Security Management - October 2005

Frequently Asked Questions About Passwords

Published: October 12, 2005

By Jesper M. Johansson [http://blogs.technet.com/jesper_johansson/]
Senior Security Strategist
Security Technology Unit

See other Security Management columns
[http://www.microsoft.com/technet/community/columns/secmgmt/smarch.mspx] .

I seem to have become the person people go to for answers to questions about passwords these days. Frankly, I am fine with that because I am very interested in passwords. It is a good area to have some interest in because we will not get rid of passwords for quite some time to come and they are used to protect very sensitive information. Operating systems and applications today are architected around passwords and even if you use smart cards or biometric systems, all accounts still have passwords and they can still be used in some circumstances. Some accounts, notably accounts used to run services, cannot even use smart cards and biometric tokens and therefore must have use a password to authenticate.

People often ask many of the same questions about passwords over and over again. To try to answer a few of them at one time, I figured it made sense to write an FAQ. Questions that I do not answer here might be answered in my blog [http://blogs.technet.com/jesper_johansson.aspx] , and, if events warrant, maybe I will update this article to reflect any fundamental new insights we come up with.

Frequently Asked Questions about Passwords and Password Attacks

How are passwords stored in Windows?

The Microsoft Windows operating system stores passwords many different ways for different purposes. For use in Windows networking, including Active Directory domains, the password is stored two different ways by default: as the LM OWF and as the NT OWF. OWF stands for One-Way Function and is a term that denotes a one-way mathematical transformation of some data The data that is being transformed can only be converted one way, into the obfuscated form. The obfuscated form of the data cannot be reversed into the original form, hence the use of the term one-way function. The most common type of OWF in use is a *cryptographic hash*. A hash is a small set of data that is mathematically tied to some larger set of data from which the hash is calculated. If the larger set of data is changed the smaller set, the hash, also changes. Hashes are useful, for example, as a checksum to verify that data has not been modified in transmission. A cryptographic hash is a hash that fulfills certain properties. A cryptographic hash must, for instance, by created in such a way that it is mathematically infeasible in a reasonable amount of time to infer the larger set of data from only the hash. Likewise, it is mathematically infeasible to find two sets of large data that generate the same hash.

Although the LM OWF is not actually a hash, its output is commonly called the "LM hash" since the NT OWF generates the "NT hash." For the sake of simplicity, I will use the term "hash" to denote each of these, even though hash is not always correct. The term OWF refers to the function used to calculate a hash.

There are many different types of one-way functions. All hash functions are, by definition, one-way functions. However, ordinary cryptographic functions that are typically reversible can also be used to create a one-way function. This can be done by swapping the data and the key in a cryptographic function and encrypting the fixed value, the key, using the data as the key. This is how the LM hash is computed. The LM hash is computed as follows:

1. The password is padded with NULL bytes to exactly 14 characters. If the password is longer than 14 characters it is replaced with 14 NULL bytes for the remaining operations.
2. The password is converted to all uppercase.
3. The password is split into two 7-byte (56-bit) chunks.
4. Each chunk is used as the key to encrypt a fixed string.
5. The two results from step 4 are concatenated and stored as the LM hash.

This entire process is shown in figure 1.

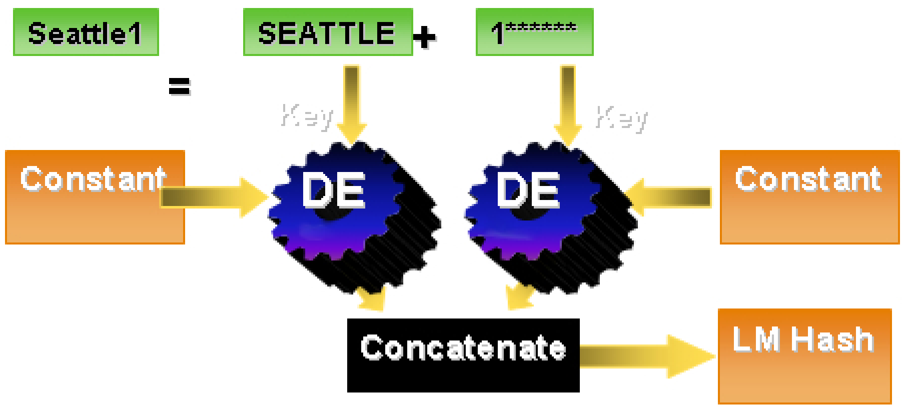

Figure 1

The LM OWF algorithm is now over 20 years old. It was originally created for use in the LAN Manager family of operating systems and is included in recent versions of Windows for backward compatibility with software and hardware that cannot use newer algorithms.

The NT hash is simply a hash. The password is hashed using the MD4 algorithm and stored. The NT OWF is used for authentication by domain members in both Windows NT 4.0 and earlier domains and Windows 2000 and higher Active Directory domains.

The NT OWF process is shown in figure 2.

