

# Windows Authentication Schemes

By Mark Minasi

Anyone reading Microsoft security PR stuff would be led to believe that running a modern NT/2000/XP/2003 system would reap the benefits of their newer, more secure operating system software. And that's true in the sense that their newest authentication systems, things called NTLMv2 and Kerberos, are indeed quite good. But there's an old security hole that's existed in NT since its earliest days that is easily exploited (which is bad) but easily closed for free (which is good).

In other words, having a state-of-the-art Active Directory or even an NT 4-based domain can be very secure but for one thing: there's some software hanging around from the late 1980s that every single Microsoft system runs just on the off-chance that you're still running a LAN Manager 2.1 server, and that software (called "LM authentication") can be used to steal passwords on your system. In this article, I want to explain basically how Microsoft systems do authentications, how this older software creates a security hole, and how to close that hole. I also want to advocate strongly getting rid of something called "NTLM" in favor of "NTLMv2." So I've got a lot of explaining to do, but trust me, it'll be worthwhile.

Over the years, Microsoft has created a variety of protocols designed to allow people to authenticate to network servers securely. First there was LM, the authentication system used by LAN Manager, an OS/2-based network operating system available in the 1989-1994 time frame. Windows for Workgroups, Windows 9x and ME systems still use LM. Then 1993 saw an improvement in LM with the introduction of NT 3.1 through a new authentication system called NTLM.

By default all NT, 2000, XP, and 2003 systems use NTLM to log onto workgroup members or NT 4 domains. NT 4 systems also use NTLM to log onto Active Directory domains. NT 4.0 Service Pack 4 introduced more improvements to NTLM, resulting in NTLMv2. I don't know of any operating systems that use NTLMv2 by default but, as you'll see, I hope you'll configure your systems to use it.

Finally, 2000, XP and 2003 systems can log onto Active Directory domains using Kerberos, which is an even more secure system than NTLMv2. There is currently no way to make a Windows 9x, ME, or NT 4 system log onto an Active Directory using Kerberos. In short, then, we've got four authentication systems to work with — LM, NTLM, NTLMv2, and Kerberos. I'll talk about the first three in this article.

## Challenges: Sending Passwords Without Transmitting Them

No matter how much talk we hear about using fingerprints, retinas, smartcards or other high-tech methods to control access to network resources, 99.9 percent of us still control access to our servers with the same old simple approach: passwords. You claim to be someone, and prove it by showing that you know something that only that person knows. Basically, then, every time you try to access a server, the server needs to verify that you are indeed who you say you are. In the Microsoft world (and others also, but we're talking the Windows and NT families here) the server "challenges" your client (probably a workstation), in effect saying to the client "tell me the name and password of the person sitting at you so I can decide whether or not to grant the access that you want."

But how to exchange this password information? At first glance, the simplest way to do this would be for the client to simply say something like "I'm working on behalf of a guy named Mark whose password is 'swordfish.'" But you probably know that'd be a bad idea, as anyone on a network can easily run a "sniffer" that shows every byte transferred across the network. If the server simply asked the workstation, "please send me your user's password," then anyone could sniff the network and see that information. So, instead, servers and clients check passwords by sending data back and forth to prove that they use the same password, but without actually *sending* the password. Let's take a minute and see how that works, and how it might be compromised.

# Windows Authentication Schemes

By Mark Minasi

Vastly oversimplified, here's how a challenge authentication works. The client says, "please give me access for Mark." The server says "hmmm, Mark's password is 'swordfish.' But I can't ask the workstation to send 'swordfish' over the wire. So I'll convert 'swordfish' to a number." This is simple to do, as you may recall that computers actually store everything as numbers — strings of ones and zeros — anyway. Let's say that 'swordfish' works out to equal 143 in numeric form. Again, it wouldn't — it'd be much larger — but it's easier to follow the example with simple numbers. The server then asks the client, "take Mark's password and divide it by 7, then send me only the remainder."

