [F-Secure],
W32/Nachi-A [Sophos],
Win32.Nachi.A [CA],
Worm.Win32.Welchia [KAV]

**Systems Affected**: Microsoft IIS 5.0 on Windows 2000 server

Windows 2000 SP4 and earlier
Windows XP SP1 and earlier
Windows Server 2003

## Exploit Description

The Welchia worm is an example of a new breed of worms that include the capability to exploit multiple vulnerabilities within the same malicious code. Welchia exploits the RPC DCOM vulnerability as described in the Microsoft Security Bulletin MS03-026; as well as the WebDAV vulnerability as described in Microsoft Security Bulletin MS03-007.[1] While the worm is capable of two separate exploits, the code executed after infection the same, and the worm will display the exact same signature on the victim machine regardless of the vulnerability exploited to get onto the victim.

There are several unique characteristics with regard to this worm. First off, it deletes another notorious worm, MSBlaster, which is described in the Variants section of this paper. Secondly, it takes advantage of two vulnerabilities using two separate exploits: the RPC DCOM vulnerability and the WebDAV vulnerability. Next, it attempts to patch the RPC DCOM vulnerability; one of the vulnerabilities it attempts to exploit. Lastly, as if it is doing us a favor, it delets itself on 1 January 2004. If it weren't for the prolific ICMP traffic that brought

[1] MS03-026 and MS03-007 have been superseded by MS-03-039 and MS03-013 respectively.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

networks to their knees all over the country coupled with the backdoor left in its wake, this might be a fairly benign worm. Unfortunately, it is not, and it has ruined the weekends of many a network administrator who thought they were safe. A full description of the worm as described by Trend Micro is available at the end of this paper.

## Protocols/Services/Applications Affected

TCP 135 (RPC DCOM): RPC DCOM is a feature of the Microsoft Operating System. RPC DCOM is actually two separate entities; namely RPC and DCOM. To understand how these services work together, it is necessary to understand how they work separately.

## RPC

In the early days of programming, programs were huge monolithic constructions filled with 'goto' statements. All functions that the application had to accomplish needed to be included in the code. Naturally, these programs could become quite unwieldy and maintenance was cumbersome.

To facilitate the construction of programs, a procedure-oriented model was developed. Within the procedure-oriented model, a main procedure ran the overall application and called a subordinate procedure when necessary. This was referred to as a local procedure call. These subordinate procedures could contain frequently-used pieces of code called "libraries[2]," or be an actual separate piece of code associated with the main procedure. The main procedure did not need to know the inner-code associated with the subordinate procedure, it simply needed to provide standard input to that subordinate routine, and receive standard output in return.

Local procedure calls soon evolved to calling entirely different programs or processes rather than incorporating the functions of those processes within the main procedures or subordinate procedures of that program. This provided for an environment where a program did not require all aspects of hardware interaction to be included in the code. Called process (server) and calling processes (client) would link themselves together using addresses in the same local memory space. The static interaction – referred to as "linking" – would be transparent to the application-level process and the end user.

Remote Procedure Call (RPC) takes the local procedure call paradigm and brings it into an environment where the calling procedure (client) and the called procedure (server) reside on physically separate nodes connected by a

[2] There are many types of libraries; run-time libraries and compile-time libraries are two examples. Run-time libraries are consistent pieces of code used when the program is run, and compile-time libraries are consistent pieces of code used when the program is compiled. Most Microsoft libraries are stored in .dll files.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

communications network, such as in a LAN, WAN, or Internet environment; or logically separate processes in local memory, such as on a single physical workstation running separate instances of a Java Virtual Machine.

RPC works by a client dynamically binding itself to a server and vice versa. To allow a bind to be seamless, the client and server need to establish a binding handle on one another. A binding handle represents a logical instance of the client server connection. The complete binding information for remote entities is maintained in the respective binding handles. A client obtains binding information from the server, and the server maintains the binding information for every client it serves. The following components are necessary binding information:

Protocol Sequence – A combination of communications protocols to include a network protocol, a transport protocol, and an RPC protocol. A server specifies a specific set of protocol sequences it will use when listening for incoming calls.

Network Address Information – The complete network address of a remote entity. This would consist of an address and an endpoint; or rather an IP address and a port.

Transfer Syntax – An understood standard communication sequence between client and server.

RPC Protocol Version Numbers – Client and server must have compatible RPC Protocol Version Numbers to communicate.

UUID – Universal Unique Identifier. Usually combined with the Transfer Syntax and Protocol Version Number through the Interface Definition Language (IDL) to determine an Interface Identifier (Microsoft uses the proprietary Microsoft Interface Definition Language or MIDL)

In RPC, enpoints and endpoint mappers play a significant role. An endpoint is the address of a specific server instance on a host system (Opengroup, Remote Procedure Call Model). There are two types of endpoints; well-known and dynamic. Well-Known endpoints are controlled by a large central authority responsible for a certain Transport Protocol. For example, the Internet Assigned Numbers Authority is the organization in charge of maintaining endpoints for the IP protocol. TCP/UDP port 135 is the endpoint for Microsoft's RPC DCOM.

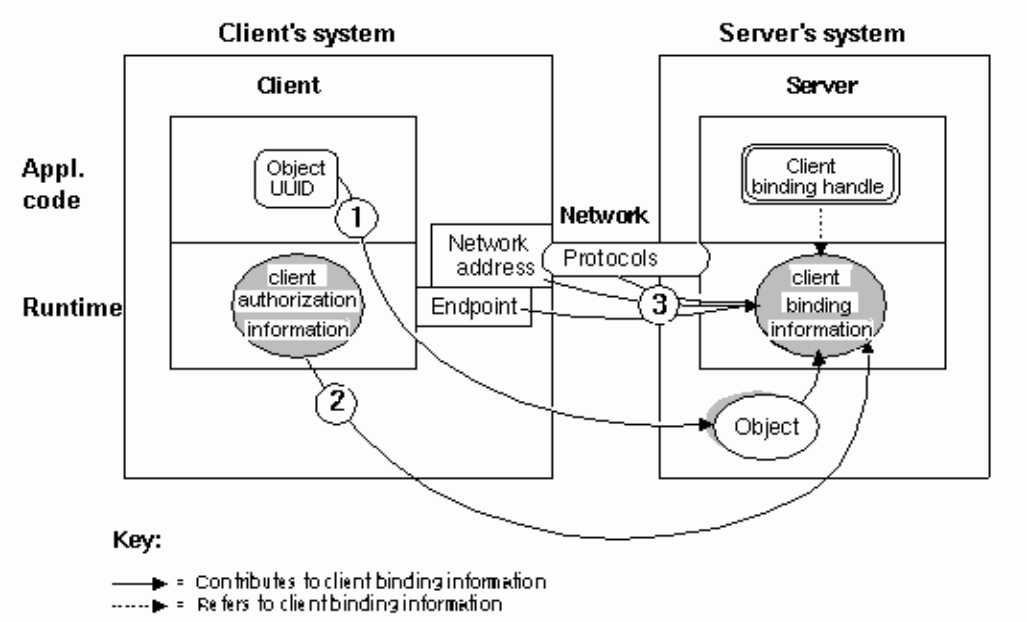
Dynamic endpoints are requested and assigned at the runtime of a particular RPC session. Dynamic endpoints require an endpoint mapper. When using a dynamic endpoint, a server must register its address and binding information with a name service. The server will also register its services and binding information with a local endpoint mapper. When a client wishes to use the services, it will query the name service for the server. The name service will then redirect the


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

client to the server's endpoint mapper so that communication may commence. The process involved with the endpoint mapper presents potential vulnerabilities, but that discussion is beyond the scope of this paper.

When any client initiates an RPC connection with a server, it must specify its own binding information to include Source address and endpoint; a valid transfer syntax; protocol sequence, and version number; the UUID of the requested service; and any client authentication information.[3] An example is shown here:

*Client Binding Information Resulting from a Remote Procedure Call*




The callouts in the figure refer to the following:
1. The requested object UUID, which may be the nil UUID
2. Client authentication information, which is optional
3. The address from which the client is making the remote procedure call, which the communications protocols supply to the server.[4]

The server maintains the client binding information and makes it available to the server application by a client binding handle. After communications are established, things look a bit more organized:[5]

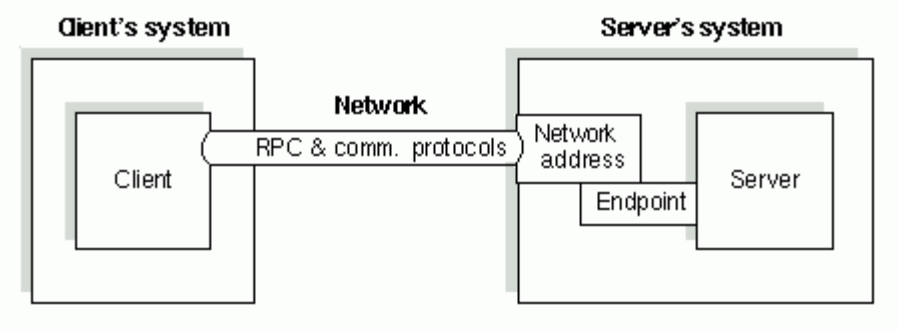
[3] RPC depends on another service to provide security for the transaction. Security in the Microsoft model is provided by COM and DCOM.

[4] Image from: http://support.entegrity.com/private/doclib/docs/osfhtm/develop/appdev/Appde273.htm

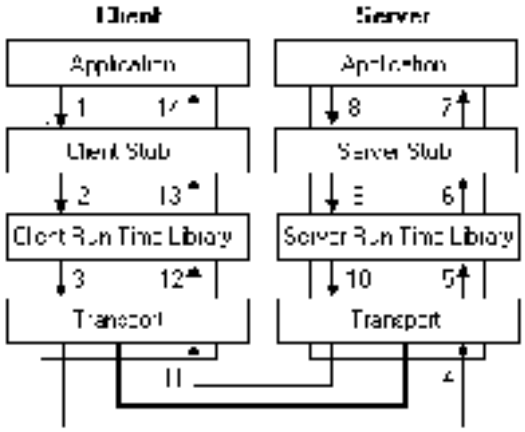
[5] Image from: http://support.entegrity.com/private/doclib/docs/osfhtm/develop/appdev/Appde265.htm



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.




The real purpose of RPC is to seamlessly integrate remotely called functions into the application that made the call. To establish an interface between the client application and the server application through RPC, an IDL compiler must be used to create the interface the applications will use to communicate. Microsoft uses the MIDL to create the interfaces. To create the IDL interfaces, the IDL compiler generates code called a stub. A stub will take the data from the application and will interact with the RPC run-time libraries, which will actually make the remote procedure call. When the data is received from the server, it is translated once again by the stub so that when the data is presented to the application it appears as though it was presented from a local service. This process helps ensure transparency of the underlying functions to the higher level application. An example of how the communication process works is shown here:[6]




Step 1: A Client application needs information on a remote server, and makes a local call for the information.

Step 2: The Client Stub receives the request, translates the information and sends it to the RPC run-time libraries.

Step 3: The RPC run-time libraries translate the information and send it to the transport layer for encapsulation and transmission to the remote server.

[6] Image from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Step 4: The request is received by the Transport layer of the remote server.

