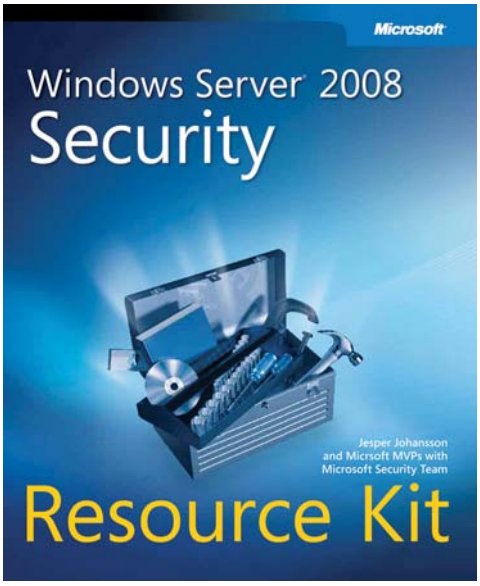

Windows Server® 2008 Security Resource Kit

Jesper M. Johansson and MVPs with the Microsoft Security Team

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/11841.aspx

Microsoft® Press

978-0-7356-2504-4

Table of Contents

- LM Hash
- NT Hash
- Password Verifier
- In Memory
- Reversibly Encrypted

Authenticated Protocols

- Basic Authentication
- Challenge-Response Protocols
  - Digest Authentication
  - LM and NTLM
  - NTLM v2
  - NTLM++
  - Kerberos

Smart Card Authentication

- Smart Cards and Passwords

Attacks on Passwords

- Obtaining Passwords
  - Ask For Them
  - Capture the Passwords Themselves
  - Capture the Challenge-Response Sequence
  - Capture the Hashes
  - Guessing Passwords
- Using the Captured Information
  - Cracking Passwords
  - Pre-computed Hash Attacks
  - Pass-the-Hash Attacks
- Protecting Your Passwords

Managing Passwords

- Use Other Authenticators
- Record Passwords, Safely

Chapter 13 Patch Management

Chapter 14 Securing the Network

Chapter 15 Securing the Branch Office

Chapter 16 Small Business Considerations

Chapter 4

Authenticators and Authentication Protocols

Jesper M. Johansson

Once you have a subject, that subject needs some way to prove that it really is who it claims to be. Consider the very real-world case in which you want to purchase something with a credit card in a store where they actually understand security. You have your identity: you. However, the store's personnel do not know who you are so they require some proof—an authentication that you are who you say you are. To provide proof of identity you use an authenticator of some form, such as an identity card or a passport. You present this to the store clerk in a fairly routine fashion, as an authentication protocol.

The virtual world is no different, with the exception that the entity to which you have to authenticate understands that a signature on the back of a credit card is not an authenticator. Therefore, you need a stronger form of authentication. In this chapter, we will discuss how Windows handles authenticators and which authenticators it supports.

Something You Know, Something You Have

Generally speaking, there are three types of authenticators:

1. Something you know
2. Something you have
3. Something you are

Something You Know

A secret that you know, and in many cases share with the system you want to access, is the simplest and most pervasive form of authenticator. A password is a perfect example of something you know.

Something You Have

A token of some kind that you are in possession of is a different kind of authenticator. You authenticate as yourself by proving that you are in possession of this token. An example is a smart card (discussed later in the chapter) or a SecurID one-time password device (*http://www.rsa.com/node.aspx?id=1156*). These types of tokens are almost always combined with something you know, and can greatly strengthen the quality of the authentication claims.

Something You Are

Some systems use something you are as an authenticator. These typically fall in the category of biometric authenticators: tokens that attempt to measure something about you. Examples include retina scans, fingerprints, blood samples, voice recognition, and

typing cadence. Biometric systems are inherently imprecise and, unlike the other two types of authenticator, must operate on a range, not an exact value. When you store your authenticator you must record it several times. Based on this, the system develops an acceptable range for your authenticator. To successfully authenticate, subsequent attempts must fall within that range.

Biometric systems suffer from many shortcomings. First, with the exception of typing cadence, they require hardware devices on every client, some of which can be quite intrusive.

Second, biometric systems are imprecise and a close match is all that is needed. If, for some reason, your biometric authenticator has changed, you will fail the authentication. For instance, if you use voice recognition you may not get in if illness or fatigue affects your voice.

Third, many people consider biometric authentication very intrusive. Having extremely personal details such as fingerprints stored on a computer system is not to many people's liking.

Fourth, many security experts consider biometrics oversold. The companies in the business of selling biometric systems often make impossible claims. For example, a company making a software solution that measures typing cadence claims to protect customers against keystroke loggers, making stolen passwords worthless. But this is impossible. The user must still type the password on the client, and a keystroke logger on the client could just be augmented to capture all the same information that the biometric software is capturing. This information could then be easily replayed to successfully authenticate.

Fifth, there is a common perception that biometric systems are secure because they are inherently a part of the user and cannot be left lying around the way passwords written on a sticky note can. However, this ignores the fact that biometric authentication sequences can not only be captured, but the tokens themselves are also most definitely removable. There have already been recorded instances of thieves making off with biometric authenticators.

Finally, there are relatively few choices for biometric authenticators. For example, in a system using fingerprints you only have 10 choices. If one of them is compromised or lost you have nine left to choose from. This makes cycling your authenticators difficult because you will run out relatively soon, and a weekend of ill-conceived do-it-yourself handiwork may very well prevent you from accessing your computer on Monday morning. As capturing and replaying credentials is a real risk this is a threat not to be discounted.

For all these reasons, Windows does not natively support biometric authentication. Third parties do produce add-on software for biometric authentication, and Microsoft also sells a fingerprinting device, although this latter device is clearly labeled as a non-enterprise grade security device. However, for all the reasons stated previously, these are not enterprise-class authenticators and should not be used in enterprises or to protect sensitive personal or corporate information. For enterprise use, smart cards and passwords can be far more secure, flexible, and easily integrated into ordinary business practices. The remainder of this chapter will focus on those two technologies.

Password Storage

Smart cards rely on certificates. (For more information about certificates, see Chapter 10, "Implementing Active Directory Certificate Services.") The smart card itself holds the secret portion of the certificate. The authentication system, in this case an Active Directory domain, holds the public portion. Therefore, when you use smart cards, no are secrets stored on the domain controllers (DCs) that need protection. This makes smart cards simpler in some ways than passwords to manage.

> As a practical matter, most systems that use smart cards escrow the secret keys in a central location. Windows includes that functionality as well. By doing so you gain the ability to access any secrets protected with smart card credentials, for example, for forensic purposes. However, it also means that you now have a sensitive secret to protect.

Passwords, in virtually every implementation available today, are shared secrets. The secret the user uses to log on with is the same as the one the authentication server uses to authenticate the user's access. This means that passwords are sensitive secrets and must be protected.

In early computer systems, passwords were simply stored in clear-text in a text file. The passwords in those systems were never really meant to keep people out because only a small group of people had access to the system in the first place. They were mostly used to control which environment you received. Eventually, however, the passwords in the password file were encrypted or hashed.

Encryption and Hashing

Encryption is based on the word cryptography, which, literally, means "hidden writing." Encryption is the process of using cryptography to hide writing, or to convert something from a readable form—typically called clear-text or plaintext—into an obscured form, typically referred to as the ciphertext. Decryption is the reverse operation—converting something from ciphertext to plaintext.

While encryption uses cryptography to convert something into unreadable but reversible form, hashing is a closely related function that converts plaintext into unreadable and irreversible form. A hash can, for example, be used as a checksum to compare to plaintexts. If they both generate the same hash, you have reasonably good assurance that they are identical. A hash is also typically far smaller—proportional to the plaintext—than a ciphertext. Therefore, hashes are very well suited to uses like password storage.

Most Unix-based systems still use this exact form of password storage, with two slight modifications. First, the password file, typically stored in /etc/passwd, now contains no password hashes but just user names and IDs. The actual hashes are stored in the shadow password file—for example, in /etc/passwd.shadow. While the password file itself is world-readable, the shadow file is readable only by super users.