The workstation then says to itself, "well, I know that the password's 'swordfish,' and if I convert that to a number, I get 143. Divide 143 by 7 and you get 20 with 3 left over." So it sends the server the answer "3." Of course, the server can do its own calculation and gets the same number. The fact that the server computes the same remainder as the client leads the server to believe that the client has the same password for Mark's account as the server, and the challenge has been met, so to speak. When that happens, the server makes a little note to itself that says something along the lines of "I believe that I'm talking to Mark through such-and-such client," so that the client needn't re-authenticate between each byte transmitted from server to client. That little note to itself is called an "access token."

## Attacking a Challenge-Based Authentication

So far, so good; we've proven that the client has the same password as the server for Mark's account. Again, this was a vastly oversimplified explanation of a challenge, but I promise I'll fill in the blanks as we go. Let's next see how some scum-sucking dirtbag might try to steal a password even if we used a challenge. He'd put a sniffer on your network and see this:

```
Client: Let me in.  
Server: 7.  
Client: 3.  
Server: Okay, you're in.
```

Is that enough information for Joe Dirtbag to figure out your password? It might be, or it might not be.

## Strengthening Challenge-Based Authentication With Random Challenges

First of all, what if the server always responded "7" when issuing a challenge? That'd be troublesome and very helpful to the dirtbag. In that case, all Joe Dirtbag has to do is to just sniff a logon and record the messages that Mark's client sends the server -- the "Let me in" and "3" responses in my simple example. If we can assume that the server's challenge will always be "7," then our recorded answer of "3" will always cause a successful login.

Such an attack is called a "replay attack." Notice that Joey D still doesn't know Mark's password. But he *does* know how to log on as Mark, which is just as good.

He could even take it further and try to figure out Mark's password in a "server always challenges with '7'" world. He could change *his* password — we're assuming that he's got some kind of account on the system if he's close enough to sniff the network — to some new value and then try a logon, sniffing his own logon. If he ends up with a response whose value equals 3, then he knows that whatever he set his own password to equals Mark's password. You could imagine Joe writing a program that loops through a list of every word in the dictionary, first changing Joe's password to that

# Windows Authentication Schemes

By Mark Minasi

word, logging Joe on and sniffing the response to the '7' challenge and storing it. He'd then have a nice handy-dandy way to reverse-engineer any response to a challenge into the password that led to that response, thereby knowing the user's password.

There are two defenses built into Microsoft's authentication protocols that avoid this. First, you can set up domains, whether NT 4 or Active Directory, to only keep users from changing their passwords more than once a day. Second, Microsoft authentication algorithms have always issued *random* challenge values.

(Time for a definition. What I've been calling a "challenge value" — that is, the "7" — is called the *nonce*, the *challenge*, or the *challenge token* by different sources. )

How is that possible? Well, first of all, nonces are pretty big, although it's not clear how big. Some sources say that NTLM employs 64-bit nonces; others say 128-bit. (I apologize for being fuzzy on this and other details about NTLM authentication, but apparently Microsoft hasn't completely documented the process and so different sources quote different values.) A 64-bit nonce can have any one of 18,446,744,073,709,551,616 (that's over 18 sextillion) values, a 128-bit nonce can have any one of about 340,000,000,000,000,000,000,000,000,000,000,000,000,000 values (and please don't ask me what the word for *that* number would be, I haven't a clue).

Microsoft even offers a third protection via NTLMv2, which issues nonces that are not only random, they're unique; that is, once a server sends a nonce with a given value, it'll never send a nonce with that value again. Replay attacks are, then, completely impossible in NTLMv2.

## Strengthening Challenge-Based Authentication With Better Hash Algorithms

So we've seen that random, non-repeating nonces make the chances of pre-cracking passwords based on pre-computed responses to particular nonces and replay attacks pretty bleak. Assuming constantly-changing nonces, how could Joe Dirtbag next try to crack our passwords?

Recall my example where the server says just "7" and the client responds "3." That's all that we'd see if sniffing the communication -- "Let me in," followed by "7," followed by "3," followed by some kind of welcome message. Notice that the server never actually says what to *do* with the 7; the algorithm that says "take the password, make it a number, and divide it by the number that I send you and send me back the remainder" is built into the operating system — actually in a program called *lsass.exe*.