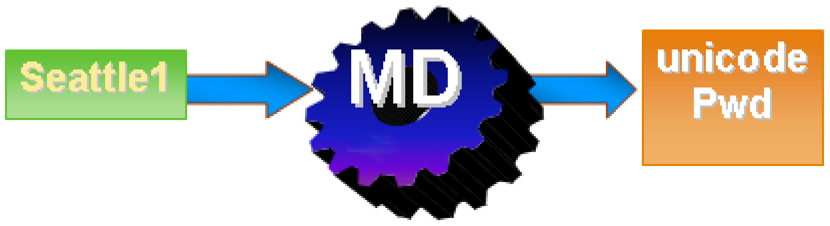

Figure 2

Neither the NT hash nor the LM hash is salted. *Salting* is a process that was first used on UNIX-based computers over 20 years ago. On those computers password hashes were stored in a world-readable file. A user could simply search the file for any other users who had the same stored hash. If any were found, it would mean they shared the same password. To solve this problem, the designers of the UNIX operating system decided to *salt* the passwords prior to storing them. The process of salting combines the password with a small number the salt - before computing the OWF. The salt could be stored in clear text in the password file. This ensured that two users with the same password had two different password representations stored. Windows has never stored hashes in world-readable form, so there has never been a need to salt them.

Windows also stores a password verifier on domain members when a domain user logs on to that domain member. This verifier can be used to authenticate a domain user if the computer is not able to access the domain controller. In Windows XP the password verifier is also used to speed up domain logon when the computer is cold-booted. The password verifier is also commonly called a cached credential. It is computed by taking the NT hash, concatenating the user name to it, and then hashing the result using the MD4 hash function.

Internally, Windows represents passwords in 256-character UNICODE strings. The logon dialog is limited to 127 characters, however. Therefore, the longest password that can be used to log on interactively to a computer running Windows is 127 characters. Theoretically, programs such as services can use longer passwords, but they must be set programmatically because the password change dialog will not allow a password longer than 127 characters.

How are passwords used in Windows?

When a user logs on, the password the user types is converted into both types of OWF and held in memory by the LSASS process. If the user is authenticating using a local account, the NT OWF is compared against the locally stored NT hash, and if the two match the user is logged on. If the user is authenticating against an Active Directory domain accessing a resource using a host name., the NT hash is used in a Kerberos logon against the Key Distribution Center (KDC, typically the domain controller). The password verifier is computed by WINLOGON, not LSASS.

In some situations Kerberos cannot be used. The following list shows situation where some other protocol must be used.

- Authenticating against a Windows NT 4.0 or prior domain
- Accessing a resource on an Active Directory domain member using an IP address rather than a host name
- Accessing a resource on a computer that is not a member of an Active Directory domain

- Accessing any resource on a Windows-based computer from a computer running Windows 9x or Windows NT 4.0, or a third-party operating system that does not support Kerberos

In these situations the authentication process uses two different protocols, called LanMan and NTLM. The process starts with the client requesting a challenge from the authentication server. Once the challenge is received the client computes a response to this challenge. This is done by first padding the two hashes of the password with NULLs to 168 bits. The 168 bits of each hash are then split into three chunks of 56 bits each, with each chunk usable as a DES key. The six DES keys are then used to encrypt the challenge. The three cipher texts produced using the LM hash are concatenated and become the LanMan response. The three cipher texts produced using the NT hash are concatenated and become the NTLM response.

The functions used to compute the response may be modified by the LM Compatibility Level setting discussed in Microsoft Knowledge Base article 147706. (LMCompatibilityLevel is presented in Group Policy as Network security: LAN Manager authentication level.) If that value is set to 1 or lower, the client will send the original LanMan and NTLM responses. If it is set to 2, only the NTLM response is sent. If it is set to 3 or higher a new version of both protocols is used. The NTLM version is called NTLMv2. The LanManager version is often referred to as LMv2. Both protocols use the NT hash to compute the response, and both use a client-side challenge, either instead of, or in addition to, the server challenge. In addition, if the LM Compatibility Level setting is set to 1 or higher, the NTLM response is time-stamped to prevent replay attacks. This is shown in table 1.

Table 1 Client-Side LMCompatibilityLevel Impact

Level	Group Policy Name	Sends	Accepts	Prohibits Sending
0	**Send LM and NTLM Responses**	**LM, NTLM**	**LM, NTLM, NTLMv2**	**NTLMv2, Session security**
1	**Send LM and NTLM[md]use NTLMv2 session security if negotiated**	**LM, NTLM, Session security**	**LM, NTLM, NTLMv2**	**NTLMv2**
2	**Send NTLM response only**	**NTLM, Session security**	**LM, NTLM, NTLMv2**	**LM and NTLMv2**
3	**Send NTLMv2 response only**	**NTLMv2, Session security**	**LM, NTLM, NTLMv2**	**LM and NTLM**

The server evaluates the responses in a particular order, also governed by the LM Compatibility Level setting. All servers since Windows NT 4.0 Service Pack 4 start out by evaluating the NTLM response as if it were computed using NTLMv2. If this fails and the LM Compatibility Level is set to 4 or lower, generally the server will then evaluate the NTLM response as if it were an original NTLM response. If this fails, and the server is set to LM Compatibility Level 3 or lower it evaluates the LanMan response. If none of these matches or if the LM Compatibility Level setting is set to prevent the server from evaluating the LanMan and/or NTLM response, the authentication fails with an error message of bad username or password. This is shown in table 2.