Step 5: The request is set by the Transport Layer to the server's RPC run-time libraries.

Step 6: The request is translated by the Server run-time libraries and sent to the server's stub.

Step 7: The server's stub process translates the information back into a request that can be understood by the server application and it is sent to the server application itself.

Step 8: The server application responds to the request for data and replies.

Step 9: The data is translated by the server's stub and sent to the server's RPC run-time libraries.

Step 10 -13: The data goes through the same transformation and communication processes back up to the Client's stub level.

Step 14: The Client's initial local call is responded to by the client stub with the data provided by the server.

Microsoft's RPC (MS-RPC ) is an implementation of the Open Standard RPC, with a few minor exceptions. For example, MS-RPC includes support for marshalling of interface pointers,[7] support for custom marshalling, and support for additional protocols such as Appletalk and Local RPC (Open Group, Remote Procedure Call Model). As long as the run-time libraries are supported, a program written using MS-RPC will be compatible with an RPC on another platform.

## DCOM

Microsoft's Distributed Component Object Model (DCOM) is Microsoft's Component Object Model (COM) in a distributed environment such as a Client-Server environment, a Local Area Network (LAN), a Wide Area Network (WAN), or over the Internet. Based on MS-RPC, COM's inception was to solve several problems:

Basic Interoperability – How do developers create unique programs and processes and have them work together on the operating system?

[7] Marshalling of interface pointers refers to a feature of DCOM



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Versioning – How does one piece of the system upgraded without requiring an entire system upgrade?

Language Independence – How does a program written in one language communicate with a program written in another language?

Transparent cross-process interoperability – How do all of the previous items happen in a fashion that is transparent to the application at hand and the end user?

What COM provides is an environment in which different programmers on different platforms using different languages to create applications. When these applications are interfaced with COM, they will have a standard set of communication protocols to connect to any other application on the operating system. COM is the set of binary standards that logically encapsulates an object, creating a component object. This set of binary standards dictates the data traffic that enters and exits the object, thus creating a standard of communication independent of any one programming language. Therefore, a Java object can be encapsulated in COM and be able to communicate with a C++ object that is also encapsulated in COM.

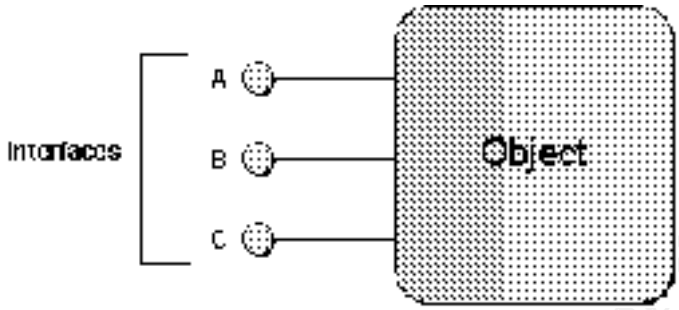
Component Objects are constructed in the same client-server relationship as described in RPC. Like RPC, COM communicates through the use of interfaces. However, the interfaces specified in COM are different. According to Microsoft, a COM interface is the definition of an expected behavior and expected responsibilities. In other words, a COM interface establishes how a client may interact with a server to obtain a certain server behavior. A friend of mine explained it this way: if the server component object was "car", COM would form a driver interface so that the client component object "human" would operate the driver interface as defined by COM (namely the steering wheel, accelerator, breaks) but would not interact with the inner workings of the server component object "car". If the "human" component wanted to turn right, it would turn the steering wheel to the right. However, the "car" component would actually run the mechanisms necessary to actually make the "car" component turn right.

A component object may define more than one interface, each with a unique identifier. Interfaces are not upgraded with their component object; if a new version of an object adds new features to a component object, new interfaces are defined with separate unique identifiers to handle calls to that feature.

Clients never actually interact with the interfaces themselves, rather, they interact through interface pointers. Interface pointers are references a client keeps that directs the client to the memory address of the component object interface it needs. Therefore, client calls are made to the interface pointer (much like an RPC stub). The interface pointer redirects the call to the memory address of the


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

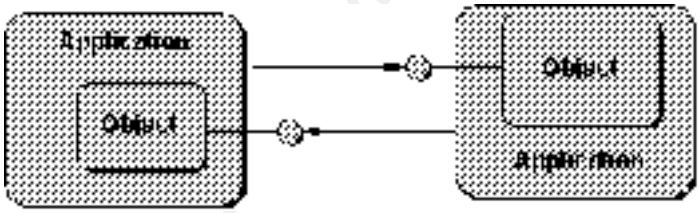
component object interface. The interface receives the call and handles data interaction with the actual object.




Above is an example of a component object with attached interfaces.[8]




Above is an example of how an interface pointer would interact with an interface.[9]



Above we have full communication through the Component Object Model.[10] Once COM establishes the communication between client and server, it drops out of the conversation. This alleviates any unnecessary overhead.

COM also handles all interaction of the component object with the operating system and any services the component object provides or requires. This binary standard interaction between component objects and their environment solves the problem of basic interoperability. The issues of versioning is solved in that component objects never actually interact with each other at a binary level, and therefore can be upgraded independently and still provide the standard input and output necessary.

Because the COM standards are set in specific COM libraries, data is subject to binary object standards vice Source code standards. This standard format of

[8] Image from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/msdn_comppr.asp

[9] dito

[10] dito.

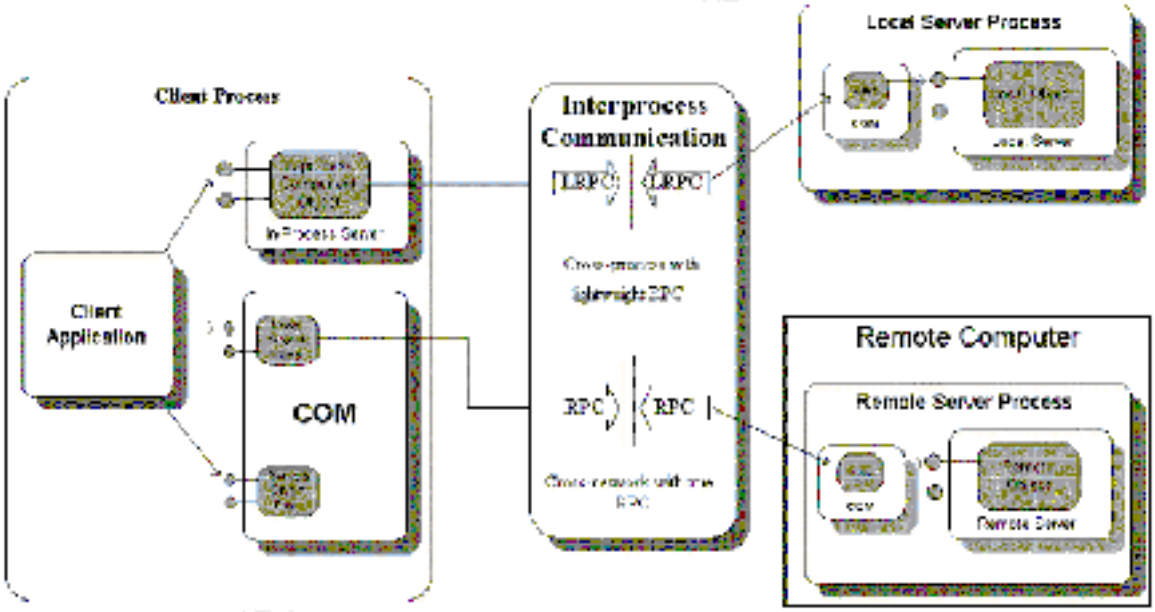


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

communicating ensures that each component object will be able to effectively communicate with another component object. Finally, COM uses the elements of RPC's transparent procedure calling through MS-RPC to provide transparent cross-process interoperability.

### RPC DCOM; Putting it All Together

DCOM, as previously stated, is simply COM in a distributed environment. While a normal COM client would look to its interface pointer to give it the address of the local server interface in local memory and perform a Local RPC using lightweight RPC, a DCOM's server process exists in a physically remote computer. Therefore, when a client COM object on a Microsoft machine wishes to access a remote process, DCOM translates the request into an RPC call using MS-RPC. On the server side, DCOM listens to the MS-RPC port for incoming calls, translates those calls to interact with the Microsoft server component object, and transmits the data. This process is demonstrated in parallel below:[11]




### WebDAV

WebDAV stands for "Web-based Distributed Authoring and Versioning". It is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote web servers (webdav.org). These days, most of the World Wide Web consists of read-only web pages. However, internet websites were originally conceived to provide collaboration between physically separate

[11] Image from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/msdn_comppr.asp



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

entities by allowing people to post and edit information on websites so that all could use the data. Unfortunately, most of the web browsers at the time lacked the capability to do this.

Then, in 1995 a company named NaviSoft made NaviPress, and another company named Vermeer made a package called FrontPage. Users now had the ability to edit web pages remotely. Then AOL bought NaviSoft and Microsoft bought Vermeer. This put major corporate money into research and development, as well as major corporate marketing toward the consumer. However, there existed a problem: to author web pages, a product had to assign extensions to HTTP. Since the products were developed independently, they developed independent methods of extending HTTP to meet their own needs. Naturally, these independently developed products were incompatable with each other.

In December 1995 a group of like minded individuals began the WebDAV working group to address the issue of interoperability. The WebDAV working group established standard practices for web authoring. Once WebDAV was had an accepted standard, web authoring exploded. Soon a multitude of products such as Microsoft Word and Corell WordPerfect, and Lotus WordPro had web editing capability built in.

Microsoft's implementation of WebDAV comes installed by default in the Windows 2000 package when IIS 5.0 is installed. The intended use of WebDAV is the remote configuration of IIS 5.0 servers and above. When integrated into IIS, WebDAV allows clients to do the following (Microsoft, About WebDAV):

**Manipulate resources** in a WebDAV publishing directory on your server. For example, users who have been assigned the correct rights can copy and move files around in a WebDAV directory.
**Modify properties** associated with certain resources. For example, a user can write to and retrieve a file's property information.
**Lock and unlock resources** so that multiple users can read a file concurrently. However, only one person can modify the file at a time.
**Search** the content and properties of files in a WebDAV directory.

### Variants of Welchia

**W32.Blaster.Worm** – AKA MSBlaster. Exploits the DCOM RPC vulnerability. The worm downloads the msblast.exe file to the %WinDir%\system32 directory and executes it. After execution on a host system, the worm first checks for infection. If the machine is infected, the attack stops. The worm adds the value: "windows auto update"="msblast.exe" to the registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run so that the worm runs when you start Windows. The worm then uses the


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

computer's IP address, generates a random address at the class B boundary of that IP address, and begins a scan for vulnerable hosts using TCP port 135.