It would seem, then, that not telling the world what that client and server are actually *doing* with the two number, not explaining the division and remainder stuff, is something of a defense against hackers. But is it a good defense?

In the long run, probably not. Security types call the approach of not revealing the methods whereby you secure a network "security by obscurity." To an extent, we all do security by obscurity in the sense that no companies that I know of post maps explaining exactly how they secure their networks.

Making the bad guys figure out what kind of firewall you've got, what operating system you run and the like at least slows them down. But it's not great security in the authentication area for two reasons. First, eventually someone just disassembles the code, figures out the algorithm and tells the world. And, second, the problem with using a secret challenge algorithm is that because only a small number of people know about it, then there might be a flaw in the algorithm that might make reversing it — "backwards-computing" the password from a challenge and a response — really easy, even if it

# Windows Authentication Schemes

By Mark Minasi

wasn't obvious to the small number of people who knew the algorithm. A better approach would be for a whole lot of smart people to figure out algorithms that are really hard to "backwards compute."

Before I go any further, time for another definition. "Message digest" or "hash" are synonymous phrases for what you get when you put one or more numbers into some mathematical function.

We've already met a hash, even though I've not called it that. Recall that our challenge algorithm takes two numbers, the account password and the nonce, and produces a response. Stuffing a nonce of 7 and a password with numeric value of 143 into my "hash function" — the fancy name for dividing the password by the nonce and taking the remainder — results in a hash or message digest of 3. Notice the fact that this isn't trivially reversible; all you know is that the system fed a "7" and an unknown password into some hash function and got back a digest of "3." It's not immediately obvious how you'd figure out that the number stuffed into the hash function with the "7" was "143."

Now let's return to the question of whether or not to reveal our network's hash function to the world in general. A hash function could be any mathematical formula that takes one or more numbers as inputs and produces one number as an output. For example, I could have built a hash function that says, "add the nonce and the password to produce the hash." That'd be a pretty lousy hash function, however, as anyone who sniffed an authentication would could see first the nonce (7) and then the hash response (150).

Subtracting the nonce from the hash would yield the password of 143 — not good. Reiterating what I said before about leaving the algorithm design to a publicly-discussed process conducted by mathematical geniuses has worked pretty well and produced a few commonly-used hash functions.

One such function is called the Data Encryption Standard or DES. DES encrypts a message, and takes two inputs: a 64-bit message, and a 56-bit key. DES then uses the 56 bit key to encrypt the 64-bit message. (If you want to encrypt a message larger than 64 bits, DES just chops it up into 64-bit pieces and encrypts them one at a time. Recall that any text message in a computer can be represented as some large number.)

DES has been around for a while, though, and at this point common computers are powerful enough to crack DES-encrypted messages, so there's a variation called triple DES, which as you'd imagine encrypts the message, then encrypts it again, then encrypts once more, making it harder to crack. DES is designed to be an encryption method that allows decryption; if I knew what key was used to encrypt the message in the first place, then I could use that key to decrypt the message. According to some sources, NTLM and its 1980's predecessor, LM authentication, convert the user password into a 56-bit key and then use that to encrypt the nonce, and then use the encrypted nonce as the response to the server's challenge.

In contrast, there's a family of hash functions from RSA inc, called MD2, MD4, and MD5. (MD stands for "message digest" in all three cases.) These do not encrypt a message so that it can be decrypted, as DES does. Instead, they take a number of any length — the "message" — and create a 128-bit hash from it. (See <http://www.rsasecurity.com/rsalabs/faq/3-6-6.html> for more information on the MDs.) There isn't any straightforward way to take that 128-bit hash and decrypt it into the original message, and so the MDs are known as "one-way hash functions" or OWFs. Recall that I said that it's hard to get complete, reliable information on NTLM, and here's an example.