Table 2 Server-Side LMCompatibilityLevel Impact

Level	Group Policy Name	Sends	Accepts	Prohibits Sending
4	**Send NTLMv2 response only/refuse LM**	**NTLMv2, Session security**	**NTLM, NTLMv2**	**LM**
5	**Send NTLMv2 response only/refuse LM and NTLM**	**NTLMv2, Session security**	**NTLMv2**	**LM and NTLM**

Keep in mind that when we talk about the server-side or the client-side impact of the LMCompatibilityLevel setting we are referring to the impact of *any* Windows (or compatible) computer that operates as a client or a server. One of the very confusing things about this setting is that it often is described in terms of "domain controller settings." The fact is that the server-side applies to any server that holds the authentication database. For instance, if you connect to a computer running Windows XP Professional using a credential defined in the SAM on that computer, it is acting as a server and the server-side settings guide its behavior.

Windows NT 4.0, Service Pack 4 and higher, Windows 2000, and 32-bit editions of Windows XP have LM Compatibility Level set to 0 by default. Windows Server 2003 and Windows XP 64-bit edition have it set to 2 by default.

What breaks when I change the LMCompatibilityLevel?

A number of things will break if you change the LMCompatibilityLevel on a computer. If you require inbound NTLMv2 (that is, level 5) you will break the following:

1. Inbound authentication from Windows 95 or Windows 98 clients that do not have the Directory Services Client installed.

2. RRAS servers running versions of Windows prior to Windows Server 2003 Service Pack 1 will break if the Domain Controllers have LMCompatibilityLevel set to 5. You must upgrade the RRAS servers or set LMCompatibilityLevel to 4.

3. Any RAS server that needs to process CHAP will be unable to do so against a DC that has LMCompatibilityLevel set to 5.

4. Clustered computers running versions of Windows prior to Windows Server 2003 Service Pack 1 will break if LMCompatibilityLevel is set to 5. Clusters use RPC over UDP, and RPC over UDP cannot use NTLMv2 by default. This was resolved in Service Pack 1.

If you configure LMCompatibilitylevel to 3 or higher you might run into problems with third-party hardware devices that have an SMB server built in. Many of these cannot authenticate inbound traffic using NTLMv2 and therefore will break when you set LMCompatibilityLevel to 3 or higher.

How can passwords be attacked?

There are several conceivable ways to attack passwords. The simplest way is probably also the most effective. If you want to know what someones password is, just ask them! About 70% of those questioned in a 2004 survey [http://news.bbc.co.uk/1/hi/technology/3639679.stm] would reveal their password if someone traded them something they value more than your organizational secrets, their credit card information, their bank account, or whatever asset that password is protecting. The study used chocolate as the valuable asset to trade: http://news.bbc.co.uk/1/hi/technology/3639679.stm [http://news.bbc.co.uk/1/hi/technology/3639679.stm] .

This survey really highlights an important point. The absolute weakest links in a password-based security system are people. In fact, only half jokingly, I once said, There is nothing wrong with passwords that removing the people using them wouldnt solve. Passwords fail to provide security because people often create weak ones or they give them away, inadvertently or not. To be fair, though, a large part of the problem lies in the absolute overload of password-based systems and the fact that most people have never really been taught how to manage passwords. But that topic is for a different article that would probably be better written by someone with more people skills.

If we look at purely technical methods of attacking passwords, the first involves guessing the password from a logon prompt, either locally or remotely. This approach will only succeed with very poor passwords and will take considerable time. If passwords are seemingly random to the attacker then any attack on them follow a random distribution where each possible password is equally likely to match the target password. Therefore, statistically speaking, an attacker would have to test half of all possible passwords to match any given password. Of course, it is possible that the first password tested matches the target, but the likelihood of that is one divided by the number of possible passwords extremely unlikely unless the character set and length used are extremely small. Therefore, if passwords are randomly composed of eight characters chosen at random from at least three of the four types of characters (uppercase letters, lowercase letters, numbers, and symbols) a guessing attack will take between 1,500 and 1.4 million years, depending on how fast the tool is. The low number is based on the fastest tool I have heard of, which can do 1,700 guesses per second.