Next, the worm installs a backdoor on the infected machine which listens on port 4444. This will allow a possible attacker to issue remote commands to the workstation. The worm then listens on UDP port 69 for incoming calls from newly infected machines. The worm will upload msblast.exe to the requesting computer. If the current date is the 16th through the end of the month for the months of January to August, or if the current month is September through December, the worm will attempt to perform a Denial of Service attack (DoS) on windowsupdate.com.[12] The DoS traffic has the following characteristics:

- Is a SYN flood on port 80 of windowsupdate.com.
- Tries to send 50 HTTP packets every second.
- Each packet is 40 bytes in length.
- If the worm cannot find a DNS entry for windowsupdate.com, it uses a destination address of 255.255.255.255.[13]

The worm contains the following text, which is never displayed:

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

**W32.Blaster.B.Worm** – Similar functionality to MSBlaster. Installs "penis.exe" vice "msblast.exe." Adds the value: "windows auto update"="penis32.exe" to the registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

**W32.Blaster.C.Worm** – Similar functionality to MSBlaster. Installs "teekids.exe" vice "msblast.exe." Adds the value: "Microsoft Inet Xp.."="teekids.exe" to the registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
also downloads index.exe and root32.exe; both are backdoor kits.

**W32.Balster.D.Worm** – Similar functionality to MSBlaster. Installs "mspatch.exe" vice "msblast.exe" Adds the value: "Nonton Antivirus"="mspatch.exe" to the registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run on.

[12] As of August 15th, 2003, Microsoft has removed the DNS record for windowsupdate.com.
[13] Because of this aspect of the worm, it is recommended that system administrators reroute windowsupdate.com to an internal IP address.

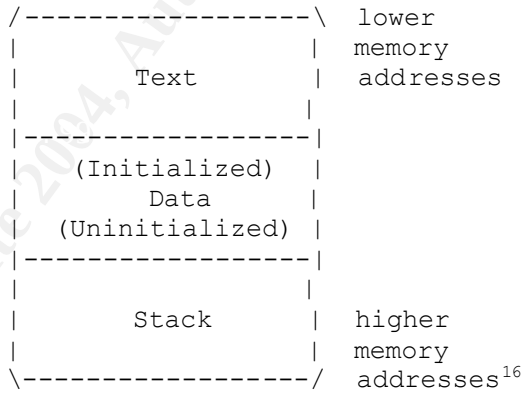

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

**W32.Blaster.D.Worm** – Similar functionality to MSBlaster. Installs "mslaugh.exe" vice "msblast.exe;" performs DoS attack against kimble.org.[14] Adds the value: windows automation"="mslaugh.exe" to the registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.

**W32.Blaster.D.Worm** – Similar functionality to MSBlaster. Installs "enbiei.exe" vice "msblast.exe;" performs DoS attack against tuiasi.ro.[15] Adds the value: "www.hidro.4t.com"="enbiei.exe" to the registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

## Description

The Welchia Worm exploits two separate vulnerabilities; the RPC DCOM vulnerability as described in MS03-026, and the WebDAV vulnerability as described in MS03-013. Both vulnerabilities are caused by unchecked code that results in a buffer overflow. The best popular description of a buffer overflow is found in the article "Smashing the Stack for Fun and Profit" by Aleph One in the periodical Phrack, volume 7 issue 49: link: http://www.securityfocus.com/archive/1/5667. As a brief description is in order, I will simplify the explanation:




As defined in "Smashing the Stack for Fun and Profit", a buffer is simply an area of contiguous memory allocated for multiple instances of the same data type. Understanding of a buffer overflow requires understanding of certain aspects of programming and how memory works on a computer. When an application is running, it must store code in the memory of a computer. When stored in memory, the application data is divided into three regions: Text, Data, and Stack. The Text section is static in nature and contains code and read-only data. The Data section contains initialized and uninitialized commands and is dynamic in

[14] As of 23 November 2003, kimble.org resolves to 127.0.0.1.
[15] As of 23 November 2003, tuiasi.ro does not resolve.
[16] Image from: http://www.phrack.org/show.php?p=49&a=14


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

nature. If it is determined that the data segment needs more room in memory to function, the process is stopped and scheduled to run at a later time in a larger memory space.

The stack is a dynamic portion of memory that handles input in a LIFO (Last In First Out) manner. This means that if the Stack is full, the last command that entered the stack queue will be the first to be eliminated. A register called the Stack Pointer (SP) points to the top of the stack, while the bottom of the stack is a fixed memory address. The CPU performs Push and Pop functions to move data onto and off of the stack; data is "pushed" onto the stack and "popped" off the stack as needed. Also involved in the process is the Frame Pointer (FP) which points to a fixed memory address on the stack

RPC calls are handled by an application with the help of the stack. While a called remote or local application is functioning, control of the stack is handled by the called application. When the process is finished, it returns control of the stack to the originating application. In most computer languages, local variables are referenced based on offsets from the SP. As data is pushed and popped from the stack, the SP changes. When the SP becomes dynamic, it becomes more and more difficult for the computer to keep track. Therefore, the static FP is often used in place of the SP.

When making a procedure call, FP is saved to an address in memory so that the remote application knows how and where to restore control to the originating process upon completion. When a call is made by the local application, the FP is moved to the SP so that SP can handle the remote application's data requirements. When the remote application is done, it uses the return memory address stored in the SP to return to FP.

When a buffer overflow occurs, the stack is simply given too much data to handle. This occurs when a parameter is placed in program code and is not checked properly to ensure the incoming variable data does not exceed memory requirements. For example, if a variable is expected to be 16 bytes in size and there is no code restricting the input size of that variable, an attacker may flood that variable with 256 bytes of data, thus causing a buffer overflow

When the buffer overflow occurs, the SP's return address for a particular function call is overwritten. Thus rather than returning to the memory address of the proper FP, it is redirected to a location in memory that contains malicious code. When the process reaches that space in memory, it executes the malicious code at the privilege level of the application that was exploited. If for example Administrator was running a process that was exploited by a buffer overflow attack, the malicious code would be executed at the Administrator level.

## The RPC DCOM Vulnerability


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The vulnerability exists where RPC and DCOM interface on a remote system. The Windows RPC service does not properly check data input under certain circumstances. An attacker connects to a Windows RPC and sends a specially formed RPC package that causes a buffer overflow in the underlying DCOM service. The vulnerability can be found in the following API (Flashsky):

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

The sixth parameter down is szName. In the local configuration, a packet can be created such as the following (Flashsky):

```
CoGetInstanceFromFile(pServerInfo,NULL,0,CLSCTX_REMOTE_SERVER,STGM_READ
WRITE,L"C:\\123 \4561111111111111111111111111.doc",1,&qi);
```

This packet would create a buffer overflow locally, but the local API checks for size and thus prohibits any data larger than 0x220, so the buffer overflow will not work locally. However, when the client transfers the parameter to the server, the server translates the input into the following (Flashsky):

```
L¡°\\servername\c$\123456111111111111111111111111.doc".
```

In this situation, there is no error checking before the data hits memory. Since the server process only allows 0x20 memory size for the parameter, and the input is greater than that size, a buffer overflow occurs. This vulnerability was discovered, developed and reported by the Last Stage of Delirium Research Group (http://lsd-pl.net). An excellent full analysis of the vulnerability can be found here: http://www.xfocus.org/documents/200307/2.html

Vulnerable operating systems include Windows NT 4.0 through service pack 6a, Windows 2000 through service pack 4, Windows XP through service pack 1 and Windows Server 2003.

### The WebDAV Vulnerability

The vulnerability exists by the way WebDAV for Windows 2000 server's ISS 5.0 interacts with the ntdll.dll in the core operating system component. An attacker can connect via HTTP to a vulnerable server and send a malformed HTTP request to WebDAV, such as (Madina):


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

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

Since a function call of ntdll.dll – the RtlDosPathNameToNtPathName_U – does not check parameters correctly, a buffer overflow occurs. The WebDAV exploit was developed by Roman Madina of Roman Soft Research Labs (www.rs-labs.com). An excellent, in-depth analysis of the exploit can be found here: http://www.fatelabs.com/library/fatelabs-ntdll-analysis.pdf

Vulnerable operating systems include Windows 2000 machines up to service pack 3 running IIS 5.0. WebDAV is installed by default with IIS 5.0. IIS 5.0 installs by default on Windows 2000 servers, and can be optionally installed as part of Windows 2000 professional.

NGSSoftware pointed out that the actual vulnerability exists not in WebDAV, but in a function call of ntdll.dll named RtlDosPathNameToNtPathName_U. Since the RtlDosPathNameToNtPathName_U function is a vital component in the Windows architecture, every service that uses it in a function call is potentially vulnerable to exploit. Those exploits are beyond the scope of this paper, but it is important to note that the particular vulnerability exploited by the WebDAV exploit exists on all Microsoft Operating systems and not just those that run ISS 5.0.

## Signatures of the Attack

According to McAfee, the Welchia worm will only attempt to exploit the WebDAV vulnerability after 200,000 RPC DCOM attacks. Since the network being attacked has no vulnerable IIS 5.0 servers attached; and the attempt to use WebDAV happens so infrequently, it is more informative and helpful to concentrate on the attack signatures most often detected during Welchia infection.

There are two glaring symptoms of Welchia infection: workstations spontaneously rebooting themselves (after infection); and the loss of bandwidth due to the ICMP traffic caused by Welchia-infected workstations attempting to find other vulnerable machines. Since any and all traffic not viewed as internal to the subnet is sent through the default gateway, as infection increases firewalls will be quickly overwhelmed and your network connectivity will die.

Fortunately, Welchia's prolific use of ICMP is also the easiest way to tell which host is infected. Simply follow the arp:

```
22:01:50.435946 arp who-has 192.168.0.115 tell 192.168.0.253
22:01:50.438301 arp who-has 192.168.0.116 tell 192.168.0.253
22:01:50.445362 arp who-has 192.168.0.117 tell 192.168.0.253
22:01:50.460087 arp who-has 192.168.0.118 tell 192.168.0.253
```