# Windows Authentication Schemes

By Mark Minasi

Where some sources say that NTLM uses DES to create a response to a server challenge, others say that NTLM combines the password and the nonce and then hashes it with MD4 to create the response. Apologies, but I don't know which is correct. Personally I'd guess that MD4 is what Microsoft uses, but inasmuch as Microsoft doesn't let me peek at the source code, I couldn't say for sure. I'm going to assume that it's MD4 for the rest of this discussion.

Let me finish this algorithmic discussion by showing how our friend Joe Dirtbag might exploit the incredibly terrible design of my hash function to figure out a password. Let's assume here that Joe knows our hash algorithm — the server hands the client a number, and the client must then view the password in numeric form and divide the password by the number, returning the remainder to the server. So he knows that the client took the password — which hasn't been transmitted on the network, recall — and divided it by 7, and came up with a remainder of 3. Does Joe D have the client's password yet?

No, clearly not. All the sneak knows is that it's a number equal to some exact multiple of 7, plus 3. That could be any number of values: 3, 10, 17, 24, 31...; actually, an infinite number of values. But he's got a clue. And he keeps listening. So perhaps later in the day, the client and a server talk like this:

```
Client: Let me in.  
Server: 11  
Client: 0  
Server: Okay, you're in.
```

Notice that the server used a different nonce in this second authentication — 11, not 7.

Can Joe Dirtbag guess the password now? Well, he knows that it's a number that's exactly divisible by 11, and is also equal to a multiple of 7, plus 3. The password must be a whole number, not a fraction, so he couldn't just set up a couple of equations and solve for them. But he's got a lot more clues, and bit of figuring shows that the first number that meets that criterion is 66, the next 143, and so on; just keep adding 77 to the previous number and you get the next candidate. There are only 13 candidates in the first 1000 numbers. Time to start getting nervous...

You can see where this is going. Suppose Joe sniffs another authentication challenge — the server says "23," and the client responds, "5." So the password is a number that's a multiple of 7, plus 3, it's evenly divisible by 11, and is a multiple of 23, plus 5. A few lines of VBScript let me quickly check the first 1000 numbers to find that the number of possible candidates is ... oops ... *one*. 143 is the only possibility in the first 1000 numbers. Just a few sniffs, then, provided enough clues to nail down a password.

But how likely is it that someone sniffing your network would get to see a lot of authentication challenges? Plenty likely. You don't just log on once in the morning; your workstation handles dozens of logons for you every day. And if you run an XP desktop, then you may be surprised to hear that XP does a whole BUNCH of authentications. Apparently XP periodically looks around for nearby systems and pre-negotiates connections to their shared devices, when there are any.

Those systems ask XP to prove that they should answer XP's question, asking XP to log on. XP uses your user name and password to log on, and gets the information. Thus, an XP system spends a lot of time responding to challenges with your account, tossing around responses that implicitly contain

# Windows Authentication Schemes

By Mark Minasi

your password. What's that you say, you find that a bit troubling? Then go to any folder and choose Tools/Folder Options/View and clear the check box "Automatically search for network folders and printers." (Or look in Knowledge Base article 320138.)

In any case, the moral of the story is that lousy hash algorithms are really bad. Worse news is that DES is crackable at this point (it takes days to do it, but it's still possible) and MD4 is apparently broken as well, at least according to RSA. Another reason to move up to more complex authentication algorithms, as you'll see.

## Strengthening Challenge-Based Authentication with Big Numbers

But can you see how we'd strengthen even this mega-lame system of mine? I said that a check of the first 1000 numbers turned up one candidate. But who'd use a system where you could have one out of only one thousand possible passwords? What if instead there were trillions or octillions of possible passwords and nonces? Well, in that case, my VBScript program, which can test 1000 possibilities in a few seconds, would be stymied. Recall that Microsoft nonces are either 64 or 128 bits, depending upon whom you ask. Either way, that's a lot of bits, and a lot of possibilities.

But once you get a good algorithm, and nonces that are large, random, and never-repeating, then do you have a truly secure authentication system? No. There's one more piece, recall — the password. No matter how good your security system is, if I can easily guess your password, then you don't have security.