A second way to attack passwords is to capture the challenge-response pairs and then attempt to crack those. Crack is a generic term that basically encompasses many different methods of attempting to guess the password. All of these methods rely on producing a test password, running that through the same algorithms the real one would have been run through, and then comparing the result to the captured value. For instance, if an attacker captures the LanMan challenge-response pairs, he might start by guessing that the password is password1. In this case he first computes the LM hash of password1. Then he would compute a response using the LM hash and the challenge sent by the server. If this response matches the one the client sent, the password has been found. If not, the process starts over with a new test password. This type of cracking is considerably faster than a guessing attack, but still is quite time-consuming. For LM hashes it involves computing five different DES functions, each of which is very slow. A fast computer can typically compute about 3,000,000 LanMan challenge-response pairs per second through an optimized algorithm. If it were used in a LanMan challenge-response computation, the eight-character password discussed earlier would take about 3 years to break using a brute force attack where the attacker simply tries all possible passwords. The same computer can compute around 1,500,000 NTLMv2 or LanManv2 challenge-response pairs per second. Since the NTLMv2 and LanManv2 challenge-responses use a case-sensitive hash the time to crack increases to 70 years using those protocols.

Cracking the NTLM challenge response might actually be faster, depending on the character set used and the length of the password. The NTLM challenge-response only three DES computations and one MD4 hash. Computing an MD4 hash is much faster than computing a DES encryption. However, the NTLM challenge-response is computed using a case-sensitive hash, meaning there are far more options to consider. A rough estimate is that the same computer could test 4,500,000 NTLM challenge-response pairs per second, resulting in a crack time of 23 years for our 8-character password.

A third way to attack passwords is to break into the computer that stores the hashes. Those hashes could then be cracked using essentially the same technique as for captured challenge-response pairs, but using far fewer computations. Using the eight-character password example it would take only 6 days to brute force the entire space of LM hashes. This is partially because there are now only two DES computations to perform, but most of the speed-up comes from the fact that the LM hashes stores an eight-character password as one 7-character password and one 1-character password and the two pieces can be cracked separately. Had the password been stored as a single 8-byte chunk it would have taken over a year to crack.

What are Rainbow tables?

A new twist on an old attack arose in 2004. For many years security analysts have known that cracking times could be speeded up by several orders of magnitude if the cracker could simply compare the captured hashes against hashes that were already computed from known passwords. This attack, known as a precomputed hash attack, works only if the passwords are not salted; otherwise the same password would have many different possible hashes. The method promises huge improvements in speed if it can be made to work. Possibly the first implementation of this type of attack was in a product called Quakenbush Password Appraiser, which came out in 1998. The problem with this approach is that it requires huge storage space to hold the computed OWF functions, and it takes a very long time to compute them all. In 2004 Philippe Oechslin discovered a solution to the former problem. Instead of storing the entire hash, a Rainbow table stores only a piece of it along with the passwords that are known to generate that hash. The cracker can now simply look up the pieces of the hash that are stored and compute the hash only for those passwords that are known to produce that hash. This approach speeds up cracking by several orders of magnitude. Even a poorly optimized early implementation of this attack resulted in cracks that took less than an hour instead of 57 days. The approach was implemented in the tool RainbowCrack.

Rainbow tables work equally well for the LM hash and the NT hash, with the caveat that the storage space required for the NT hash is several orders of magnitude greater than that for the LM hash if hashes for passwords longer than eight characters are desired. Storage for the LM hash is within reach of most users, however. A set of Rainbow tables for the LM hash computed from the alphanumeric keyspace and the 14 symbols on top of a US English keyboard takes about 17 GB of storage. A set of Rainbow tables for the LM hashes computed from the full character space on a US English keyboard takes about 47 GB of storage.

Rainbow tables still take very long to compute, due to the huge number of computations required and to the poorly optimized tools currently available. We should expect huge improvements in performance as the tools get better. In addition, Rainbow tables have been available for purchase online for over a year, and users of various file-sharing services can now download Rainbow tables alongside the music, movies, and software typically available on those services. In fact, various security researchers have developed improved generation techniques to make it faster to generate the tables.

Should I be concerned about password cracking?

The answer is a qualified no. Cracking against captured hashes is not an interesting attack. The hash is the only secret used in challenge-response protocols today both on Windows and on other operating systems. An attacker with the hash has all that is required to authenticate as the user and cracking is simply a waste of time. Tools that implement this type of attack, known as a pass-the-hash attack, are available on the Internet already. Although the tools are still relatively unstable and unreliable, we should expect their quality to improve significantly as passwords get better. In addition, in order to capture the hashes the attacker must break into the authentication server, which is typically the domain controller on a Windows-based network. If this has occurred, the network has been fully compromised already and cracking passwords is only useful to attack other networks where the same users used the same password (a practice that should be strongly discouraged). Windows will never pass the hashes across the network in an unencrypted channel. Thus, capturing the hashes in a network transfer is pretty unlikely.