Second, because password hashes were originally world-readable in the /etc/passwd file, they had to be protected against comparison attacks. Imagine a situation in which you and I both have user accounts on the same computer. My password is "pas$word!" and,

by sheer coincidence, you create the same password. With a straight hash, we would both have the same password hash stored in the /etc/passwd file. I could search the file for my hash, and then search for any other accounts with the same hash. If I found any, I would know that they had the same password I had. This is an unacceptable situation. The solution is to add a randomly generated salt to the password before hashing it. A *salt* is simply a random value that is added to the password before hashing it and then stored in clear-text in the password database. This way, even if two passwords are identical, they will have different salts and therefore different hashes. The process is shown in Figure 4-1.

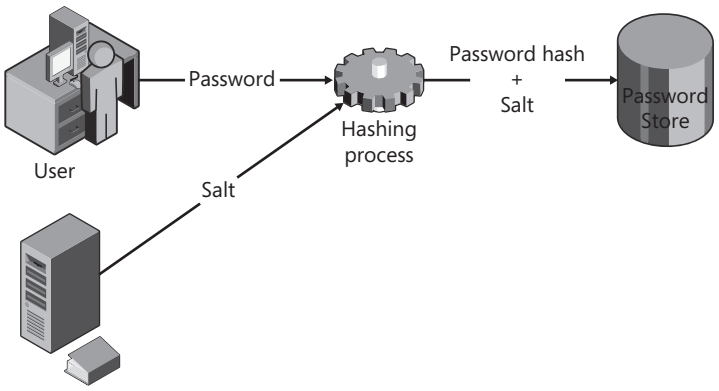

Figure 4-1 By salting the password before storing it the password file is protected against comparison attacks.

Windows uses variants on all these techniques to store its password. In the following sections I will cover the four primary ways Windows stores passwords used to authenticate users to Windows itself.

LM Hash

The LM hash is not actually a hash at all, although it has some of the same properties. It is a one-way function, and is usually referred to internally as the LMOWF (LanManager One-Way Function). In Windows Vista and Windows Server 2008 the LM hash is not stored by default, nor is it used by default during a network authentication. However, on down-level computers it is typically both stored and transmitted by default. Therefore, knowing how the LM hash works is worthwhile. Both Windows Vista and Windows Server 2008 can be configured to store or authenticate with the LM hash, but this is not recommended because of weaknesses in the algorithms.

Direct From the Source: LM Hash History

The LM hash was first used by Microsoft in its LAN Manager network operating system, the last version of which was released in the early 1990s. LAN Manager ran on top of IBM's OS/2 operating system. When Windows NT was first released in 1993 it was imperative that the new operating system interoperated with LAN Manager so that organizations that had invested in LAN Manager did not suddenly find that their investments were useless. Windows NT also provided a smoother upgrade path. However, this also meant that even though Windows NT supported far better security structures than LAN Manager, security concerns in Windows NT were caused by LAN Manager design decisions made in the mid-1980s. In 2006 Microsoft shipped the first operating system that disabled the LAN Manager password hashing mechanism by

default, although it can still be enabled. It took 13 years to deprecate it.

Jesper M. Johansson, former Senior Program Manager for Security Policy, Windows Security MVP

The LM hash is created using a large number of relatively complicated steps, shown in Figure 4-2. The process starts when a user creates a new password. The password is immediately converted to all uppercase. In other words, passwords stored using the LM hash are case-insensitive.

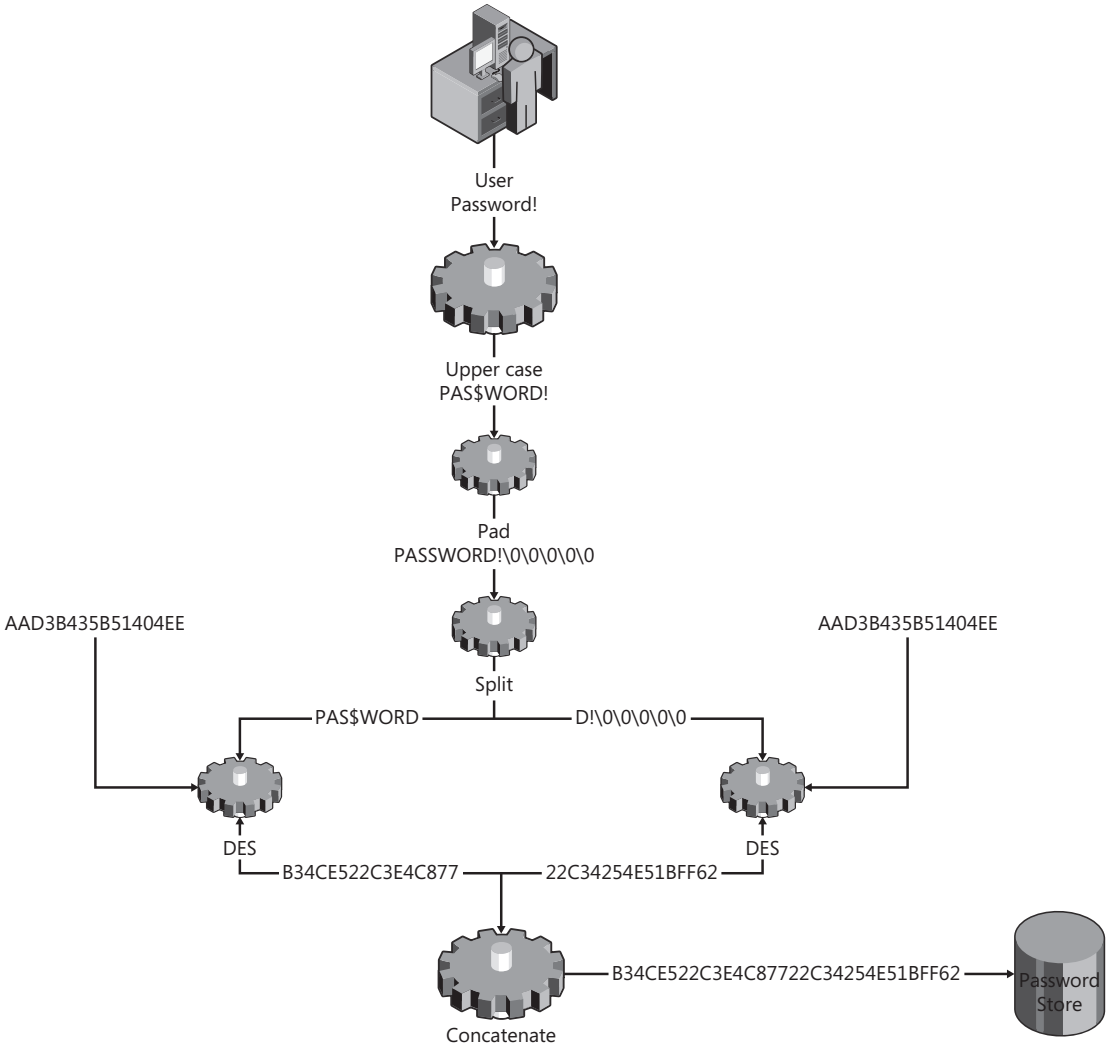

Figure 4-2 The LM hash is created using a series of complicated steps.

After the password is converted to uppercase it is padded out to 14 characters. If the password is already longer than 14 characters, it could theoretically be truncated at this point, but in practice, the process just fails and no LM hash is generated if the password is longer than 14 characters.

Next the password is split into two 7-character chunks. This is because they will now be used as a key in a Data Encryption Standard (DES) encryption, and the Data Encryption

Algorithm (DEA, the algorithm used in DES) operates on 56-bit chunks. These chunks are used as the key to encrypt a fixed value.

Finally, the results of the two DES operations are concatenated and the results are stored as the LM hash. The hash is stored either in the Security Accounts Manager database (if the password is for a local account on a stand-alone computer or a domain member) or in the DBCS-Pwd attribute of the user object in Active Directory.

This explains why an attacker is able to deduce how long a person's user name is just by looking at the hash. If the second half of the LM hash is AAD3B435B51404EE, the second half of the password is blank and the password is no longer than 7 characters. If both halves are AAD3B435B51404EE, the password is entirely blank.

NT Hash

When Windows NT first came out in 1993 a new password storage method was introduced. This mechanism is far simpler, as shown in Figure 4-3.