## NTLM's Twisted Forebear: LM

So we've seen that Microsoft's NTLM authentication system uses challenges so that it needn't send the password over the wire, employs long challenge tokens so it'd take a long time to try every possible challenge token, one at a time, and creates responses with a hard-to-reverse hash algorithm, MD4. You'd guess that as long as you pick good passwords, then Microsoft security must be pretty hard to crack, eh? Well, yes, except for one thing. Even if you *do* choose a good password, then it might well be crackable due to a bit of vestigial software — LM.

LM is the old LAN Manager version of NTLM. It basically works the same way NTLM works, with a few exceptions. The big ones are this. First, it stores passwords in two separate seven-byte chunks rather than a single 14-byte chunk. (It does that because seven bytes is 56 bits, and they use DES which, you may recall, uses 56-bit keys.) Why's that important? Several reasons. First, if you have an account password that's longer than 14 characters, then LM doesn't care, it just truncates it.

So much for those long, hard-to-guess passwords. Second, LM stores all passwords in uppercase. That makes guessing passwords far easier. For example, if you were running an old LAN Manager system and you chose a password of "cAt," the system would just convert it to "CAT" and store it that way. No matter whether you tried to log in with a password of cAT, CAT, cat, CAt, or whatever, the system would just convert it to CAT and try to match it to the stored LM version of the password, which is always stored in all uppercase. Third, cracking seven bytes of a 14-byte password is trillions of times easier than cracking a 14-byte password. Sort of like the difference between a thousand square miles... and a thousand miles square.

Getting back to this LM issue, though, you may be thinking no problem, as you're not running LAN Manager, right? Wrong. Yes, NTLM is smarter than LM and stores your passwords in whatever case you type them in without truncation — create a password "00cAt!NtheeeeHat-\*%" and that's what you'll need to type to authenticate; NTLM-based systems — that is, every OS in the NT family — do

# Windows Authentication Schemes

By Mark Minasi

indeed store passwords of up to 128 characters. (NT 3.1 to NT 4.0's User Manager only had space for a 14-character password, but third party tools could allow you to use longer passwords. Windows 2000 and later leave space for longer passwords.)

But way back when NT 3.1 first appeared, Microsoft figured that there would be people who were using LAN Manager (you remember, all five of those folks) and who would like to migrate to NT 3.1, but to do it gradually, by adding a few NT boxes to the LAN Man network.

Now, the NT boxes used NTLM and the LAN Manager boxes used LM, and that could have meant that they didn't understand how to authenticate with each other. So Microsoft decided to make NT bilingual, and so taught NT how to do LM authentications. In order to make that easier, NT 3.1 stored two copies of your password — the NTLM version, with uppercase and lowercase, and the LM version, all shifted to uppercase. You didn't need to flip a "make me LAN Man compatible" switch, nope, nor did your NT 3.1 system detect LAN Man systems automatically in some way and only *then* create the lamer LM passwords. No, it just created and stored the NTLM and LM versions of your passwords locally in the SAM file. And if some system tried to log onto your NT 3.1 server to use a shared file or printer, then the NT 3.1 server would really prefer to do an NTLM-style authentication.

But just on the off-chance that one of those LAN Manager boxes wanted to authenticate, then NT 3.1 systems stood ready to do the simpler LM authentications. As a matter of fact, part of any authentication is the negotiation phase, where the client could say, "I'd really like to log onto you, but I only know how to do LM authentications," and by default the NT 3.1 systems would say "sure, no problem."

Thus, a dirtbag trying to impersonate you could write a program that tried to log onto your NT 3.1 network by trying a bunch of passwords, and he could make life a lot easier on himself by asking the NT 3.1 system to do LM authentications. As all LM passwords are uppercase, he wouldn't have to try anything uppercase, staggeringly reducing the number of possible passwords to try. Also, you'd do LM authentications now and then just because if a server is slow then it might not respond to your NTLM authentication request, and so your workstation would automatically drop back to LM.

Anyone listening would see an LM challenge and response which, again, wouldn't be the password — but it'd be a whole lot easier to use that challenge and response to either narrow down password possibilities or perhaps crack the password altogether. Or the dirtbag could create a member server that collected logons from your workstation to pass along to a domain controller, but that member server would always insist on LM authentications. All perfectly valid ways to gather more easily-cracked LM authentication information.