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
22:01:50.466885 arp who-has 192.168.0.119 tell 192.168.0.253
22:01:50.482358 arp who-has 192.168.0.120 tell 192.168.0.253
22:01:50.484681 arp who-has 192.168.0.121 tell 192.168.0.253
22:01:50.498546 arp who-has 192.168.0.122 tell 192.168.0.253
22:01:50.505680 arp who-has 192.168.0.123 tell 192.168.0.253
22:01:50.514562 arp who-has 192.168.0.124 tell 192.168.0.253
22:01:50.531488 arp who-has 192.168.0.125 tell 192.168.0.253
22:01:50.534873 arp who-has 192.168.0.126 tell 192.168.0.253
22:01:50.546532 arp who-has 192.168.0.127 tell 192.168.0.253
22:01:50.554933 arp who-has 192.168.0.128 tell 192.168.0.253
22:01:50.570009 arp who-has 192.168.0.129 tell 192.168.0.253
22:01:50.577407 arp who-has 192.168.0.130 tell 192.168.0.253
22:01:50.588931 arp who-has 192.168.0.131 tell 192.168.0.253
22:01:50.600770 arp who-has 192.168.0.132 tell 192.168.0.253
22:01:50.606802 arp who-has 192.168.0.133 tell 192.168.0.253
```

As we learned about Welchia, after a Class-B subnet has been chosen for a scan, Welchia will send ICMP requests in a "1-up" fashion at the Class-B boundary from 0-255 in rapid succession, and wait for a response before attempting infection. In the above example, The Welchia has chosen the 192.168 Class-B segment based on the IP address of it's host. It will search from 192.168.0.1 through 192.168.255.255 for hosts to infect. It is quite clear that address 192.168.0.253 has been infected with Welchia.

Welchia's ping payload is also a dead giveaway due to the prolific number of 0xaa values contained within (Perriot).

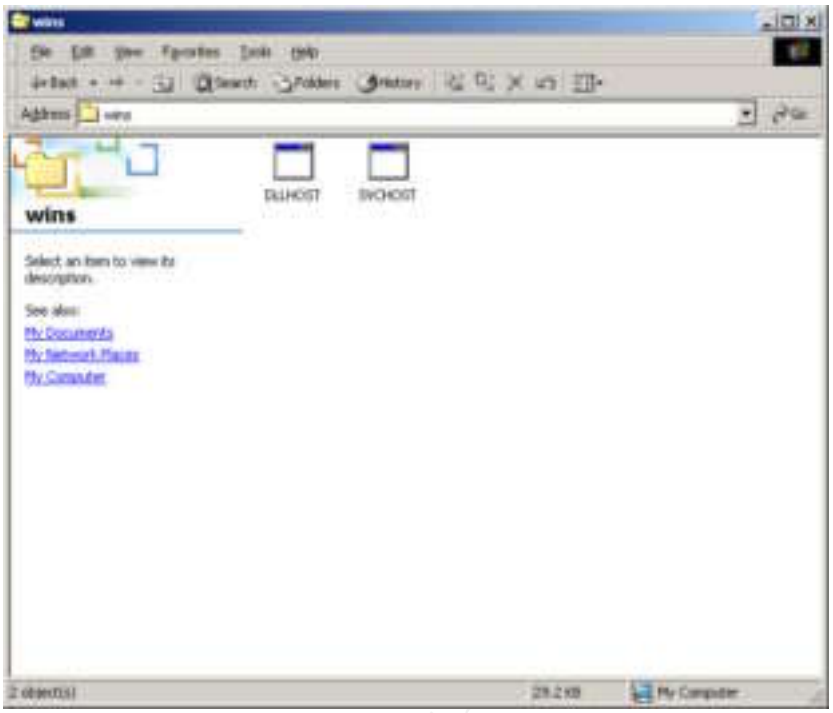
```
11:47:47.576542 169.254.56.166> 169.254.189.84: icmp: echo request
0x0000 4500 005c 599d 0000 8001 970c a9fe 38a6 E..\Y.........8.
0x0010 a9fe bd54 0800 fa51 0200 a658 aaaa aaaa ...T...Q...X....
0x0020 aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa ................
0x0030 aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa ................
0x0040 aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa ................
0x0050 aaaa aaaa aaaa aaaa aaaa aaaa ............
```

However, perhaps the quickest way to track down infected machines is to simply perform the command "tcpdump –qn icmp" or "windump –qn icmp" on the afflicted firewall. The amount of ICMP traffic from the infected machine will be prolific and "1-up" in nature. The signature will be impossible to miss.

Once the machine you suspect of carrying out the attack has been located, infection can be verified by examining the machine itself. If Welchia has infected the machine, the following symptoms will be displayed:

- The %WinDir%\System32\Wins directory will contain the file dllhost.exe and the file svchost.exe. Dllhost.exe is the name of a valid Windows file, but it should not be located in the %WinDir%\System32\Wins directory. Svchost.exe is actually the Tftpd.exe copied from the %WinDir%\Dllcache. An example is given below of the C:\Winnt\System32\Wins directory of an infected machine.

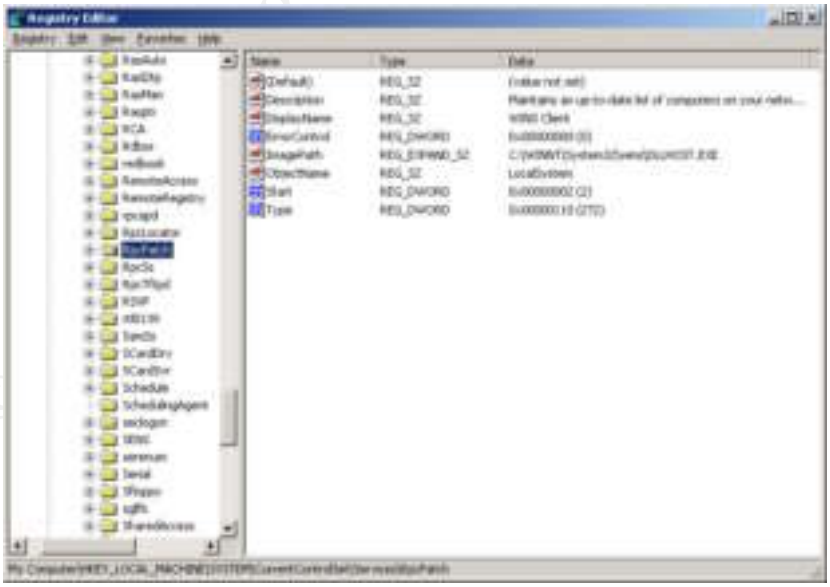

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.




- The registry key

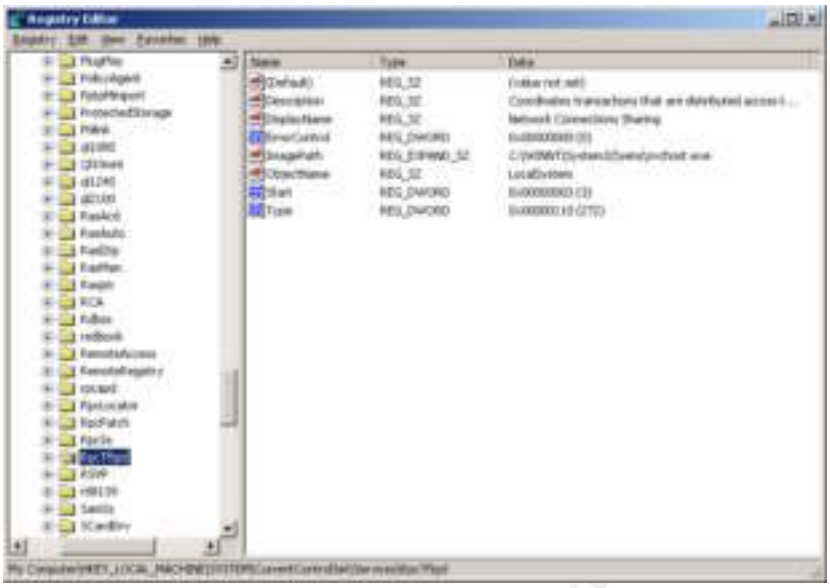
  ```
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
  ```

  Will contain the subkeys RpcPatch and RpcTftpd. Example is below.

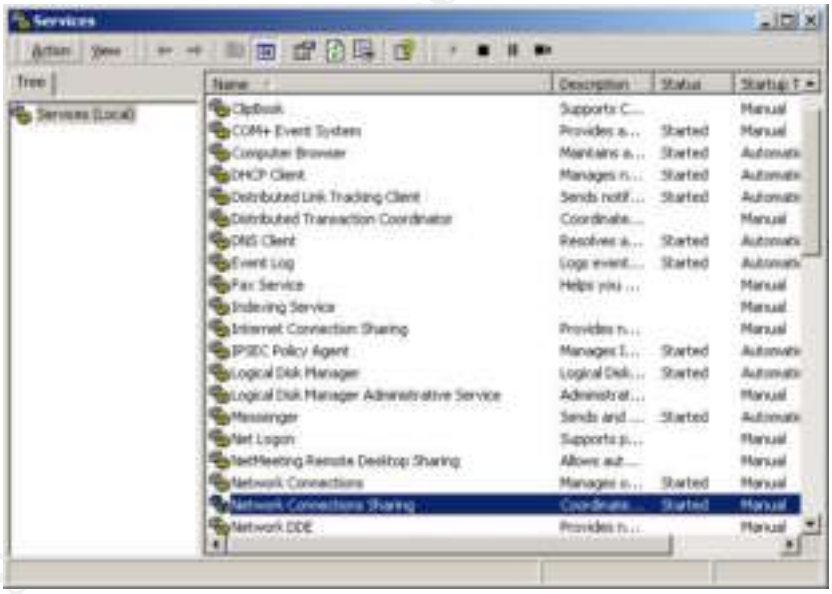





© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



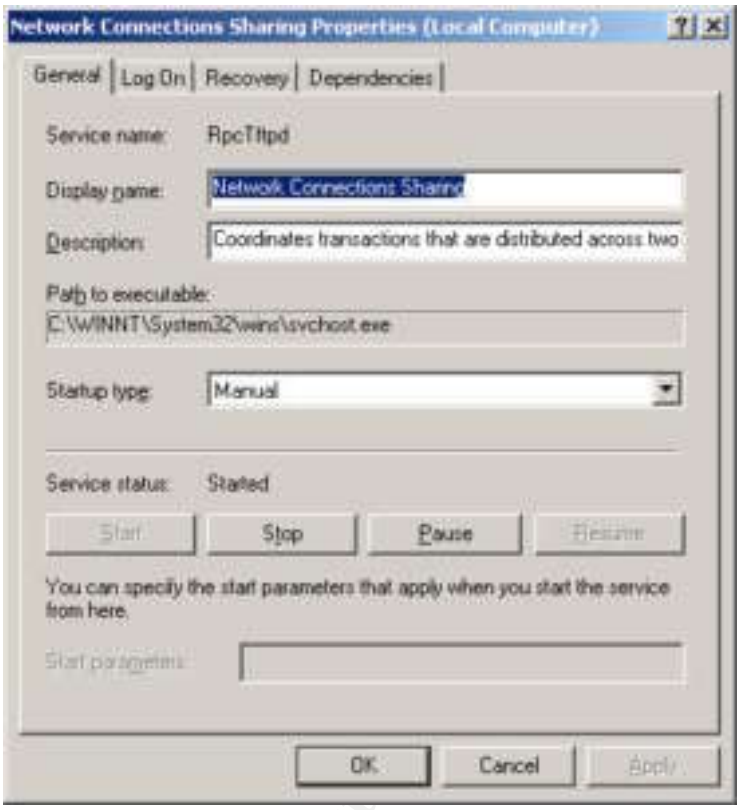
- In the Services of the Windows 2000 workstation (Start → Settings → Control Pannel → Administrative Tools → Services), you will notice an entry named “Network Connections Sharing.” Example is below.



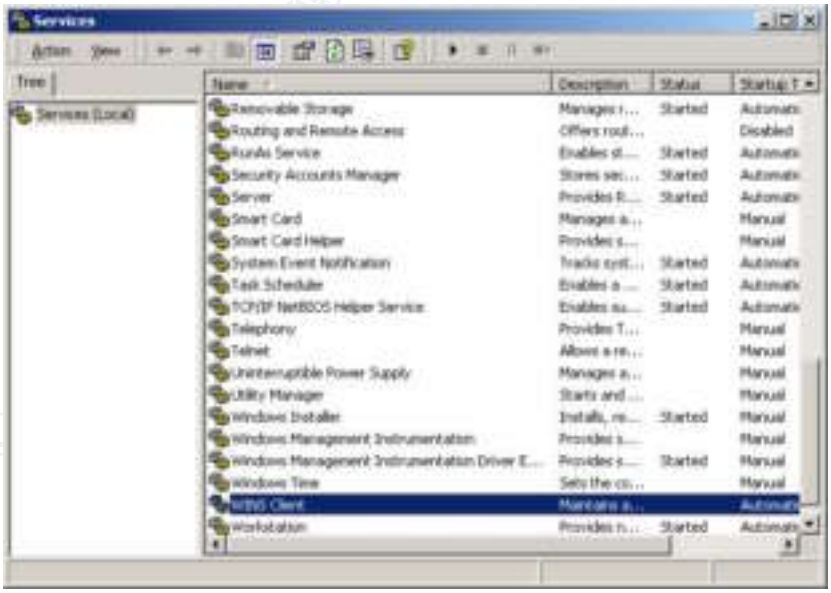