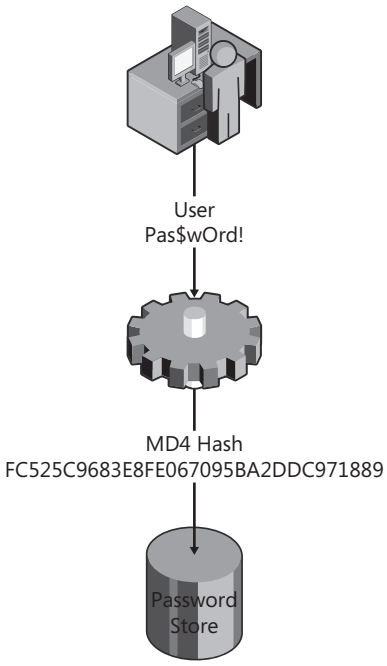

Figure 4-3 The NT hash is a straight MD4 hash.

The NT hash, or NTOWF as it is referred to internally, is stored either in the SAM or in the Unicode-PWD attribute of an AD user.

Note that neither the NTOWF nor the LMOWF are salted. Windows has never salted passwords for the simple reason that the password databases were never readable to others, so the lookup issue was never particularly interesting as an attack vector. To read the databases you have to be an administrator in the first place, meaning you have already fully compromised the computer or domain. Furthermore, shared-secret authentication systems have a very interesting property that we shall discuss shortly.

Password Verifier

If you have worked in a Windows Active Directory environment before you probably noticed that you can carry a domain-joined laptop computer with you and authenticate to it using a domain account even though you are not connected to the domain. This particular bit of magic is thanks to something called the password verifier. The password verifier, often referred to as cached credential outside of Microsoft, is a local copy of your domain password hash that you can use to log on locally. In operating system versions prior to Windows Vista, it was created using the process shown in Figure 4-4.

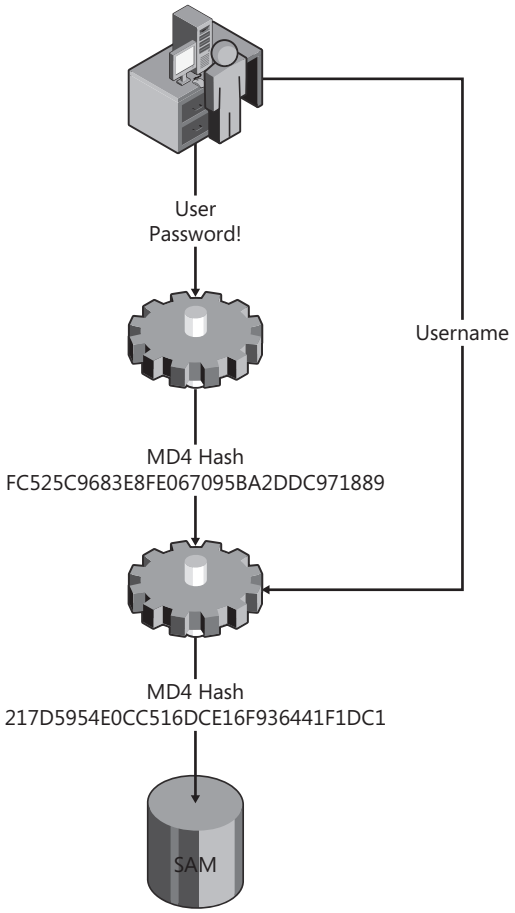

Figure 4-4 In down-level versions the password verifier was simply a hash of a hash, salted with the user name.

In recent years attackers have focused in on the password verifier and started creating tools to crack it. While it is a salted hash of a hash, and therefore quite difficult to crack, cracking it is possible if the password is not very strong. To combat this, in Windows Vista and Windows Server 2008 the calculation for the password verifier was modified, as shown in Figure 4-5.

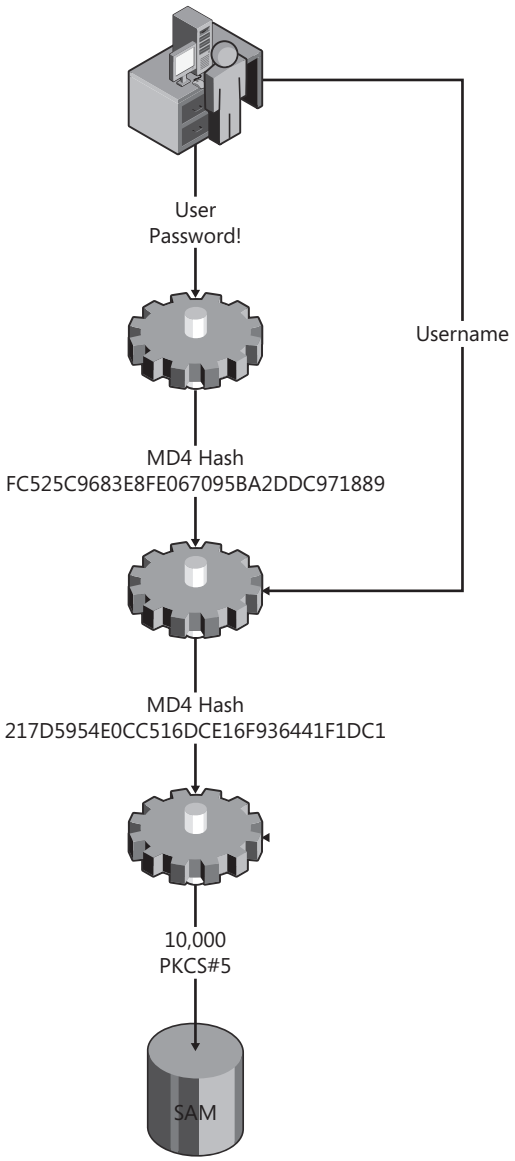

Figure 4-5 The password verifier is far stronger in Windows Vista and Windows Server 2008 than in prior versions.

While there is no way to protect weak passwords, the improved password verifier calculation makes for a much stronger verifier. By running the old verifier through 10,000 PKCS #5 operations, a brute-force cracker would only be able to compute about 10 tests per second. This provides adequate protection against all but the very weakest passwords.

In Memory

When a user logs on interactively or using terminal services Windows caches the user's password hash (the NT hash and, if the computer is configured to store it, the LM hash). The hash is held in a memory location available only to the operating system, and of course, any process that can act as the operating system. When a user tries to access a network resource that requires authentication, the operating system uses this cached hash

to authenticate with. As soon as the user logs off or locks the workstation the memory location is automatically purged.

These hashes have been subject to a fair bit of debate after it was shown that if a domain administrator is logged on, any other user that is an administrator can read that domain administrator's password hash and use it to authenticate to a DC as a domain admin. This really should be obvious to any observer, however, and quite frankly, is putting far too much effort into it. If an attacker has compromised a workstation, it would be far easier to simply install a sub-authentication package, which gets the password in clear-text when it is typed during the logon process. These packages are supported to enable pass-through, single sign-on to non-Windows network devices, just like the NT hash is cached to support single sign-on to Windows devices. Although it would be possible to not support that, most users would rebel at having to type their passwords every time they accessed a network resource.

The problem, therefore, is really not with how Windows caches the NT hash, nor with sub-authentication packages, but rather with operational practices. A domain administrator should never log on interactively to a workstation used by a user with local administrative privileges unless that user is as trusted as all the domain administrators. By following this simple principle, you can keep this legitimate functionality from becoming an attack vector. For more information on managing this, see Chapter 12, "Securing Server Roles."

Reversibly Encrypted

Finally, Windows has an option to store passwords reversibly encrypted. When a password is stored reversibly encrypted, it can be reversed to plaintext. Obviously this means that no cracking is needed. Storing passwords reversibly encrypted is disabled by default, and is generally only needed in two circumstances. First, it is required if you need to use certain older authentication protocols for remote access, such as the CHAP protocol. Second, it is required if you want to perform advanced analysis on your passwords after they are set. For instance, some organizations want to go through and analyze whether passwords contain certain words. Those organizations must store the passwords reversibly encrypted.

To enable reversible encryption, or check whether it is still disabled, use the Group Policy editor, as shown in Figure 4-6.