Since January 2005, cracking against the stored password verifier (the cached credential) has received renewed interest due to the publication of the first commonly available tool to extract those verifiers from a compromised computer. Cracking the password verifier is reasonably efficient, taking roughly three times longer than cracking against the NT hashes. This, however, is really not an interesting attack either. First, the verifier is a hash of a hash, making it at least twice as resilient as the original hash. Second, since the password verifier is salted with the username it is unique even for two users using the same password. While a pre-computed hash attack can still be used to speed up the first computation, it is useless against the second hash. Finally, the password verifier serves a very important purpose. Without it users would have to use local credentials when logging on to a computer while disconnected from the domain but use domain credentials when connected. Clearly this process would create a nuisance for users. More important, however, is the effect it would have on the captured hashes. Most security researchers would agree that a vast majority of users would use the same password for both accounts. If an attacker compromises a computer that has local credentials and those credentials match credentials on a domain, the attacker would now have all the information needed to authenticate as the user to the domain and would not have to crack a single password. The OWF representation stored locally for the local account is identical to the one stored for the users domain account if the two have the same password. Clearly the dynamics of the human-computer interaction mean that the password verifier probably increases security instead of decreasing it. Therefore attacks against the password verifier are probably also not interesting.

Cracking against captured challenge-response pairs is, however, still an interesting attack. However, using the LM Compatibility Level switch it can be effectively invalidated by not transmitting the LanMan challenge-response pairs. This is a valid approach in almost all environments. Some (quite possibly not all) problems with LMCompatibilityLevel were listed above. If you are not subject to any of those issues, we recommend that you configure LMCompatibilityLevel to 5.

Why is the LM hash stored if it is so vulnerable?

The LM hash is stored for backward compatibility reasons. Many environments no longer need it and can disable storage of that value. Knowledge Base article 299656 [http://support.microsoft.com/default.aspx?kbid=299656] discusses a registry value that can be used to configure this behavior. This will prevent attacks against captured LM hashes from a compromised authentication server. However, it will not prevent any computer from sending the LanMan response during an authentication sequence. In addition to configuring the registry value discussed in KB 299656 [http://support.microsoft.com/default.aspx?kbid=299656] , storage of the LM hash can be prevented by using a password longer than 14 characters or by using certain Unicode characters in the password. Discussion of which Unicode characters result in not storing the LM hash is beyond the scope of this article but is covered at length both in the Windows 2000 Security Hardening Guide [http://www.microsoft.com/downloads/details.aspx?familyid=15E83186-A2C8-4C8F-A9D0-A0201F639A56&displaylang=en] and in *Protect Your Windows Network* [http://www.awprofessional.com/title/0321336437] , by Johansson and Riley.

Please note that the actual realized security value of not storing the LM hash is negligible in the face of a skilled and sophisticated attacker. The only attacks that this prevents are those that rely on cracking captured LM hashes from a compromised authentication server. As mentioned earlier, cracking of such hashes should be considered a waste of time for sophisticated attackers. First, the NT hash is perfectly adequate for authenticating as the user without cracking. In a very real sense, there is no difference in security value between a 1-character password stored using an LM hash and a 127-

character highly complex password stored using the NT hash. Both generate a hash that can be used to authenticate as the user, and if the LM Compatibility Level value has been set to 4 or higher on the target server the LM OWF is useless anyway.

What should I do to protect my passwords?

The most important thing to do to protect passwords is to use good passwords. Good passwords defeat all the attacks above except for the one using captured hashes, which is defeated only by adequately protecting your authentication servers. Some characteristics of good passwords are:

1. Long -- A good password should be at least eight characters long. The longer the better. Passwords shorter than eight characters are inadequate today. If we assume that a very optimized cracker could test 3,000,000 captured LanMan challenge-response pairs per second, a captured challenge-response pair of a fully random eight character password using all 69 characters (including the space bar) on a U.S. English Keyboard, will resist cracking for almost three years - far longer than the typical password change interval. If NTLMv2 authentication is used, the efficiency of the best crackers today drop to less than 800,000 per second, meaning that the same password will resist cracking for over ten years. Note, however, that these calculations are based on current computing power available. Five years from now, common computers will be able to crack the LanMan authentication in only 98 days, based on an increase in power following Moore's law. That means that five years from now, passwords based on those 69 characters must be nine characters long to resist cracking for 180 days.

   Note that the length of the password is far more important in cracking resistance than the number of characters in the character set. For example, a 7-character long case-insensitive password using all characters on a US English keyboard (69 characters) will resist cracking against captured challenge response pairs for only 14 days. Using a case-sensitive password (95 characters) but leaving all other parameters the same, the password resists for 135 days, still not very long. Now add a character to it, making it an 8-character password. The case-insensitive version will resist cracking for 991 days and the case-sensitive version will resist for over 35 years! These calculations clearly indicate that password length is far more important than the number of characters in the password set; *assuming that the password truly appears random to the attacker* and cannot be attacked using dictionaries or heuristics. If you are trying to improve password strength in your organization, teach people to use longer passwords that are not based on common words. One technique is also to base passwords, or better yet, pass phrases, on words in other languages than the primary one at the site.