  - If you double-click on this entry, you will find that its service name is “RpcTftpd,” its display name is “Network Connections Sharing,” and it’s directory is “%WinDir%\wins\svchost.exe.” Example is below.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

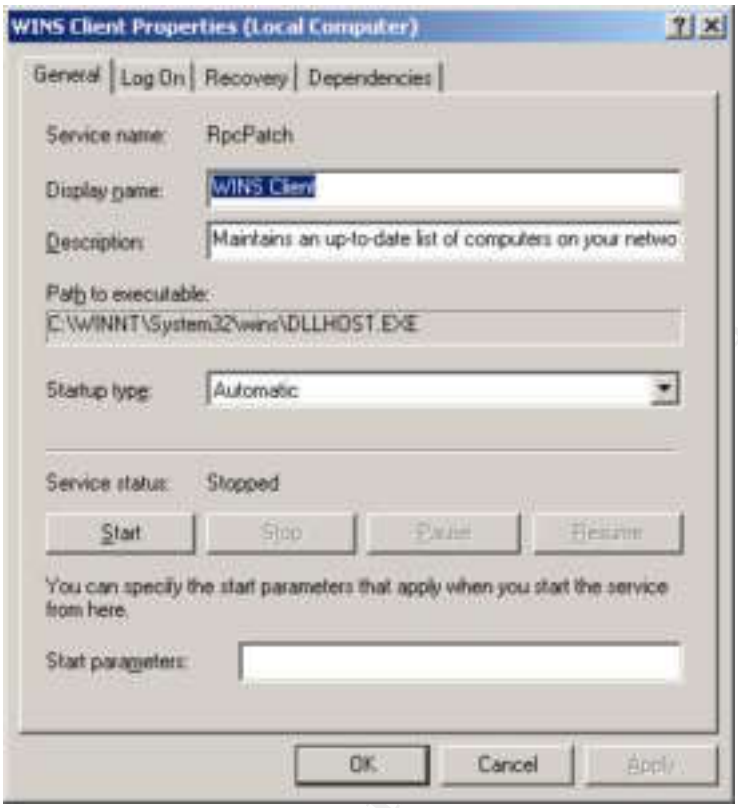



- In the Services of the Windows 2000 workstation, you will also notice an entry named “WINS Client.”

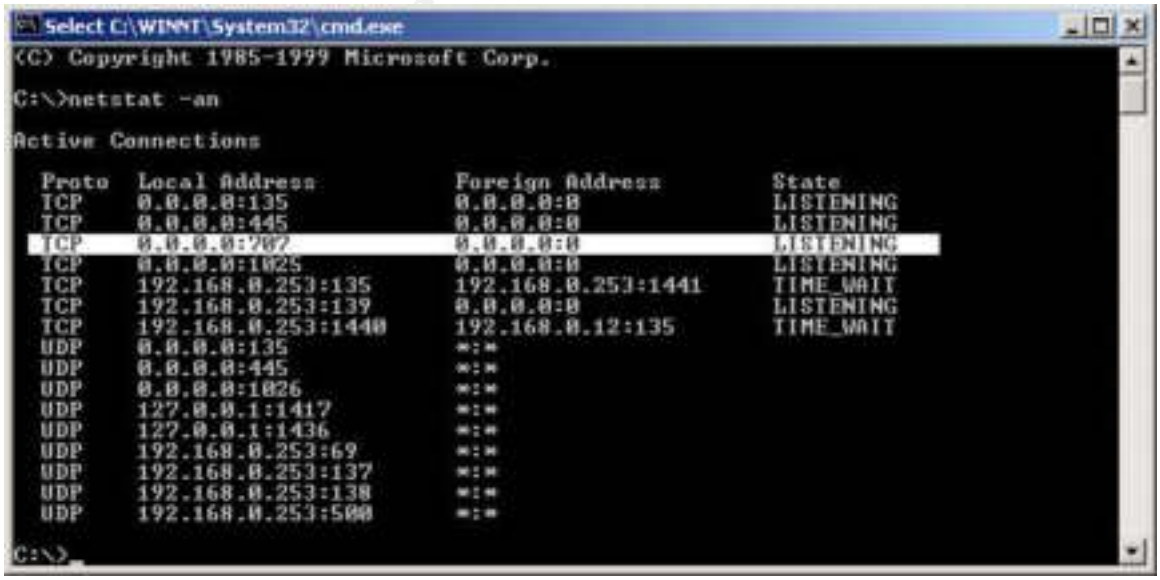



  - If you double-click on this entry, you will find that its service name is “RpcPatch,” its display name is “WINS Client,” and it’s directory is “%WinDir%\wins\dllhost.exe.”


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



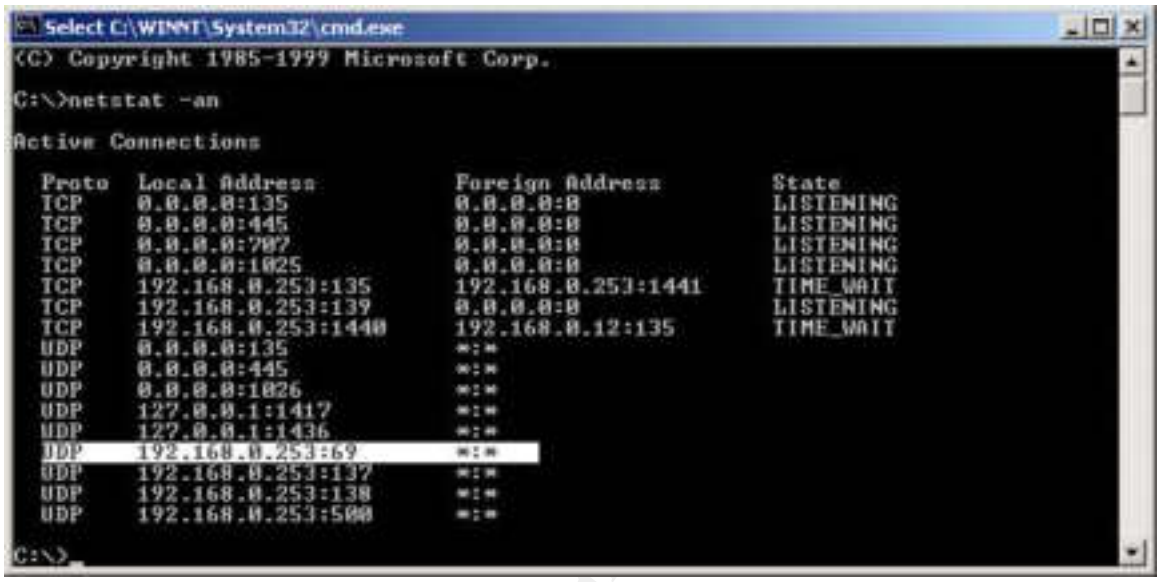

- The command “netstat –an” will reveal the machine listening on a random number TCP port between 666 and 765 (because of the method in which the Welchia code interacts with the Microsoft runtime dlls, this port is usually 707). The output will contain the following entry:





© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- The Command “netstat –an” will also reveal the machine listening on UDP port 69. This is the typical port associated with tftp, and is used by Welchia to transfer the malicious logic between Source and Victim machines. NOTE: Tftp is a legitimate service and not malicious in nature by itself.




© SANS Institute 2004, Au



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

## The Platforms/Environments

### The Victim

The victims of this attack were those workstations on the network that had not been patched with the MS03-026 patch. A typical installation involved Windows 2000 with Office XP as a baseline, the service pack levels ranged from SP2 to SP4.

### The Source

The Source of the infection was a laptop that had been out “in the wild” for an extended period of time whiles its owner was away on business. The machine had been infected with the Welchia worm. When this machine was put back on the LAN, Welchia went wild.

### The Network

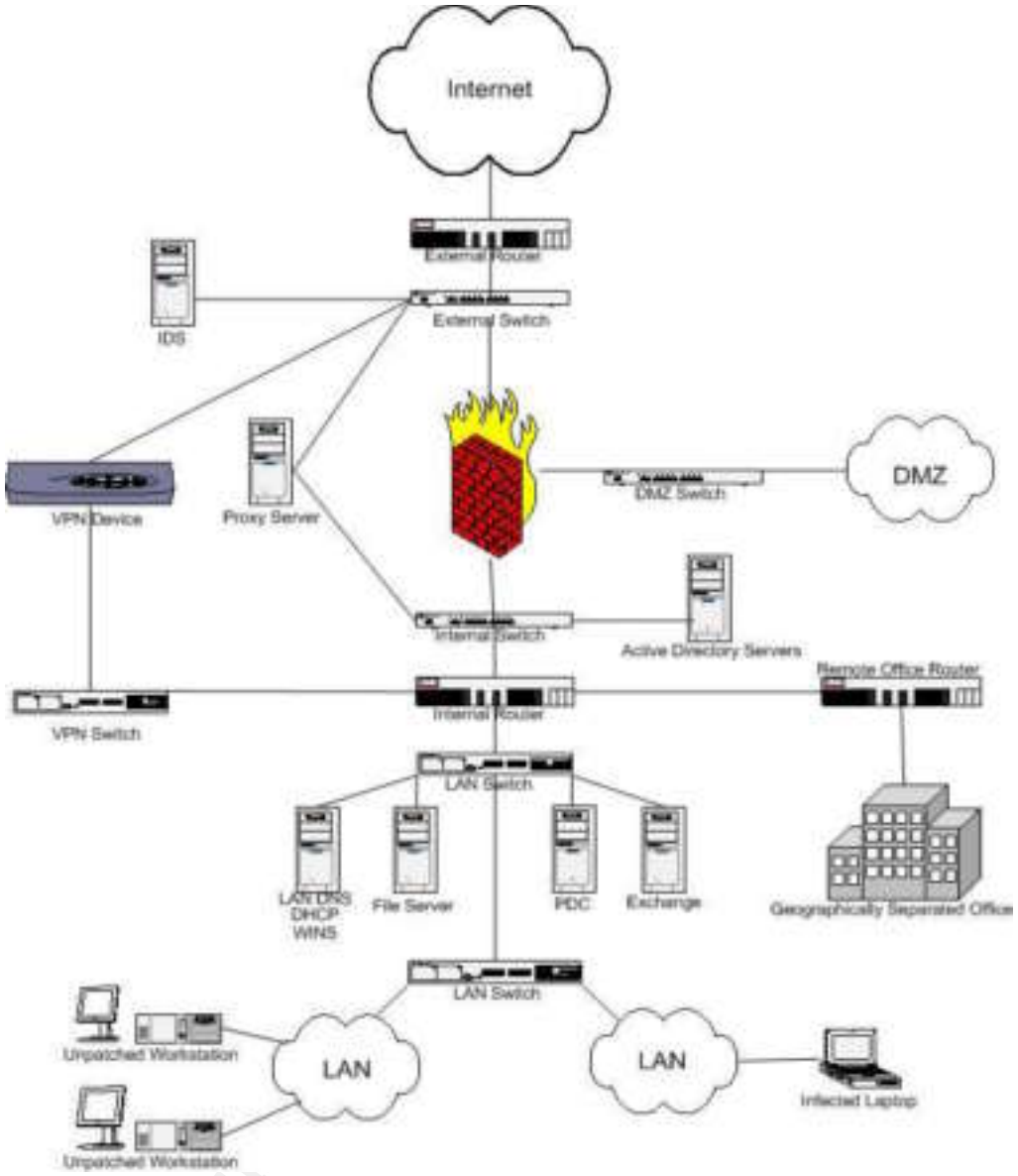
The network attacked was the Network Operations Center (NOC)/Tier 1 Help Desk area of the enterprise. The LAN at the NOC consists of approximately 20 personnel that include Network Engineers, Tier 1 Help Desk Personnel, and network management. Sharing the NOC’s firewall is a GSO of approximately 1000 users. The rest of the enterprise lives in multiple WANs beyond the firewall.

Below is an abbreviated diagram of the network:

© SANS Institute 2004, Author retains full rights.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.




## Stages of the Attack

### Reconnaissance

The Welchia worm's creator probably wrote Welchia in response to the mass MSBlaster infections sweeping the world. This would have alerted the author to not only the wide-spread use of the operating system, but also the fact that the vulnerability was largely unpatched. The only reconnaissance contained in the Welchia code itself is a test to ensure the worm is not being run in an enclosed lab. An attempt is made to resolve windowsupdate.com. If the attempt is unsuccessful, the worm will not start.


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

## Scanning

Scanning is integrated into the worm's code. Before attempted infection, the worm discovers the IP address of the host, then uses the Class-B boundary of that IP address and commences an ICMP scan from A.B.0.0 through A.B.255.255. If a host responds to the ICMP echo request, the worm commences an RPC DCOM attack against that host.

## Exploiting the System

Once a host responds to the worm's ICMP echo request, the worm sends RPC DCOM data to the host in an attempt to perform a buffer overflow against the host. Below is the ICMP exchange between the Source and the Victim:

```
22:01:49.061916 Source > Victim: icmp 72: echo request
22:01:49.062302 Victim > Source: icmp 72: echo reply
```

Next, the Source sends the Victim traffic on port 135 to exploit the RPC DCOM vulnerability.

```
22:01:49.061916 Source > Victim: icmp 72: echo request

22:01:49.062302 Victim > Source: icmp 72: echo reply

22:01:49.063087 Source.1174 > Victim.135: S
3871455588:3871455588(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)

22:01:49.063514 Victim.135 > Source.1174: S
4082656788:4082656788(0) ack 3871455589 win 17520 <mss
1460,nop,nop,sackOK> (DF)

22:01:49.063552 Source.1174 > Victim.135: . ack 1 win 17520 (DF)

22:01:49.063806 Source.1174 > Victim.135: P 1:73(72) ack 1 win
17520 (DF)

22:01:49.073330 Victim.135 > Source.1174: P 1:61(60) ack 73 win
17448 (DF)

22:01:49.073483 Source.1174 > Victim.135: . 73:1533(1460) ack 61
win 17460 (DF)

22:01:49.073517 Source.1174 > Victim.135: P 1533:1777(244) ack 61
win 17460 (DF)

22:01:49.075224 Victim.135 > Source.1174: . ack 1777 win 17520
(DF)
```

If the exploit is successful, the Source instructs the Victim to contact the Source on a TCP port between 666 and 765 (usually TCP port 707) to receive further instructions:



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
22:01:49.133140 Victim.1091 > Source.707: S
4082718584:4082718584(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)

22:01:49.133194 IP Source.707 > Victim.1091: S
3871525836:3871525836(0) ack 4082718585 win 17520 <mss
1460,nop,nop,sackOK> (DF)

22:01:49.133521 IP Victim.1091 > Source.707: . ack 1 win 17520
(DF)
```

Next, the worm launches a TFTP server on the Source, and instructs the Victim to download dllhost.exe from the Source. If %WinDir%\dllcache\tftpd.exe is not available on the Victim computer, the Source has the Victim download svchost.exe as well.

```
22:01:49.133583 Victim.3009 > Source.69: udp 20
```

Once the code is downloaded, the Victim executes the code and begins a network scan of its own, searching for other victims.

Lastly, the worm attempts to obtain the MS03-026 patch from Microsoft. If successful, the worm downloads and attempts to install the patch. After installation, the afflicted machine is rebooted.

According to Microsoft, because the worm does not properly assess the service pack level of the machine in question, it does not properly install the MS03-026 patch in all cases. This improper installation of the MS03-026 patch leaves the machine vulnerable to future infection and may even prohibit successful installation of the MS03-026 patch in the future.

## Keeping Access

As previously stated, any infected machines will have a random port open between TCP 666 and 765 to listen for further instructions, most frequently, this port will be TCP 707.

## Covering Tracks

This worm is not very effective at covering its tracks. It is loud, it leaves obvious files that are easily found and whose location is well documented, and it will clog the network with ICMP Echo Requests. However, this worm was designed to quickly identify and exploit any machines that had not already been patched with MS03-026. It did so on a wide scale and will continue to do so until the year 2004.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

# The Incident Handling Process

## Prevention

Just the week before the incident, our enterprise was alerted to the MSBlaster worm bringing down networks all over the world. Specific instructions were given to area network administrators to patch their workstations and servers with MS03-026, and to block all UDP ports 135, 137,138 and 445 as well as TCP ports 135, 139, 445, 593 and 4444 at the firewalls and external routers. Locally at the NOC, the network engineers in charge of the firewall and the routers blocked all ports as necessary. The mass patching of systems was a different story. While the GSO has a LAN Administrator for practically every segment, in the NOC all engineers have Administrator-level privileges on their respective workstations and laptops with the understanding that they would police their own systems. What nobody had provided for in this policy was the shared network assets such as the file-server and the servers installed on the network for the future migration to an Active Directory environment. The policy also did not provide for the notification of certain individuals who had been gone on business for extended periods of time. Finally there was no provision for dealing with those individuals who took their laptops home, connected them somehow to the internet (in violation of the security policy), then returned to work and connected those laptops back to the network.

In our office, there are government engineers and there are IT contractors. The contractors were brought in with literally two-day's notice to set up the NOC and get it running. Since the NOC is in charge of the entire enterprise, operations tempo was too time-consuming to and information assurance considered a low priority to come up with a written security policy with regard to patching systems. The government thought the IT contractors were doing it, and the IT contractors were told by certain government senior-level engineers not to touch certain systems.

As put in place, there was no standard for handling Malicious Logic incidents at an individual level save for restoring connectivity as quickly as possible and filing a Malicious Logic Report after the fact with the proper personnel. There were reporting responsibilities in place in the event of an actual compromise by a hacker.[17] There are backup communications and alternate control responsibilities already in place in case connectivity was lost for an extended period of time. However, there was no guidance for local incident handling procedures.

The incident handling team was constructed with personnel available at the time. These personnel consisted of myself (IT contractor information assurance engineer), the senior IT contractor engineer, several tier 1 help desk engineers,

[17] Incident response teams were deployed from headquarters.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

and a government engineer. I was the one who had been there the night before examining the firewall in an attempt to determine the Source of the problem. I thought I had contained the problem by removing the affected node from the network. I had the day off and had to be called back into work as the incident escalated. The senior IT contractor engineer knew the Windows operating system like the back of his hand. He also had knowledge of every node connected to the network, who it belonged to, and what its purpose was. The Tier1 Help Desk engineers were able to provide the manpower and skill necessary to address every machine on the network individually. The government engineer acted as the interface between the contractors and the government, and also provided extra manpower where necessary.

## Identification

The first indication of a problem manifested itself the evening before. The firewall began acting in an unusual manner. A 'top' command showed the access control list daemon consuming much higher resources than usual. Most of the NOC and GSO personnel had gone home for the day, so there should not have been such a large consumption of resources at this time. Fearing a DoS in progress, I examined the real-time denied traffic output. I found nothing unusual. I then did a "tcpdump –npi" on the external side of the firewall and once again found nothing unusual. Having ruled out a DoS, I thought perhaps a process had just gone crazy on the firewall, which is rare but does occur. I restarted all related services to no avail. I decided that since it was after regular business hours, I would reboot the firewall completely. Once the firewall came back up, everything appeared normal. At this point, I was 90% sure this was just a freak anomaly. It was 7 pm, and I had an engagement through 9 pm, so I decided I would call in at 9 to see how things were going.

At about 8:45, I got a call from the swing shift saying things were not going well. I came in and found the firewall in a stat similar to what it was in before I rebooted it earlier. This time, I ran a "tcpdump –npi" command on both sides of the firewall. At that point, the problem was obvious. The machine at address .77 was infected with the Welchia virus. Traffic was so prolific it was consuming more bandwidth than our What's Up Gold box. With limited access to network resources, tracking the box down proved difficult, but I found nmap useful in resolving hostnames on the local network when the "nslookup" command wasn't resolving addresses. I then had to track down its actual physical location. It turned out to be the laptop of one of the government senior engineers who had just returned from a business trip. He had permissions to ping through the firewall, which is why his traffic did not show up in the denied traffic logs of the firewall. I disconnected his system from the network, but could not clean his machine as I had no administrative access to it and the engineer had gone home for the evening. It was now about 12 am.


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Now we had a quagmire. I knew that this particular engineer's machine contained sensitive material relative to the contract, and that if it was even suspected that I altered that machine in any way, the entire contract would be in jeopardy. Also, the political embarrassment of having a worm infection present on the laptop of this particular engineer coupled with the problem of making the customer look good made this usually black and white decision very very grey. I called my project manager and told him I wanted to quarantine the laptop, but I asked his opinion of what actually should be done. It was decided that the laptop should remain disconnected from the network and the engineer personally notified when he arrived in the morning (i.e. no yellow-stickies). I examined the firewall logs for ICMP activity and it indicated that the .77 host was the only host that had been infected. Since it was only one machine that had been infected, and that machine was now off the network. It was not deemed necessary to wake anyone.

The only thing not discussed was who was going to notify the engineer. I was unaware that my project manager was taking the day off, and my project manager had forgotten that I was taking the day off. I thought about coming in anyway, but I re-examined the firewall logs for ICMP activity to prove to myself that the .77 host was the only host that had been infected. By all indications it was the only host infected, and since it was off the LAN, the situation was contained. I went home at about 2 AM.

The government engineer was quite annoyed when he discovered that someone had disconnected him from the network. Another senior-level government engineer was continuing his work setting up the Active Directory and put an entire server block online. I was called back in at 8 AM. When I examined traffic I realized that at least two additional machines had been infected; one of which was at the GSO. I informed the senior IT contractor engineer of the situation. By the time I notified management, the senior IT contractor engineer informed me that six additional machines had been infected. We were going down fast.