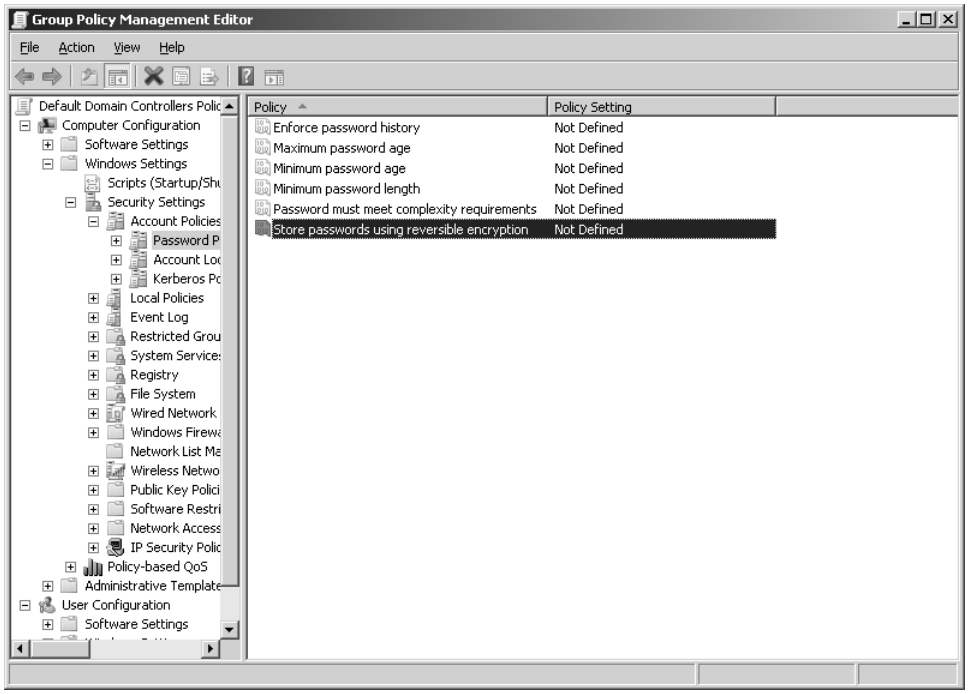

Figure 4-6 To configure a computer or a domain to store passwords reversibly encrypted use the appropriate Group Policy setting.

The vast majority of organizations do not use reversible encryption, and as clients are upgraded to support more secure authentication protocols, there should be fewer and fewer reasons to do so. However, reversible encryption is another way Windows can store passwords, and it is important to know that it is there.

> Many people cringe when they hear that Windows can store passwords reversibly encrypted. After all everyone knows that storing passwords in plaintext is bad. However, this really misses the point. In every password-based system today, *passwords are plaintext-equivalent*! Password-based systems use shared secrets. In the authentication process, the only secret used is the one that is stored on the authentication server. If an attacker gets hold of the authentication server's password database, he has everything he needs to authenticate. The only thing he needs to do now is insert himself at the appropriate step in the authentication process so that he can send the shared secret instead of the password it is derived from. Currently several tools are freely available on the Internet that do this with Windows authentication across the network.
>
> The fact that passwords are plaintext-equivalent is not a security problem by itself. It only becomes a problem when an attacker obtains a password hash. However, as you should realize by now, those are fairly well protected in Windows. If an attacker manages to obtain a password hash, he has already compromised the computer as much or more than he would be able to with that password hash! In other words, that password hash gives him no additional privileges on an already compromised computer.
>
> If passwords are reused across a network, however, it is possible that an attacker can further a compromise using the password hashes. Furthermore, because password hashes are cached in memory, an attacker may be able to obtain domain administrative credentials from a member computer if a domain administrator is logged on. This, however, is largely an operational problem related to how you run your network. If you follow the advice in Chapter 12, you will adequately protect yourself against that vector.

Authentication Protocols

So far we have discussed how passwords are stored on Windows. However, perhaps even more important is how they are used. Passwords are authenticators—they are used to authenticate a user to a computer. If the user is logging on interactively to a local account, the flow is quite simple:

4. User uses the Secure Attention Sequence (SAS, also known as the "three-finger salute," or just Ctrl+Alt+Delete) to bring up the logon dialog box. This causes the Local Security Authority Sub-system (LSASS) to spawn a new session and load WinLogon in that session. WinLogon in turn loads the LogonUI.
5. User types in the user name and password.
6. The WinLogon process takes the password, hashes it to an NT hash, looks up the user name in the local SAM, and compares the NT hash to the one that is stored for the user. If the two match, the logon is successful.
7. If sub-authentication packages are installed on the computer, the logon information is passed to those for additional processing. Otherwise, user32.exe is invoked and the user's environment is loaded.

This process is quite straightforward because there is a secured channel all the way from LogonUI, which takes in plaintext credentials, to the comparison of credentials. However, when authentication is taking place over the network it becomes a bit more complicated because you have to worry about how the authentication claims are transferred between the client where the user is sitting and the authentication server that hosts the accounts database. On Windows, this can take many forms, which I'll discuss in the following sections.

Basic Authentication

Basic authentication is the simplest of all forms of authentication. It just transmits the raw logon information across the network. In other words, the user name and password are sent across the network. This is also sometimes referred to as the Password Authentication Protocol (PAP). Basic authentication is quite common in older network protocols such as Telnet, FTP, POP, IMAP, and even in HTTP. Today it may be used, for example, in the RPC/HTTPS connector mechanism used to connect an Microsoft Office Outlook client to an Exchange server across the Internet. In that case the credentials are traversing inside an encrypted channel up to the Exchange Server or the ISA Server, whichever is terminating the connection. However, other than across an encrypted channel such as this basic authentication should be avoided.

Challenge-Response Protocols

Challenge-response protocols are designed to obviate the need to transmit a password in clear-text across the network. They all essentially operate the same way, shown in Figure 4-7.

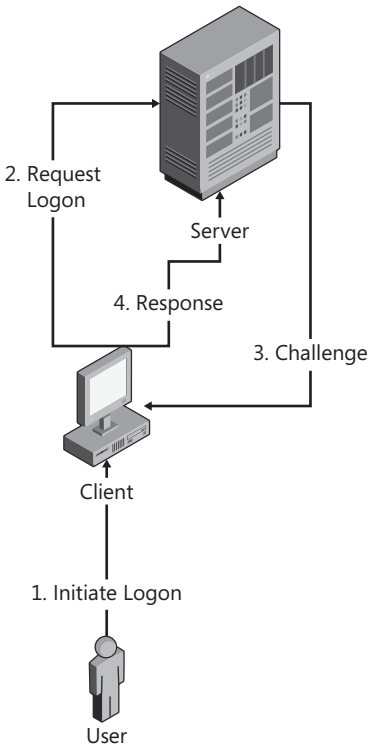

Figure 4-7 All challenge-response protocols are based on the same model.

The basic model for a challenge-response protocol is that a user initiates a logon, upon which the client makes a request to the server. The server creates a challenge, which often is just a random value, and sends this to the client. Meanwhile, the client has collected the user's credentials. The credentials are then combined with the challenge in a cryptographic operation. The result becomes the response. The actual implementation may differ, but the basic structure is always the same.

Digest Authentication

Digest authentication is not a native protocol in Windows. It is used primarily with Internet Information Services (IIS) in accordance with RFC 2617for Web-based authentication, and also with some third-party Lightweight Directory Access Protocol (LDAP) servers. Digest authentication is designed as a replacement to basic authentication. It is considered relatively weak, and makes some security tradeoffs on the authentication server.

The challenge-response sequence in a digest authentication is composed as follows:

1. The server generates a random nonce and sends it to the client.
2. The client computes an MD5 hash of the user name, authentication realm (domain in Windows), and password.
3. The client computes an MD5 hash of the method and the digest URI.
4. The client computes an MD5 hash of the result of operation 2, the server nonce, a request counter, a client nonce, a quality protection code, and the result from operation 3. This is the response value provided by the client.
5. The server computes all the same values.

The main concern with digest authentication happens in step 5. As you can tell, the client response is computed with the actual password, not a hash of the password. This means that to validate the client's response the server must have access to the clear-text password. Hence, if you want to support digest authentication, you must configure your domain to store passwords using reversible encryption.