   Because length is so important in passwords, an approach that has become very popular recently is to use pass phrases instead of passwords (see Pass phrases vs. Passwords [http://go.microsoft.com/fwlink/?LinkId=28592]). A pass phrase is a phrase, complete with spaces and punctuation, which is used as an authentication token instead of a password. Windows is perfectly capable of using pass phrases already. Pass phrases also seem to be easier for people to remember, making them even more attractive.

2. Complex -- A good password should have a mix of all the four character types, uppercase and lowercase letters, numbers, and non-alphanumeric symbols. Preferably, all four should exist in a given password. Remember, any character on your keyboard is legal in a password. Using a pass phrase and interspersing it with randomly chosen characters and spaces considerably improves the strength of your password.

3. Changed frequently -- A poster from NativeIntelligence.com [http://www.nativeintelligence.com/ni-posters/poster-next-prev.asp?PosterID=115a] has a great phrase on it: "Passwords are like bubble gum; they are better when fresh." Passwords should be changed every 90-365 days, depending on the value of the asset they are protecting and the strength of the password. If you use eight-character passwords, 180 days is a reasonable change interval. If you use nine-character passwords today you can probably leave them valid for 360 days or even longer without a problem.

4. Used only in one place -- Reusing passwords significantly increases the exposure of the assets you are protecting with the passwords. Essentially, any given asset is only as secure as the least secure computer that protects that asset. When you reuse passwords the asset is only as secure as the least secure computer where you use that password.

5. Used only by one person -- Another characteristic that passwords borrow from bubble gum is that they are much better when used by a single person. If at all possible, each user on the computer should have their own account with their own password. This increases accountability and decreases exposure for the password. If you allow multiple people to use the same password you effectively give up all possibilities of monitoring their actions unless you install closed-circuit TV systems to oversee the computer. In addition, if users must perform administrative actions (computer maintenance, account management, running DirectX games, using poorly written software, etc) in addition to non-administrative work (browsing the web, reading e-mail, etc) they should have two accounts one which is an administrator and one which is not. It would be prudent to prevent the administrative accounts from browsing the web and accessing e-mail. However, this must be done on the network firewall or another computer which the administrative account cannot access to be effective. You cannot constrain the actions of a local administrator on the local computer.

6. Not typed on untrusted computers -- A password is only as secure as the computer or network it is used on. Keystroke loggers frequently target public kiosk-type computers, such as those used in Internet cafs, conferences, and hotel and airport lounges. The instant a password is typed on one of these computers fitted with a keystroke logger, the asset protected by the password is no longer secure. Keystroke loggers exist both in software and hardware form, and range from free for many software implementations to around $25 for a hardware version that sits between the keyboard and the computer capturing and storing everything that is typed. Some of the more sophisticated ones even have Web servers that enable an attacker to retrieve your password remotely across the Internet. It is good practice to view any public-use computer, or any private-use computer that has been physically insecure, as a compromised computer. Likewise, capturing passwords on wireless and public-access networks is

extremely common. There are even underground sites that trade in these passwords.

Given all these requirements, and given the fact that we all have many different passwords, how are we to remember them all if they are all long, complex, only used once, and so on? Unfortunately, the security industry has for years perpetuated the notion that writing down passwords is a security breach. Nothing could be further from the truth. Writing down your password is an excellent idea as long as you *adequately protect* the medium you wrote it on! Doing so allows you to remember more and better passwords, thereby *increasing* security, not decreasing it. Of course, writing your password on a sticky note and posting it on the monitor leaves it vulnerable to the binocular attack, but if you write it on a sticky note and put it in your wallet it is fairly well protected. Software products for managing passwords abound. Most simply allow you to store a password that you create; many also will enter it for you automatically on a Web site. Some of the better ones will actually generate passwords for you, creating passwords that are much more complex than what the average human would create. There are some that create the passwords on the fly from known inputs and do not need to store anything. The basic idea behind these tools is that while you retain the ability to have a single secret that protects all computers the secret is not exposed on any of those computers. One such tool comes with *Protect Your Windows Network* [http://www.awprofessional.com/title/0321336437] .

An even better idea than using complex passwords is to use smart cards. Versions of Windows since Windows 2000 are capable of using smart cards for logon to Active Directory domains. A complete discussion of smart cards is beyond the scope of this article. However, there is one key fact to keep in mind: even when you require smart cards for interactive logon, each user still has a password and it can still be used for network logon purposes. It is highly unlikely that we will see a usable, completely password-less system in the next 10 years, and even then there are millions of services, such as e-mail and Web sites, that will continue to use passwords. Clearly, learning about the risks inherent in passwords and learning how to better use them is a worthwhile endeavor.

What about pass phrases?

About a year ago, I wrote an article on whether pass phrases were better than passwords [http://www.microsoft.com/technet/community/columns/secmgmt/sm1004.mspx] . The article essentially stated that we *think* pass phrases are better than passwords, but we do not have any scientific proof of it.