## Containment and Eradication

The first step we took in containing the situation was to attempt to stop the propagation by identifying infected hosts and removing them from the network. However, the worm was spreading too rapidly for this to be successful. The next option we exercised was to shut down all workstations and file servers in the NOC and disconnect the GSO to prevent further contamination. Then we burned a CD that contained all necessary implements to fix the afflicted workstations. This included the following:

Symantec's Welchia Fix tool (http://www.symantec.com/avcenter/FixWelch.exe)

Windows 2000 Service Pack 3



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The MS03-026 patch

Symantec's latest virus definitions file.

To obtain these implements, we moved a fully updated workstation to the external switch, assigned it an IP address, and downloaded the necessary tools. Next, we burned the tools to CD, printed off some instructions on how to use the individual tools, and deployed the CDs to the LAN and to the GSO with the following instructions:

1.) All tools are to be run against every server or workstation on the LAN, regardless of known infection.

2.) The tools are to be run in the following order:

- Welchia Fix Tool
- Windows 2000 Service Pack 3
- The MS03-026 Patch
- Symantec's latest virus definitions
- Welchia Fix tool again
- Do a full system virus scan

The Welchia Fix tool is not fully successful unless it is run against a system after that system is booted into safe mode. To do this, restart the machine and hit the F8 key during Windows 2000 initialization.

3.) If the Welchia Fix tool finds no instances of the worm on the first pass:

- Install Service Pack 3
- Install the MS03-026 Patch
- Load the virus definitions
- Perform a full system virus scan.

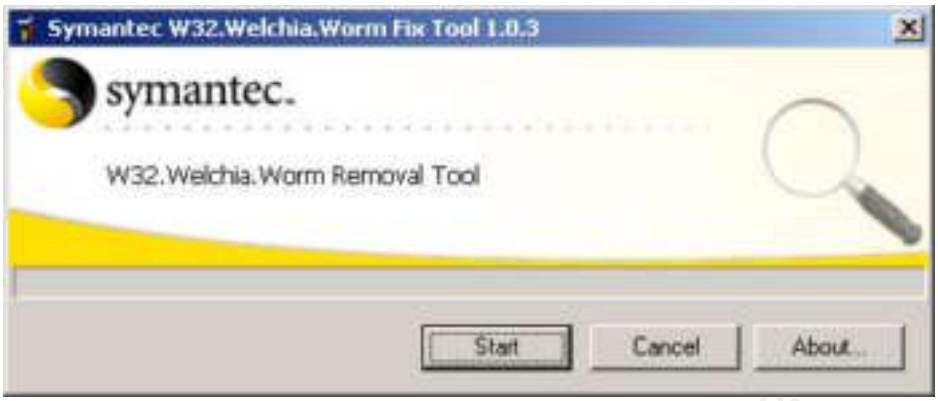
The Fix tool does NOT need to be run a second time.

3.) After all steps are taken, the machine may be returned to the network.

When Symantec's Welchia fix tool is executed, the following screen appears:

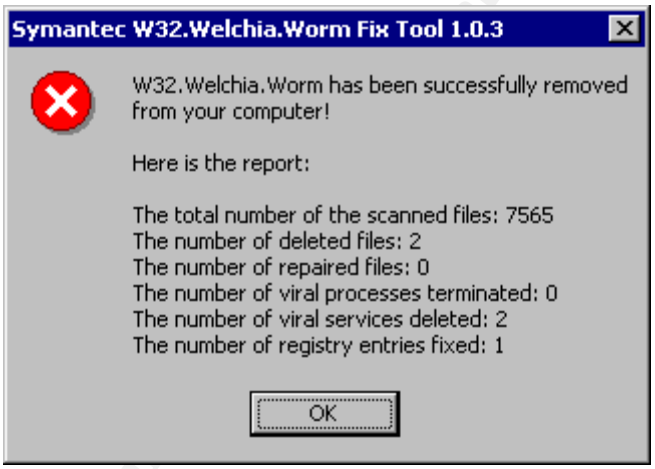


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



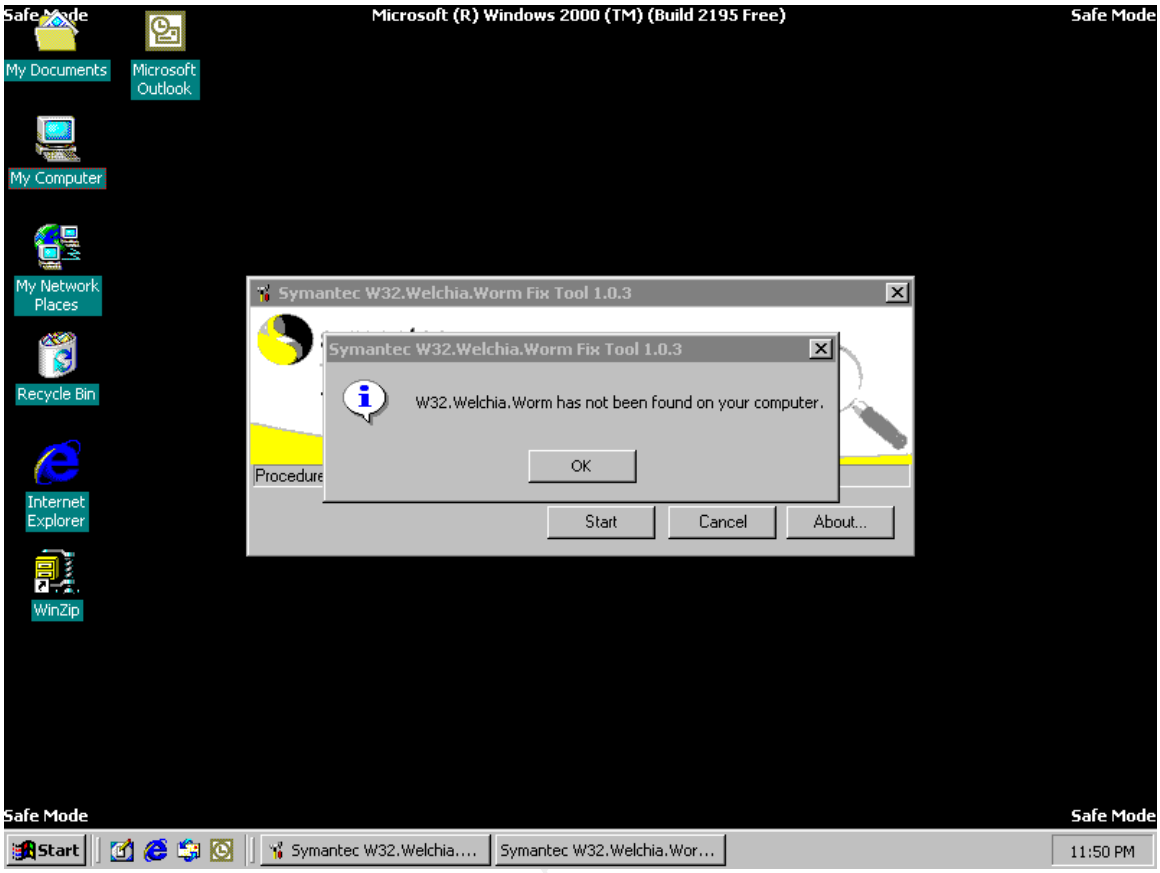

Simply hit “Start” and let the tool run.

After a successful eradication of the worm, the following report is displayed on the screen:




Just to make sure everything is gone (of course I’m paranoid!), the Welchia Fix Tool should be run again. When that happens, the following message should be displayed on your screen:


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.




Congratulations!!! Your box is clean.

## Recovery

Once all workstations and servers at the NOC were hardened against the threat, connectivity was restored to the GSO. The NOC experienced approximately 4 hours of downtime due to the infection. The GSO experienced approximately 10 hours of downtime.

## Lessons Learned

A meeting was held by the government engineers to determine what happened and how to prevent it happening in the future. Several key areas were addressed:

1.) Why was the laptop infected and how did it get on the network?

The laptop was infected because it did not have service pack 3 installed when the engineer installed the MS03-026 patch. On Microsoft's MS03-26 patch download page it clearly states that workstations had to be at Service Pack 3 or later to correctly install the patch. However, the engineer downloaded the patch from a government site that did not contain that information. This information


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

was relayed to the proprietors of the site who have included information on the assumed baseline of the system to be patched.

Why wasn't the patch installed even though it was reported to have been installed?

It was also discovered that there was confusion in the guidance pertaining to the MS03-026 patch. Certain LAN managers were under the impression that blocking UDP ports 135, 137, 138 and 445 and TCP ports 135, 139, 445, 593 and 4444 – which was only done at the network boundary and not on internal routers – represented a successful mitigation of the threat. The security policy states that a successful mitigation of the vulnerability was equivalent to applying the appropriate patch. This aspect of the policy was implemented to account for several home-grown applications that were often broken by the application of certain security patches. Due to that aspect of the security policy and the misinformation surrounding the MS03-026 patch, certain system administrators reported compliance with the patch implementation directive, but had never applied the patch.

With regard to service pack level, a check of service pack was never incorporated into the patch installation procedures that were available from the government website. Most administrators obtained the patch from the same website as the government engineer. Therefore, some administrators who actually did apply the patch failed to realize the vulnerable systems needed to be at least at Service Pack 3 for the patch to be successful.

Why were some systems not equipped with the proper antivirus suite?

This is the sole responsibility of the network administrators responsible for those particular segments of the network. The security policy specifically states that all nodes deployed on the network will have an antivirus suite installed and that virus definitions will be frequently updated. There is no excuse for the lack of an antivirus suite.

Who was responsible to ensure shared network resources are patched?

It was decided that in the future, all network resources that can conceivably be patched by the IT Contractors will be patched by the IT contractors. To ensure a greater level of control of these assets, only select individuals were given administrator-level passwords to the shared network resources. In the case of network assets that were controlled by government engineers who were not willing to allow IT contractors access to that system, those government engineers would be held responsible for patching all systems under their control. Those systems were documented and were to be disconnected from the network when not in use.


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

There were many things done incorrectly in this scenario. The most easily corrected was the assignment of responsibility for patching network assets to the IT Contractors. In addition, the IT contractors received permission to remove any and every workstation and server from the network should an incident such as this occur in the future. There was an investigation into the violation of policy that occurred when machines were placed on the network without an antivirus suite applied to the machine. This investigation resulted in a machine by machine hands-on investigation to ensure every machine connected to the network had a valid antivirus suite installed and was receiving proper updates from the antivirus server.

Finally, those engineers with responsibility for their own laptops were given training on methods and practices to keep their laptops up do date when away on business. The policy governing individual control of workstations was reviewed, but not changed.

## Conclusion

While the idea of a good worm is certainly a worth while venture to be explored (and could be quite profitable to the programmer that develops it), any such venture would have to have security restrictions to ensure it does not leave the intended network. Welchia demonstrates what happens when these little monsters escape. At the very least, the lessons learned from dealing with Welchia brought our organization to a greater understanding of what is necessary to properly secure our network. Perhaps then the real benefit of Welchia was not the patching of unpatched systems, but rather the revelation of flaws in the security policy as it was understood. While we would like to believe our network is secure against further attack, common knowledge tells us that the false sense of security is worse than not being secure. At the very least, with 2004 approaching rapidly, Welchia will soon be a distant, bad memory.