LM and NTLM

Contrary to digest authentication, both LM and NTLM are considered native protocols in Windows. They are very similar, differing mainly in the hash used to compute the response. LM was first used in the LanManager product mentioned earlier. NTLM was designed as a replacement and released with Windows NT 3.1.

LM and NTLM are used in authentication in workgroups in Windows NT–based operating systems. They are also used in a domain environment if either the client or the server is not a domain member, or if the resource being accessed is specified using an IP address as opposed to a host name. Otherwise, Kerberos is used in Active Directory domains. The reason LM/NTLM must be used when accessing a resource using an IP address is that Kerberos is based on fully qualified domain names (FQDNs) and there is no way to resolve one of those from an IP address because each host can have multiple aliases.

The authentication flow in LM and NTLM is typically conjoined. The aggregate flow is shown in Figure 4-8.

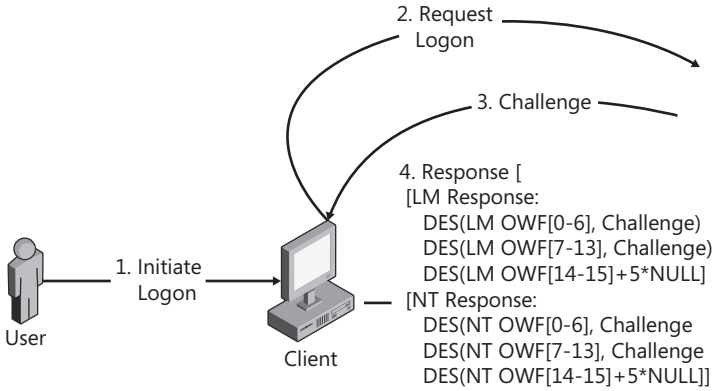

Figure 4-8 The LM and NTLM protocols are typically sent together.

All Windows NT–based operating systems prior to Windows Server 2003 worked as shown in Figure 4-8, sending both the LM and NTLM responses by default. In Windows Server 2003 only the NTLM response was sent by default, but both were accepted inbound. Starting with Windows Vista and Windows Server 2008, this has changed.

NTLM v2

Starting with Windows Vista, and also with Windows Server 2008, both LM and NTLM are deprecated by default. NTLM is still supported for inbound authentication, but for outbound authentication a newer version of NTLM, called NTLMv2, is sent instead. Technically speaking LM is also accepted for inbound authentication but neither Windows Vista nor Windows Server 2008 store the LM hash. Therefore there is no way for them to authenticate an inbound LM response.

The authentication behavior is controlled using the LMCompatibilityLevel registry setting, shown in Group Policy as Network Security: LAN Manager Authentication Level. See Figure 4-9.

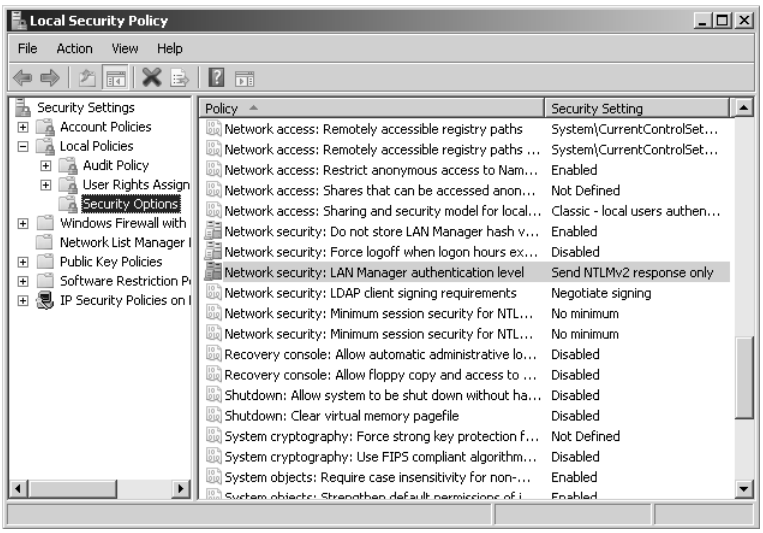

Figure 4-9 The LAN Manager Authentication Level setting governs the authentication behavior in non-domain authentication.

The default value for LMCompatibilityLevel in Windows Vista and Windows Server 2008 is 3, or Send NTLMv2 Response Only. Table 4-1 and Table 4-2 show how the possible values affect a computer when acting as the client and authentication server, respectively. It is important to recognize that the settings in Table 4-2 only relate to the server that performs the authentication, which is the one that contains the user accounts database. Any intermediate servers simply pass on the request to that server.

Table 4-1 Impact of LMCompatibilityLevel on Client Behavior

Level	Group Policy Name	Sends	Accepts	Prohibits Sending
0	Send LM and NTLM Responses	LM, NTLM NTLMv2 Session Security is negotiated	LM, NTLM, NTLMv2	NTLMv2 Session Security (on Windows 2000 below SRP1, Windows NT 4.0, and Windows 9x)
1	Send LM and NTLM—use NTLMv2 session security if negotiated	LM, NTLM NTLMv2 Session Security is negotiated	LM, NTLM, NTLMv2	NTLMv2
2	Send NTLM response only	NTLM NTLMv2 Session Security is negotiated	LM, NTLM, NTLMv2	LM and NTLMv2

Table 4-1 Impact of LMCompatibilityLevel on Client Behavior				
Level	**Group Policy Name**	**Sends**	**Accepts**	**Prohibits Sending**
3	Send NTLMv2 response only	NTLMv2 Session Security is always used	LM, NTLM, NTLMv2	LM and NTLM

Table 4-2 Impact of LMCompatibilityLevel on authentication server behavior				
Level	**Group Policy Name**	**Sends**	**Accepts Inbound**	**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

NTLMv2 is a much improved version of NTLM. It also uses the NT hash. However, it also includes a client challenge in the computation. The aggregate flow is shown in Figure 4-10.

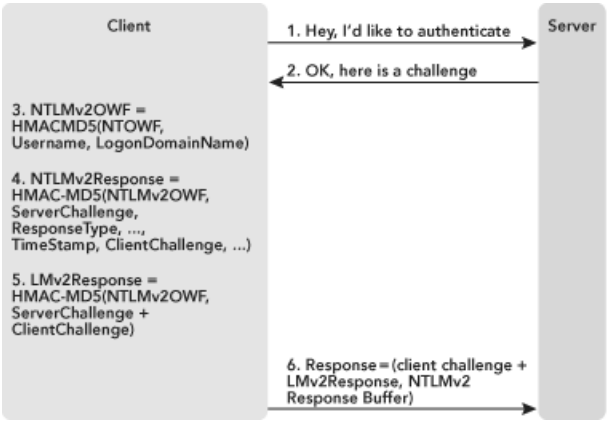

Figure 4-10 The NTLMv2 protocol uses HMAC-MD5 and a client challenge.

As Figure 4-10 shows, the NTLMv2 protocol uses not only a client challenge, but also computes two HMAC-MD5 message authentication codes to create the response. It also includes a time stamp that mitigates replay attacks. Figure 4-10 also shows an LMv2 response, which is included in the response. The LMv2 response is a fixed-length response as opposed to the NTLMv2 response. It is included to provide the ability for pass-through authentication with down-level systems, such as Windows 95. Those systems did not support NTLMv2 natively, but did pass through the LM response. When NTLMv2 was first designed, those systems were prevalent, and they would strip pieces of the variable-length NTLMv2 response, breaking the authentication. To prevent this problem the LMv2 response was included in the LM response field. Because it has the same length as the LM response it is passed through to the authentication server unharmed and can be used to complete the authentication. Today it is still passed. However, the authentication server always starts out the authentication process by seeing whether there is an NTLMv2

response that validates successfully. If there is, the authentication succeeds. Therefore, while the LMv2 response still exists, it is rarely used for authentication.