Man, those NT 3.1 systems were pretty insecure, weren't they? Good thing we run state-of-the-art networks with NT 4, Windows 2000, XP and 2003, eh?

Well, actually, there's some bad news there. You know that LM backward compatible thing? It's, um, still there. Even if you've got an Active Directory in native mode, then your DCs and other servers are ready, willing and able to not only do standard Kerberos-style logons, but also to do NTLMv2, NTLM, or LM authentications. And yes, your Active Directories and SAMs (on your NT 4 DCs or NT, 2000, XP or 2003 member systems) do still include the LM versions of your passwords. Yikes.

## De-Clawing LM

# Windows Authentication Schemes

By Mark Minasi

And let's talk about those stored passwords for a moment. Your account's password is actually never ever stored on a hard disk anywhere. Instead, when you type a password into any of the security dialog boxes then the operating system runs it through a hash function and stores the hash value, rather than the password. These passwords are sometimes called the "OWF passwords" or simply "the OWFs" because they're hashed with MD4, a one-way function hash. That hash is then encrypted, and *that's* what's stored on the hard disk in the SAM file. So your password is twice protected — first hashed, then encrypted with DES.

Sounds good, but I *did* mention that DES can be cracked nowadays, didn't I? Someone getting ahold of the SAM on your workstation or servers, then, could instantly get the MD4 hash of your password ("the OWF," recall) using a tool called pwdump or its follow-ons, pwdump2 and pwdump3. Once they've got your OWF, then they can run password cracker programs like L0phtcrack and potentially learn your password. So make sure those domain controllers are locked up! You can also use a tool called Syskey to replace the DES encryption with another encryption method called RC4, making SAMs a bit less crackable.

RC4 is used in SSL for secure Web transactions; read more about it at:

<http://www.rsasecurity.com/rsalabs/faq/3-6-3.html> .)

I warn you, though, Syskey is a pain in the neck in that it password-protects the SAM and requires you to type the password in every time you reboot the system.

But to return to LM, it presents two potential security holes. First, any time your system tries to authenticate to another system using LM rather than NTLM, then you expose the LM OWFs (also called the "LM hashes") of your password, where they can be snatched and decrypted offline. Second, if you let someone get physical access to your computer or if they get administrator-level access over the network, then that person can steal your entire SAM (and, I'm told, your Active Directory, although I've not seen that done) and, and again, decrypt your account passwords offline.

Those things are theoretically possible for NTLM hashes as well, but NTLM hashes are 14 or more bytes long, and so that's a lot more computationally difficult, at least at the moment. (When the one terahertz chips come out, that might change.)

Removing the LM security hole, then, involves two goals. First, you want to make your servers — and recall that basically all systems are servers in a Microsoft network, even the workstations — refuse to serve LM authentications. Second, you want those old easily-cracked LM passwords out of your SAMs and Active Directories.

Could this cause a problem of backward compatibility? As far as I can see, stomping LM doesn't cause problems for anyone but dirtbags. I suppose there might be the odd network appliance that's so old or so badly designed that it uses LM authentications, but in my experience I have never missed LM, once it was gone. Let me stress, though, that I've been running my network like this for a few months, but the network only includes Windows 9x, NT 4, Windows 2000, XP and 2003 systems. I don't have any Samba set up, any NAS devices, any OS/2 systems, or the like.

In fact, I'll recommend that you take things a step further, and shut off not only LM but NTLM, and only use NTLMv2 (and of course Kerberos if you're running Active Directory). NTLMv2 improves logons in several ways.

# Windows Authentication Schemes

By Mark Minasi

- First, as you've already read, NTLMv2 never sends the same nonce twice, making replay attacks impossible.
- Second, it hashes passwords with MD5, a more secure one-way function than MD4.
- Third, NTLMv2 supposedly makes snatching password information from authentications altogether impossible, given the current level of computer hardware.
- Fourth, it time-stamps information, further thwarting replay attacks. That, by the way, makes NTLMv2 *very* time-sensitive. Make sure that your systems are all synchronized to within 30 minutes of each other. (Time zones count, so don't worry — a system in Chicago that thinks it's 10 AM is perfectly synchronized with a New York system that thinks it's 11 AM.)