© SANS Institute 2004, Author retains full rights.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

## Extras

The following is a description of the Welchia Worm by Trend Micro:

**Installation and Autostart Technique**

Upon execution, this worm drops a copy of itself as DLLHOST.EXE in the %System%\wins directory. Then, it creates a service for this dropped copy with a display name, WINS Client.

(Note: There is a normal system file named DLLHOST.EXE that is 6 kilobytes.)

(Note: %System% refers to the Windows system folder, usually C:\Windows\System on Windows 95, 98 and ME, C:\WINNT\System32 on Windows 2000 and NT, and C:\Windows\System32 on Windows XP.)

It also copies the file TFTPD.EXE to the %System%\Wins folder as SVCHOST.EXE and then creates a service for it with the display name "Network Connections Sharing".

TFTPD.EXE or SVCHOST.EXE is a TFTP (Trivial File Transfer Protocol) server that is used by this worm to set the affected system as a download site for its copy. This worm is then able to propagate by instructing remote systems into downloading it using TFTP.

Creating the described services allows this worm and the TFTP server to execute every time Windows starts.

It listens to a randomly selected port, ranging from port 666 to port 765, on the affected system. This port is where remote vulnerable systems connect to and receive commands as part of this worm's propagation routine.

To prevent having multiple copies of itself in memory, this worm creates the mutex "RpcPatch_Mutex".

**Propagation**

To propagate, this worm sucessively uses all of the following algorithms to look for hosts to infect:

- Algorithm #1
  It scans for hosts from the affected host's subnet. For instance, if the IP address of the affected host is A.B.C.D it will scan for A.B.0.0 to A.B.255.255.
- Algorithm #2
  It scans for three class B networks near the affected host's network. Thus, if the IP address of the affected host is A.B.C.D, it starts its scan for three class B networks starting from A.(B+1).0.0 or A.(B-3).0.0. It increments the last two octets from 0 to 255 and then increments the second octet until three class B networks are scanned.
- Algorithm #3
  It scans for one class B network, which is randomly chosen from 75 predefined network addresses. For example, it scans for A.B.0.0 to A.B.255.255 where A.B is chosen from predefined list. Possible values of A are: 61, 220, 202, 203, 210, 211, 218, 219 and 220.


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- Algorithm #4
  It generates 65,535 random IP address to scan. However, the first octet is always one of the following: 61, 202, 203, 210, 211, 218, 219 and 220.

It checks for running hosts by first sending an ICMP ECHO request packet.

Similar to the earlier MSBLAST variants, this worm also exploits the RPC DCOM Buffer Overflow, and instructs target systems to download the worm copy from the affected system using TFTP. The RPC DCOM Buffer Overflow exploit is initiated via port 135 of target systems.

Additionally, it also uses a WebDAV exploit in order to propagate to vulnerable systems. For detailed information about the said exploit, please refer to the following Microsoft Web page:

http://support.microsoft.com/default.aspx?kbid=815021

Using these exploits, it sends a shell code to a vulnerable system, which in turn will execute a remote shell on the target system. The remote shell connects to a random selected port between port 666 to port 765 of the infected host where it receives commands to download the worm copy via TFTP.

**Automatic Malware Removal**

This worm removes itself from infected systems when the current year is 2004. When it finds that the current year is 2004, it removes the created services and deletes itself.

This malware may also terminate and delete other MSBLAST variants. It does this by looking for the process MSBLAST and then terminating this process. It then deletes the file, MSBLAST.EXE, from the Windows system folder.

**Downloading Patches**

This malware also patches the system against the RPC DCOM Buffer Overflow Exploit. First, it checks the operating system version and locale information. Then, it downloads the appropriate patch from the designated Microsoft Web site. After executing the said patch, it reboots the system.

Note that this worm does not check the system for the required service pack necessary to install the patch.*

Some of the patches it downloads into the system are as follows:

- http://download.microsoft.com/download/
  6/9/5/6957d785-fb7a-4ac9
  -b1e6-cb99b62f9f2a/Windows2000-KB823980-x86-KOR.exe
- http://download.microsoft.com/download/5/8/f/
  58fa7161-8db3-4af4-b576-
  0a56b0a9d8e6/Windows2000-KB823980-x86-CHT.exe
- http://download.microsoft.com/download/2/8/1/
  281c0df6-772b-42b0-9125-6858b759e977/
  Windows2000-KB823980-x86-CHS.exe



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- http://download.microsoft.com/download/0/1/f/01fdd40f-efc5-433d-8ad2-b4b9d42049d5/Windows2000-KB823980-x86-ENU.exe
- http://download.microsoft.com/download/e/3/1/e31b9d29-f650-4078-8a76-3e81eb4554f6/WindowsXP-KB823980-x86-KOR.exe
- http://download.microsoft.com/download/2/3/6/236eaaa3-380b-4507-9ac2-6cec324b3ce8/WindowsXP-KB823980-x86-CHT.exe
- http://download.microsoft.com/download/a/a/5/aa56d061-3a38-44af-8d48-85e42de9d2c0/WindowsXP-KB823980-x86-CHS.exe
- http://download.microsoft.com/download/9/8/b/98bcfad8-afbc-458f-aaee-b7a52a983f01/WindowsXP-KB823980-x86-ENU.exe

The downloaded patch has the file name, RpcServicePack.exe. This worm deletes this file after it is run.

Before downloading or installing the patch on the system, this worm first checks if the system has been previously patched by checking for the following registry keys:

```
HKEY_LOCAL_MACHINE\SOFTTWARE\Microsoft\
Updates\Windows 2000\SP5\KB823980

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Updates\Windows XP\SP1\KB823980

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Updates\Windows XP\SP2\KB823980
```

**Other Details**

The following strings are visible in this worm's body:

```
========== I love my wife & baby :-)~~~ Welcome Chian~~~ Notice: 2004
will remove myself:-)~~ sorry zhongli~~~========== wins
```

## Related Websites:

Symantec:
http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html

Sophos:
http://www.sophos.com/virusinfo/analyses/w32nachia.html

Trend Micro:
http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_MSBLAST.D

McAfee: http://us.mcafee.com/virusInfo/default.asp?id=description&virus_k=100559


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

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

The Analysis of Roman Madina's WebDAV Exploit:
http://www.fatelabs.com/library/fatelabs-ntdll-analysis.pdf

Smashing the Stack for Fun and Profit:
http://www.phrack.org/show.php?p=49&a=14

Source Code for RPC DCOM vulnerability:
http://www.metasploit.com/tools/dcom.c

Source Code for the WebDAV Exploit:
http://www.rs-labs.com/exploitsntools/rs_iis.c

Download Welchia (this is the actual worm, be very careful):
http://r0eins.dvhosting.de/Alexander/Nachi.zip

© SANS Institute 2004, Author retains full rights.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

## References


Aelph One. “Smashing the Stack for Fun and Profit.” Phrack Magazine, Volume 17, Issue 49, Article 14. 8 November 2003. URL: http://www.phrack.org/show.php?p=49&a=14. (7 Dec 2003).

Common Vulnerabilities and Exposures. “CAN-2003-0109.” 3 December 2003. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109 (3 Dec. 2003).

Common Vulnerabilities and Exposures. “CAN-2003-0352.” 3 December 2003. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109 (3 Dec. 2003).

Day, Greg and Fisher, Lee. “McAfee Best Practices Advisory for W32/Nachi.worm.” URL: http://www.mcafee.dk/support/W32Nachi_Worm_Advisory.pdf (3 Dec 2003).

Entegrity. “Overview – RPC Fundamentals.” URL: http://support.entegrity.com/private/doclib/docs/osfhtm/develop/appdev/Appde262.htm (4 Dec 2003).

Flashsky. “The Analysis of LSD's Buffer Overrun in Windows RPC Interface” 25 July 2003. URL: http://www.xfocus.org/documents/200307/2.html (7 Dec 2003).

Hines, Eric. “Analysis of the ntdll.dll WebDAV Exploit.” Fate Research Labs Internet Warfare and Intelligence Team. 25 March 2003. URL: http://www.fatelabs.com/library/fatelabs-ntdll-analysis.pdf (7 Dec 2003).

Medina, Roman. “rs_iis.c” 26 March 2003. URL: http://www.rs-labs.com/exploitsntools/rs_iis.c (7 Dec 2003).

Microsoft Corporation. “DCOM Technical Overview.” November 1996. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp (4 Dec 2003).

Microsoft Corporation. “Microsoft Component Services: Server Operating System, a Server Overview.” 15 August 1998. URL: http://www.microsoft.com/com/wpaper/compsvcs.asp (4 Dec 2003).

Microsoft Corporation. “The RPC Model.” February 2003. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rpc/rpc/how_rpc_works.asp (4 Dec 2003).

Microsoft Press. “Microsoft Security Bulletin MS03-026; Buffer Overrun In RPC Interface Could Allow Code Execution (823980).” 10 September 2003. URL:




© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp (3 Dec. 2003).

Microsoft Press. “MS03-007: Unchecked Buffer in Windows Component May Cause Web Server Compromise” 30 October 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;815021 (3 Dec 2003).

Microsoft Press. “Virus Alert About the Nachi Worm.” 17 September 2003. URL: http://support.microsoft.com/default.aspx?kbid=826234 (3 Dec 2003).

Nahorney, Benjamin; Knowles, Douglas; and Perriot, Frederick. “W32.Welchia.Worm.” 29 November 2003. URL:http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html (3 Dec 2003)

Open Group, The. “The Component Object Model.” The Active X Core Technology Reference. Copyright 1999. URL: http://www.opengroup.org/onlinepubs/009899899/toc.htm (7 Dec 2003).

Open Group, The. “Remote Procedure Call Model” DCE 1.1: Remote Procedure Call, Part 4. Copyright 1997. URL: http://www.opengroup.org/onlinepubs/9629399/chap6.htm#tagcjh_11_03_01 (4 Dec 2003).

Rapoza, Jim. “Up with Good Worms.” Eweek Enterprise News and Reviews. 21 April 2003. URL: http://www.eweek.com/article2/0,3959,1037004,00.asp (3 Dec. 2003).

Stickler, Dustin. “Re: Purging Blaster.worm.” 14 August 2003. URL: http://www.securityfocus.com/archive/105/333267/2003-08-09/2003-08-15/2 (3 Dec 2003).

Stuart. “Re: Purging Blaster.worm.” 13 August 2003. URL: http://www.securityfocus.com/archive/105/333135/2003-08-09/2003-08-15/2 (3 Dec 2003).

Whitehead, E. James, Jr. “World Wide Web Distributed Authroing and Versioning (WebDAV): An Introduction.” 5 February 1998. URL: http://ftp.ics.uci.edu/pub/ietf/webdav/intro/dav_acm_sv.pdf (7 Dec 2003).

Whitehead, E. James, Jr. “World Wide Web Distributed Authroing and Versioning (WebDAV): An Introduction.” 5 February 1998. URL: http://ftp.ics.uci.edu/pub/ietf/webdav/intro/dav_acm_sv.pdf (7 Dec 2003).

**THANKS:**



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Eric Piegols
Jodi Sandvick
Ronald Blankebneckler
Eugene Bransfield Sr.
Jodi Sandvick
Larry Halliburton

© SANS Institute 2004, Author retains full rights.



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.