NTLM++

Around the Windows 2000 time-frame, Microsoft added another NTLM-family protocol to Windows. This one does not have an official name. In some places in the implementation it is referred to as NTLM2, to contrast with NTLM3, which is actually NTLMv2. In other places it is called NTLM++. It was never documented, but was discovered externally by several people, including Eric Glass, Christopher R. Hertel, and Hidenobu Seki, and is even picked up by the Ethereal network traffic analyzer, which refers to it as NTLM2 Session Security. This is because it was always observed in conjunction with LMCompatibilityLevel set to 1, which enabled NTLMv2 Session Security. NTLM++ was added to make certain man-in-the-middle attacks more difficult, while retaining the ability to pass through authentication when connecting to down-level clients. In a sense, NTLM++ is an intermediate step between NTLM and LMv2/NTLMv2.

When NTLM++ is used the LM response field is populated with a client challenge instead of the LM response, as shown in Figure 4-11.

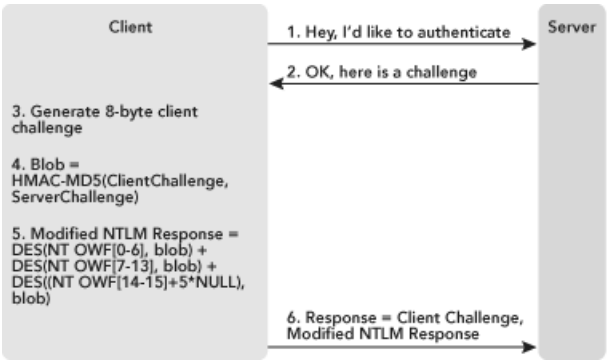

Figure 4-11 The NTLM++ protocol includes a modified NTLM response and a client challenge.

The NTLM response field contains a modified NTLM response calculated exactly the same way as the original NTLM response, but using an HMAC-MD5 of the client challenge and the server challenge as the challenge, instead of just the server challenge.

NTLM++ is used whenever NTLMv2 Session Security is enabled. Starting with Windows 2000 Security Rollup Pack 1, all computers will automatically send the NTLM++ response on the first attempt. This means that starting with that release, the effective LMCompatibilityLevel setting is actually 1 on all computers.

For reference, NTLMv2 Session Security also includes stronger computation of session keys that are used by applications that request session security after the connection is set up.

Kerberos

Kerberos is used in domain environments when host names are used to connect. This is most of the time, unless the user specifically requests a connection to an IP address. Like the NTLM family, Kerberos is implemented as a Security Support Provider (SSP) and Kerberos also uses the NT hash for authentication, but any similarities with the other protocols really end there.

Kerberos is designed to provide authentication both for the user who is trying to connect and the authentication between the client and the server. This is quite a departure from NTLM, which does not provide the user with any assurance that the server is the one she thinks it is. Kerberos is also designed with the explicit assumption that the network is hostile; that all traffic is being intercepted by the adversary; and that the adversary has the ability to read, modify, or delete any traffic sent across the network.

To accomplish all this, Kerberos relies on encryption as well as time synchronization. By default in Windows, the synchronization between client and server must be within five minutes of each other. You can modify this setting if you are in an environment with high potential skew. To do so, change the maximum Tolerance For Computer Clock Synchronization value in Computer Configuration\Windows Settings\Security Settings\Account Policies\Kerberos Policies in a GPO that applies to the computers for which you want to change the time skew.

To understand how Kerberos works, let's analyze the exchange shown in Figure 4-12, which shows how a user logs on to a workstation and then requests a file from a file server.

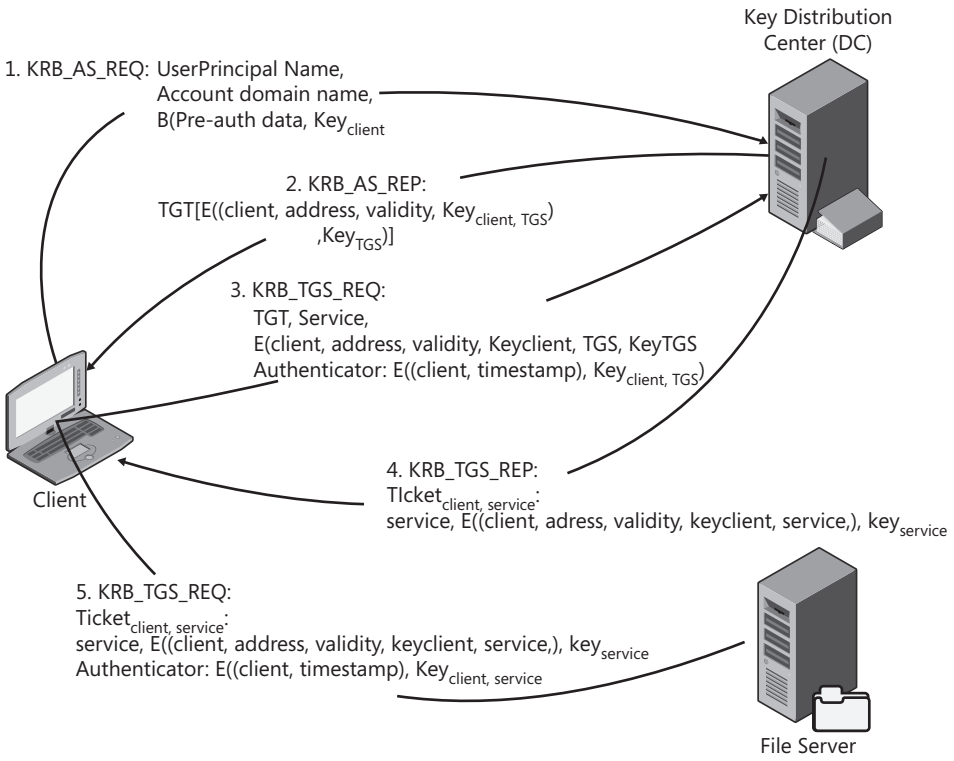

Figure 4-12 This exchange occurs when a computer starts and requests a file from a file server.

The exchange in Figure 4-12 consists of the following parts:

1. After the computer starts it creates some pre-authentication data, consisting of, among other things, a time stamp. This pre-authentication data is encrypted using a key derived from the computer's password. It is then packaged in a KRB_AS_REQ (Kerberos Authentication Service Request) packet and sent to the Authentication

Service (AS) which resides on the Key Distribution Center (KDC), which, as it turns out, is the DC.

2. The AS constructs a Ticket Granting Ticket (TGT) and creates a session key that the client can use to communicate with the Ticket Granting Service (TGS), which also resides on the DC. This key is denoted with $Key_{client,TGS}$ in Figure 4-12. It is transmitted to the client encrypted with the client's own public key. This message is sent back as the KRB_AS_REP.
3. The client now sends a KRB_TGS_REQ message to the Ticket Granting Service (TGS) on the KDC to request a ticket for the file server. This request has the TGT in it, and also includes the service the client wants to access and information on the client encrypted with the TGS public key. The KRB_TGS_REQ includes an authenticator, which is essentially a time stamp encrypted with the session key the client shares with the TGS.
4. The TGS responds with a KRB_TGS_REP message that includes a ticket for the service the client requested. It contains the same information the client sent in the KRB_TGS_REQ, but this time is encrypted using the server's public key. In other words, the client cannot read this data. The TGS also creates a session key that the client can share with the server and encrypts it with the session key the client shares with the TGS.
5. Finally, the client sends its ticket for the service to the server. The client information, along with the client-server session key, is encrypted using the server's public key, and the message also includes the client's authenticator, which is encrypted using the shared session key.

When a user logs on to the client, the same process is repeated, but this time the messages include user information. The Kerberos client sends another KRB_AS_REQ, but encrypts the pre-authentication data with a key derived from the client's password—or rather, the client's NT hash. The KDC validates the authentication based on that information. In the KRB_AS_REP the client receives a TGT that the user can use to contact the TGS. The TGT includes session keys for the KDC along with Security Identifiers (SIDs) for the user and all the groups the user is a member of. From then on, the client will use the user's TGT for requests made on behalf of the user.

Kerberos is clearly a rather complicated protocol, but it has proven remarkably robust in Windows. It also proves to be extensible in that the user's pre-authentication data can just as easily be encrypted with some secret not derived from a password. This happens in smart card–based authentication.