What we do know is that people tend to use letter substitution in their password. This is where you replace a letter with another one. For instance, replace s with $. I remember how smart I thought I was the first time I came up with that idea. Well, the bad guys had already thought about it by then. If your password is Seattle1 (which by the way is complex according to the built-in complexity rules), there is only one substitution of $ for s and the bad guy only has to check two passwords to find the right one.

But what if your password is I use a very strong pass phrase for most of my passwords.? This one is also complex because it has an uppercase letter, a few lowercase letters, and a symbol. It also has nine s characters. An attacker who suspects that you replaced one or more of them with $ would have to try 512 combinations to find the right one. Pass phrases have a very important advantage over passwords: they increase the value of letter substitution by many orders of magnitude.

Pass phrases also let you remember much longer passwords than would otherwise be possible. Adding length to passwords is the primary factor in making them stronger. In the previous article series I asked for sample pass phrases. While many of them were simply words strung together without spaces, they were long! The average length was almost 31 characters. Compare that to the average nine-character length of passwords, according to a prior study documented in the aforementioned article series. And 75% of the pass phrases appeared to be of an essentially personal nature. That means that I was unable to discern any source for the statement. The next biggest category was literature, which gave rise to at least 6% of the pass phrases.

The other nice thing with pass phrases is that they are easy to remember and can be used by virtually anyone. I got my oldest child to use a pass phrase when he was 6. Try that with a password like K%Dsi&7e, which is not even as strong as the pass phrase he is using.

What should I do about passwords on service accounts?

Service accounts present an interesting situation. In most cases, the passwords needed for service accounts will only be used programmatically, not interactively. That means they can actually be considerably stronger than passwords used interactively. For example, the graphical logon interface dialog box only supports passwords up to 127 characters. However, internally Windows 2000 and higher allow passwords to be up to 256 characters long. For accounts that only need to be used programmatically, such as most service accounts, we can actually take advantage of this to set a password that cannot be used interactively. If the password will not ever need to be typed by a human being there is no real reason not to set it to be as long as is allowed. Note that you cannot configure a password that is longer than 127 characters in the GUI. To do so you need a tool that calls the NetUserChangePassword or NetUserSetInfo APIs directly. The length limitation is only in the GUI, not in the APIs.

The second special issue with service accounts is that generally speaking we do not wish for service account passwords to expire or to be locked out. If the password is expired, or the account is locked out the service using that account will not start. Therefore, we must make sure we disable password expiration on service accounts. Instead we should have a process for changing those passwords at regular intervals. Account lockout considerations are covered below.

The final thing to remember on service accounts has less to do with the passwords than with the accounts themselves. If an attacker compromises any computer that uses a service account the attacker has access to the password for that service account. That means the attacker now has compromised any other computer that trusts that service account. This is called a service account dependency. However, since it is not specifically related to the password on the service account the interested reader is referred to Protect Your Windows Network, chapter 8, which covers this issue in depth.

How do I manage passwords on the built-in administrator account?

Everyone knows that you should set a strong password on the built-in Administrator account. However, you might want to question whether you even want to allow use of that account. It is typically not needed after the computer has been set up (note: Windows Small Business Server is a notable exception. You do need the built-in Administrator account on that platform). Unless you need the account you can disable it after you are finished setting up the computer. Every person that administers the computer should have their own administrative account. If they all use the built-in Administrator account you have no way to audit anything they do. (Keep in mind that you can only audit users who are administrators as long as they themselves effectively consent to it. Should they so desire, they can disable or modify the audit entries. They are administrators after all). Without using unique administrative accounts you lose all accountability of your administrators.

It is also critical that you set unique passwords on the built-in Administrator account. Most organizations today do not do that. We have standard build scripts that build clients and servers and they all contain a hard-coded password. The net result is that all the computers in the environment have the same password on the local Administrator account. If any one of those computers get compromised the attacker will have the hash for this password. That hash can now be used to log on to any of the other computers that use the same password on the account. This is truly the principle of most privilege in action. The entire network is no more secure than the least secure computer in the network. What we must do is set a unique password on the built-in Administrator account on each computer. This is a complicated thing to do, but very worthwhile. I am told that User Manager Pro [http://www.liebsoft.com/index.cfm/products/ump] , can help you manage this, although I do not have first-hand experience with it.

You can also use a command line tool that can set the password. Should you wish to use the account, you must be able to recover the password. That means either writing it down, or using a tool that can recover it for you. One such tool is passgen, which comes with *Protect Your Windows Network* [http://www.awprofessional.com/title/0321336437] . Passgen is designed to set a pseudo-random password on regular accounts and service accounts, across the network. The password can either be fully random, or based on two pieces of known input: a unique identifier for the computer or account, and a common pass phrase. The common pass phrase is the global secret, but it is not stored on any of the computers protected by the secret. Hence, it is much easier to protect than the administrator password on thousands of computers.