Moving to NTLMv2 requires very little work and if you decide that you don't like your network in NTLMv2-only mode, then changing back is very easy. But in any case, DO get rid of LM.

You can decontaminate your network of LM and NTLM, leaving only NTLMv2 and Kerberos, in four steps:

- Install NTLMv2-aware software on your clients. Windows 2000 and later don't need any changes. NT 4.0 needs a Registry hack. Windows 9x and ME need you to install the "DS client" and to change the Registry. You must install the DS client even if you only want to get rid of LM.
- Tell your servers to reject LM or NTLM logon attempts. You'll do that with Registry hacks on NT and a local or domain policy on 2000 and later.
- Tell your servers to stop storing the old LM version of the password. Not possible on NT 4, but possible with 2000 via a Registry hack, or a policy on XP and 2003.
- Finally, you've got to have your users change their passwords to flush out the old LM passwords.

## Getting Windows 9x/ME Ready

First, get the clients ready and let's start with the old folks... the Wintendo crowd, Windows 95, 98, 98SE, and ME. They're completely clueless about NTLM, doing their logons with LM. You can fix that with the DS Client for Windows, a free program that comes with Windows 2000 Server and that enables Windows 9x and ME systems to do NTLMv2 authentications.

The DS Client also allows your Wintendo boxes to get site-awareness (helpful with logons and fault-tolerant Dfs), gives them the ability to search the Active Directory, and access and modify Windows Address Book data.

First, make sure that the Wintendo box knows about the domain. Right-click Network Neighborhood and choose Properties. Then right-click "Client for Microsoft Networks" and again choose Properties. Check the box labeled "Log on to Windows NT domain." In the field labeled "Windows NT domain:," fill in the NetBIOS name of your domain, not the DNS name. For example, you'd fill in "ACME" rather than "acme.com." Click OK twice and you'll get the "rebuilding driver information database" dialog box.

Now you're ready to install the DS Client. It's on the Windows 2000 Server CD at \Clients\Win9x\dsclient.exe. Start it up and it'll install the client. Reboot and start up Regedit.

# Windows Authentication Schemes

By Mark Minasi

- Go to HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control and create a whole new key — a folder — named LSA.
- In that key, create a new REG\_DWORD entry named LMCompatibility and set it to 3. Close Regedit and reboot.

You can now log onto the Active Directory from a Windows 9x system using NTLMv2. Respond to the logon dialog box with separate name, password, and domain; jane@bigfirm.biz would log on as name=jane, password=whatever, and domain=bigfirm, rather than name=jane@bigfirm.biz and password=whatever.

## Getting NT 4 Systems NTLMv2-Ready

Next, teach your NT 4 boxes to eschew LM and NTLM in favor of NTLMv2. Your NT 4 box must be running SP4 or later to be able to do this.

- Open Regedit and go to HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\LSA and add a new REG\_DWORD entry, LMCompatibilityLevel — notice it's LMCompatibilityLevel, not LMCompatibility as in the Wintendo case — and set it either to 4 (reject LM, accept NTLM or NTLMv2) or 5 (reject LM and NTLM, only accept NTLMv2).
- Next, go to HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\control\LSA\MSV1\_0 and create two new value entries, NtlmMinClientSec and NtlmMinServerSec. These control the minimum acceptable level of security that the client and server pieces of your NT system will accept. Look at Knowledge Base article 147706 for all of the details, but if all you want to do is to require NTLMv2, then enter the hexadecimal value 0x00080000 for both.
- Then reboot the system. (If all you're concerned about is ensuring that your NT 4 clients use NTLMv2 in talking to Windows 2000/2003 domain controllers, then the LMCompatibilityLevel setting is all that you need, in my experience.)