Smart Card Authentication

A smart card is, in most cases, a credit card–sized device that contains a memory chip. These devices have many uses. For example, they are used to provision a phone's identity in the Global System for Mobile communications (GSM) cellular telephone system and its derivatives. Smart cards may also be used to authenticate to Windows. In that case they contain an X.509 certificate. (See Chapter 10 for more information about certificates.) The certificate contains a private key, and the corresponding public key is stored in the user object in Active Directory.

When the user authenticates using a smart card, WinLogon will ask for a PIN code instead of a password. It then contacts the smart card provider and provides it with the PIN code along with the pre-authentication data. The smart card provider uses the PIN code to access the smart card, which will encrypt the pre-authentication data that the Kerberos SSP will use in the KRB_AS_REQ message. From then on, most things happen the same way in a smart card logon as in a normal password-based logon, with one major difference: If the user logs on with a smart card, she never provided a password. This means that if the user tries to access any resources that cannot use the Kerberos system, the computer must prompt her for a password. To avoid that, Windows handles passwords a bit differently in smart card–based logons.

Smart Cards and Passwords

All accounts have a password hash stored on the DC. Even if a user logs on with a smart card, a password hash is still there. In fact, even if the user is required to log on with a smart card there is a password hash. When you configure an account to require smart card logon, the DC will actually create a random password, hash it, and store it in the user object.

When a user logs on with a smart card, the KDC actually provides the client with the user's password hash during the logon process. These credentials are sent encrypted with the client's public key. The Kerberos SSP on the client will decrypt them and cache them in the same way it would cache them if the user had entered them at the logon prompt. These credentials are then used to log on seamlessly to computers that, for whatever reason, cannot be reached using Kerberos. This means that even with smart card logon required, the hashes are still exposed on the client to any rogue software that happens to run as an Administrator. Using smart cards does not protect the password-based credentials any more than password-based logons do. Therefore, all the same cautions apply against the attacks we shall discuss next.

Attacks on Passwords

At this particular juncture, it is worth taking a little detour into attacks, if for no other reason than that so many people are concerned about them. The primary concern with respect to passwords is obviously bad guys getting at them. Once they have them, or some representation thereof, the question is how they use them. Let's start by investigating how a bad guy can obtain a password, or some form of it.

Obtaining Passwords

Bad guys have several ways to get hold of your passwords. The following sections list them in order of ease of attack and prevalence (roughly speaking).

Ask For Them

An astonishing number of people, up to three-quarters in some surveys, are willing to part with their passwords in trade for something they value more, like chocolate.

Capture the Passwords Themselves

The most fruitful, simplest, and possibly most common way to attack passwords today is to use a keystroke logger to capture them in plaintext as they are being entered. There are many different kinds of keystroke loggers. An innocuous option is using a hardware device that mounts between the keyboard and the computer and has onboard memory to hold all keystrokes. It can be surreptitiously installed or removed in a matter of seconds. A software program, commonly found in malware and spyware today, will capture all keystrokes, not just passwords. Some of these include an automatic upload feature to a Web site or an IRC channel. Others include a small Web server that the attacker can use to retrieve the goods. However, the simplest and most direct route for an attacker to capture only passwords is to write an authentication package. Windows, like any other industrial-strength operating system, includes functionality for third parties to extend its authentication subsystem to authenticate to other network devices. An attacker can, with just a few application programming interface calls, write an authentication package that will receive all passwords in plaintext when a user logs on. The package can be augmented with the same features as a more general keystroke logger, but generates far less noise. Both of the software options require administrative privileges to install, meaning that the computer must be completely compromised to get to them. Physical compromise would also be sufficient to install one of these, and it is quite telling that keystroke loggers are now found regularly on public access computers, especially at conferences.

Capture the Challenge-Response Sequence

It is rare that passwords are passed on the network in any form today, and even rarer with plaintext protocols such as FTP, POP, and Telnet. However, the attacker can often capture both the challenge and the response and attack the combination. It requires more calculations than attacking ordinary hashes, but can be very fruitful if the password is weak.

Capture the Hashes

This is the quintessential attack that everyone worries about. If an attacker has access to the password hashes, he can crack them or use them in some other way. There are several ways to crack them, as we shall see shortly. The most common way to capture the hashes is to compromise the authentication server that stores the passwords. As you will see in Chapter 14, "Securing the Network," the more dependencies you have in your network, the easier this attack is to perpetrate.

Another option—less common but equally valid—is to compromise a computer where someone is already logged on. When a user logs on, as I mentioned earlier, Windows caches that user's NT hash in memory. An attacker with complete control over the computer can retrieve that hash and use it in the same way as any other hash. Again, this is a problem largely related to your operational practices. If you do not expose sensitive hashes on computers that are less sensitive (and hence less secure) you will not have this problem. In addition, if a criminal manages to compromise a computer to this extent, she can easily capture the plaintext password as well, as we will see in the next section.

It is important not to lose sight of the fact that in every case that involves compromise of actual hashes, the bad guy has defeated all the security systems and has complete control over at least a system that will provide him with advanced access, and probably to a

system that holds all the secrets—the DC. In other words, if a bad guy has hashes to crack, you have already been severely hacked and bad guys with password hashes should be the least of your concerns. Regardless of whether the bad guy manages to use the hashes directly or crack them, your network is beyond repair already. Your only solution is to rebuild any compromised computer—including the entire network if a domain or enterprise admin account could be compromised—from scratch or a backup that is provably not compromised.

Guessing Passwords

Finally, the bad guy can simply try to guess passwords. Anyone who has an Internet-connected Windows computer and actually looks at the log files will see attempts at this. Figure 4-13 shows a failed attempt on one of my computers on the day I was writing this chapter.

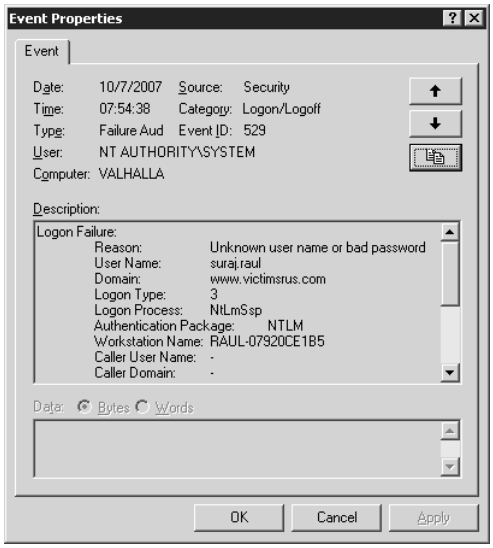

Figure 4-13 Anyone with an Internet-connected Windows computer will get failed logon attempts in their event logs.

Most the attackers use automated password "grinders" that attempt to log on using either Terminal Services or Windows Networking (Server Message Block, or SMB). The logon attempt in Figure 4-13 is actually an Internet Information Services logon attempt, which I know only because the host does not respond on either Terminal Services or SMB across the Internet.

The automated password grinders will typically try common user names, such as Administrator, with a dictionary of passwords. Shockingly, they must be successful enough with that approach to make it worthwhile to continue. Many people argue that you should rename the Administrator account to fool attackers, and some even say to create a decoy account called Administrator. This has absolutely no effect whatsoever. The error message is the same whether an account does not exist with the name Administrator or whether the attacker gets the password wrong. Therefore, from the attacker's perspective, he cannot tell whether you have an account called Administrator. He can only tell that he did not get in. You can assure yourself that he will not get in simply by setting a reasonably strong password. For example, if the password is 15 characters long, and seemingly random (meaning that it seems random from the attacker's point of view) the

attacker will have to try 542,086,379,860,909,058,354,552,242,176, or so, times before he succeeds. More than likely he will move on before he succeeds in guessing that password.

Leaving Your Passwords Blank

As with all versions of Windows since Windows XP, user accounts with blank passwords cannot log on from the network in Windows Server 2008. This is actually a genius design, which can be used to great effect for the local Administrator account.