You might consider using a blank password on the local Administrator account. Particularly on servers, which are physically secured in their racks, this is a reasonable option. By default, Windows XP and Server 2003 will not allow you to log on over the network to an account with a blank password. As long as the server is secured physically, and this feature has not been disabled, you will have a local Administrator account that cannot be used over the network but is usable locally should the need arise.

What should I set account lockout to?

You should turn it off. Account lockout is a feature that locks out an account after a certain number of attempts to log on with an incorrect password. It is designed to protect the computer against weak passwords. The problem is that weak passwords will eventually fall to an attack anyway, regardless of account lockout. The smart attacker will simply modify the attack in such a way as to not trigger the account lockout. A weak password will resist longer against such an attack when account lockout is used, but it will still eventually be broken.

In addition, account lockout makes it trivial for a less sophisticated attacker to disable the computer completely. A simple batch file can be used to lock out every account on the computer, thereby crippling it. Account lockout, while designed to protect against weak passwords, instead create a condition where a trivial denial of service attack is possible.

Account lockout, at least in Windows, is also an all-or-nothing setting. You cannot select to subject only certain accounts to account lockout. Thus, if you are interested in protecting only certain accounts with bad passwords you must enable account lockout for all accounts. This exposes all accounts on the computer to the denial of service vulnerability described earlier. If a computer has account lockout enabled and an attacker locks out an account used to start a service, for example, the service will not start. This could have dire consequences not only on uptime but also on the business objectives and revenue stream of the organization if the account is used to run the business or generate revenue.

The real solution to the problem of weak passwords is to use better passwords. As we pointed out earlier, it will take thousands of years to guess a good password, making account lockout entirely unnecessary in environments that enforce them. In addition, account lockout often necessitates a call to the helpdesk. Estimates on the cost of these calls range from $30-70 per incident.

Some would argue that we need account lockout to protect ourselves from people, because people will always chose bad passwords no matter what we do. Left to their own devices, they probably will, but we can put restrictions in place to prevent that. A password policy should enforce strong passwords. Should you wish, you can use a password filter to enforce even stronger restrictions and keep people from using dictionary words, or even permutations of dictionary words. If you cannot trust your people to pick good passwords, then you must prevent them from having to pick good passwords by using smart cards or some other form of authentication token such as RSA SecurID. Consider the problem carefully before you end up spending a good chunk of your helpdesk budget on account resets simply to allay a fear that people will not pick good passwords. In the end, weak passwords will still fall. The right solution is to train people to not use them.

People are the Weakest Link; and the Strongest Defense

I tend to be an optimist in the sense that I think people can actually be trained to be at least marginally security conscious. There are currently technical solutions, such as smart cards, that attempt to remove people from the equation. These solutions also remove the need for measures such as account lockout without compromising the stability of your network. The problem is that smart cards are not a complete solution as many applications will not support them. As mentioned earlier, accounts will always have passwords and some accounts cannot use smart cards. Nevertheless, use of smart cards mitigates some of the problem, and that is, and will remain, preferable to account lockout. However, none of these technical measures will be effective against an attacker exploiting people a social engineering attack. A person that would trade their password for chocolate would probably also trade a smart card for chocolate, although the cost to the attacker may be higher. In the end, people end up being the weak link in any human-computer system, and attackers are increasingly skilled at attacking people.

Much of the current commonly accepted wisdom about passwords, including all the arguments that we need account lockout turned on, is based on the assumption that people cannot be trained to use good passwords. However, if people cannot be trained to use a password that is resilient against a guessing attack, then we must also assume that we cannot train them not to trade it for chocolate to the first person who asks. If we have to give up all hope of training our people then security will be a losing battle forever. Attackers will always find a way around it by exploiting people, and people will always remain the weak link. Attackers who focus on our people will be much more successful than those who focus on technical attacks and we will never be able to prevent this. The thought that people can be trained to perform rudimentary security-related actions is what keeps me going to work. Please do not spoil that for me.

Should you actually succeed in training your people, they will likely be the strongest line of defense you have. They will understand the need for information protection and take reasonable steps to implement it. They will recognize attempts at attacks against people and computers and will be a very effective intrusion detection (and prevention) system. And, they will obviate the need for some of the more onerous technical security measures we put in place to guard against human mistakes deliberate or accidental. Investment in human security may be the most important security investment you undertake, and passwords are riding the front of that wave!

Learning More

If you are really interested in passwords, read Chapter 11 in *Protect Your Windows Network* [http://www.awprofessional.com/title/0321336437] . It contains an in-depth discussion that includes the kinds of passwords people actually use in the real world and why pass phrases might or might not be better than passwords.

As always, this column is for you. If there is something you would like to see, please send me a message by clicking Contact Us at the very bottom of the page. You may also contact me through my blog [http://blogs.technet.com/jesper_johansson/default.aspx] . You will find a thread there where we can carry on discussions about this very article.

Top of page