Let me stress again that I've not had any trouble with NT 4 systems using NTLMv2, but there is a KB article 236414 that says that some people have had trouble, so take a look there if things don't work.

## Telling the 2000, XP and 2003 Systems To Only Use NTLMv2

Granted, having to install software and hack the Registry on the Wintendo boxes wasn't fun, nor was hacking the Registry on the NT 4 systems, although you could have created a custom system policy to accomplish that as well. For the 2000 and later systems, it's much easier; use a policy.

In the Group Policy editor, look in Computer Configuration/Windows Settings/Local Policies/Security Options, then "Network Security: LAN Man Authentication Level" on XP or 2003, or simply "LAN Man Authentication Level" on 2000. Choose either "Send NTLMv2 response only / refuse LM" or "Send NTLMv2 response only / refuse LM & NTLM," if you can. If you've got an Active Directory, then make it a domain-wide policy. If not, then you'll have to modify local policies on your systems singly. No reboots necessary, just force a policy refresh either with

```
secedit /refreshpolicy machine_policy
```

for Windows 2000 systems, or

# Windows Authentication Schemes

By Mark Minasi

```
gpupdate /force
```

for XP and 2003 computers.

## Getting Rid Of the LAN Man Hashes

By now, your network is running merrily along with NTLMv2. But those darn easily-cracked LM hashes are still sitting in your Active Directory and local SAMs. Time to eliminate them. You've got to do that in two steps. First, you tell the NT, 2000, XP or 2003 system to stop creating the LM hashes in the first place. But that only keeps the system from creating new LM hashes; it doesn't clear the existing ones out of the SAM or Active Directory. To do that, you've got to change every user's password.

There is not, unfortunately, a way to tell your NT 4 systems not to create LM hashes. But if you don't have any NT 4 domain controllers, then you're in good shape, as there shouldn't be many local accounts on your member servers and workstations.

On Windows 2000 systems, open Regedit and navigate to:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa.
```

Then create a new key — again, a key, with a folder icon, not an entry — named NoLMHash. Don't put any entries in it, just create the key. Reboot and it'll never create another LM hash.

On XP or 2003 systems, use a policy. Look in Computer Configuration/Windows Settings/Local Policies/Security Options/Do not store LAN Manager hash value on next password change and enable the setting. Or, if you like you can alternatively create a Registry entry:

- open Regedit and go to  
KEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa.
- Now, with 2000, you create a key named NoLMHash, but in the case of XP or 2003, you create a new value entry inside the Lsa folder named NoLMHash. It's a REG\_DWORD, set it equal to 1.

## Is It Worth It?

This has been a long article and thanks for staying with me, I hope it's been useful. I wanted to do two things: first, to blow away some of the smoke about how NT authenticates, and, second, to highlight a weakness and a solution for that weakness.

The fact is that LM hashes represent a real bonanza for any internal attacker. I think you don't hear people talk about them much because it can be uncomfortable to discuss security changes that are really only necessary because we don't trust our co-workers, but unfortunately we face more security challenges from insiders than outsiders. That's not to say that everyone in our organizations are dishonest, far from it; rather, it's that when we *do* get a rogue insider, then he or she can do a lot more damage than an outsider.

I personally think that the LM "hole" is one that Microsoft should have plugged a long time ago through their defaults, but they haven't, probably because so many clients use Wintendo boxes. With

# Windows Authentication Schemes

By Mark Minasi

hope we'll see LM just a bad memory soon, though. I urge you to seriously consider rolling out this change and let me close this by offering an performance incentive to go "all NTLMv2:" logons are faster. If you've ever read my pieces on how much faster NET USE commands become when you shut off NetBIOS, then you probably wondered why they got so much faster.

I never knew either, but since shutting off NTLM and LM, I've noticed much, much snappier response from my NET USE commands. I still don't know why, but now I've got a guess: getting rid of NTLM and LM just plain simplified the logon process. As the clients and servers have fewer options, things just happen more quickly. I guess that's why shutting down NetBIOS made things faster, as eliminating NetBIOS kills LM, NTLM, and NTLMv2.