In a typical datacenter the servers are locked inside racks. In many cases, not everyone has access to every rack. Only those personnel who need to get into particular servers can get into those racks. The racks themselves are in locked rooms that require badge and PIN access. In that situation, are your servers physically secured? More than likely you would say yes. If so, why not leave the password blank for the built-in Administrator account? The only people that can use it are the ones that get past the badge scanner, have the PIN code to the right room, and the key to the right rack. More than likely, if someone has all of those, he belongs in there and needs to use that account—and has a way to get at the password should he need to. Obviously, he shouldn't use it on a daily basis, but if everything breaks and he needs to log on as the built-in Administrator, he knows what the password is and can get in very easily. In addition, because accounts with blank passwords are not usable across the network, making the password blank removes what would be a significant security dependency if you used the same password on every server.

Leaving the password blank solves one of the huge problems in network security: how do you keep the admin account from having the same password on every server in the network? It's very difficult to argue that leaving the password blank compromises security in any way at all—when you have adequate physical security. Unfortunately, it is probably going to be far more difficult to convince an ill-informed security auditor that leaving the password blank is more secure than setting the same 8-character password he requires on every single server. If you promise to try though, I'll do my part.

Using the Captured Information

Assuming the bad guy has captured something, how does he go about using it? If he has captured a plaintext password, the answer is relatively straightforward. He just needs to find somewhere to type it in. However, if he has captured a challenge-response sequence, or a password hash, the problem is slightly more complicated.

Cracking Passwords

The most common attack is to crack the password. By "crack" in this case, we generally mean that the attacker creates a password hash or a challenge-response sequence based on some trial password and compares it to the hash or response that he captured. If the test succeeds, the trial password is the right password.

As you have seen earlier in this chapter, several additional computations are involved in computing a challenge-response sequence as opposed to computing a straight hash. It stands to reason, therefore, that cracking a captured challenge response sequence takes significantly longer than simply cracking a password hash. On commonly available

hardware today you could compute anywhere from 3 million to 10 million hashes per second to try with, while you could compute only a third as many challenge-response pairs. If the bad guy only has the password verifier, he will be able to compute only 10 per second, rendering them effectively uncrackable unless the password is exceptionally weak.

Several approaches to cracking passwords speed up the process. An attacker can try with a dictionary of common words, or common passwords, such as the lists in (Burnett, 2005). The attacker can also try a brute-force attack using all possible passwords of some given character set. The character set can be greatly trimmed. My own research has shown that 80 percent of the characters used in passwords are chosen from a set of only 32 characters. Finally, the attacker can try a hybrid approach in which the test password is based on some dictionary with characters permuted. For example, the attacker may try common substitutions, such as using "!" or "1" instead of "i", "@" instead of "a" or "at", "3" instead of "e" and so on.

Pre-computed Hash Attacks

Pre-computed hash attacks are very simple in concept. The first common use of them was in Gerald Quakenbush's Password Appraiser tool from the late 1990s. The tool shipped with several CDs full of password hashes. Several years later, Cedric Tissieres and Philippe Oechslin developed Ophcrack, which cracked LM hashes using pre-computed hashes, but used a time-memory tradeoff to reduce the amount of storage space required to hold the hashes. Rather than storing all the hashes, they stored only a portion of them along with all the passwords that created that hash. At run time the cracker would simply look up which set of passwords possibly matched the hash it needed to crack, compute the hashes for all the options, and compare them to the hash. This was significantly slower than Password Appraiser, but many orders of magnitude faster than brute-force cracking. Zhu Shuanglei implemented the same technique in the immensely popular Rainbow Crack tool, which can crack almost any hash out there. Pre-computed hash attacks are often referred to as Rainbow Cracks or Rainbow Table Attacks after that tool.

Pre-computed hash attacks have created immense media buzz, and many, many people, and many security "experts" have opined about how bad they are and how they work only because Windows is flawed and how Windows should be fixed to prevent them. Typically these claims are accompanied by statements about how (of course) other operating systems had the foresight to protect against these attacks. These characterizations are gross simplifications that fail to account properly for either history or reality.

First, Windows is not flawed in that it does not take into account pre-computed hash attacks in its design. It is true that use of a salt in the password-hashing mechanism would combat pre-computed hash attacks. However, it simply was not (and still is not) an interesting threat to protect against. Furthermore, nobody should be lured into thinking that the designers of competing operating systems had the foresight to protect against these attacks. Salts were added to protect against the fact that the password file was world-readable. Pre-computed hash attacks were not relevant when those platforms were designed. Keeping gigabytes, or even terabytes, of password hashes was not particularly interesting when the computer had 16KB of core memory and a tape drive.

Second, it makes no sense whatsoever to start salting Windows password hashes to protect against pre-computed hash attacks. Consider how the authentication protocols

work. If you change the hashing mechanisms, you must also introduce a new authentication protocol because the old ones rely on the old hashes. The last time a new authentication protocol was actually retired was in Windows Vista, when LM was retired. That took 13 years from the introduction of its replacement. Changing the hashing mechanism to only add a salt would certainly stop pre-computed hash attacks. However, it would take 13 years or so before the old NT hashes were gone. Furthermore, because password hashes are plaintext equivalent, with or without a salt, it would not solve the real problem.

Pass-the-Hash Attacks

Password hashes are plaintext equivalent. This should be eminently clear by now. The secret used by the server to verify the client's identity is the same secret the client uses to prove its identity. If a criminal manages to capture that secret, he can simply use it to prove his identity, *without any knowledge of the password used to create that secret*.

This is a crucial point. If we can accept the fact that password hashes are plaintext equivalent the way we think about things changes. First, we can immediately see why replacing the current NT hashes with salted ones is meaningless, because the salted ones are also plaintext equivalent. Second, we can also see that the core problem is not password hashes, but bad guys with access to them when standard challenge-response protocols are used. The only real technical solution is to move away from challenge-response protocols to public key protocols. However, this requires a substantial change to all platforms and is unlikely to happen any time soon.

Therefore, the real solution is to stop bad guys from getting at password hashes. To do that we need to minimize the exposure of password hashes and we need to ensure that we adequately protect our authentication servers. Chapter 14 discusses these topics in depth.

Protecting Your Passwords

Every one of the attacks we have discussed so far can be mitigated by either using better passwords, or managing and operating your network more securely. Chapter 14 goes into depth about how to manage and operate the network more securely. Obviously, because password hashes are plaintext equivalent, using strong passwords will not mitigate all the attacks outlined so far. However, it will have a significant impact on many of them.

What constitutes a strong password? The answer is: a long password! No single factor is more important than length when it comes to password strength. Table 4-3 shows how long a password composed from *n* characters chosen randomly from a set of 32 characters resists both guessing and cracking attacks.

> The password resilience data presented in Tables 4-3 and 4-4 are based on a theoretical attacker than can guess 600 passwords per second or crack 7.5 million passwords per second. These numbers are significantly greater than what can be achieved today both with respect to password guessing and cracking captured challenge-response pairs.

Table 4-3 Password Attack Resilience for 32-Character Character Sets

Length	Guessing Resilience in Days	Cracking Resilience In Days
6	10	0
7	331	0
8	10,605	1
9	339,355	27
10	10,859,374	869
11	347,499,971	27,800
12	11,119,999,080	889,600
13	355,839,970,558	28,467,198
14	11,386,879,057,845	910,950,325

As you can tell from Table 4-3, the strength of the password goes up dramatically the longer it gets. A 14-character password composed of randomly chosen symbols from a known 32-character character set resists guessing for more than 31 billion (!) years. Even an 8-character password would be impossible to guess in a reasonable time frame as long as the attacker cannot rely on heuristics. The 14-character password resists a cracking attack for 2.5 million years, but of course it is still plaintext equivalent, so resistance to cracking is really only relevant in the case of a captured challenge-response sequence.

The question, however, that many want answered is how important the character set is in password strength. Obviously, the larger the character set the attacker has to contend with, the stronger the password is. However, the effect is nowhere near as drastic as length. Table 4-4 shows the same data as Table 4-3, but for passwords composed using a character set consisting of 95 characters.

Table 4-4 Password Attack Resilience for 95-Character Character Sets

Length	Guessing Resilience in Days	Cracking Resilience In Days
6	7,090	1
7	673,551	54
8	63,987,310	5,119
9	6,078,794,461	486,304
10	577,485,473,802	46,198,838
11	54,861,120,011,233	4,388,889,601
12	5,211,806,401,067,100	416,944,512,085
13	495,121,608,101,375,000	39,609,728,648,110
14	47,036,552,769,630,600,000	3,762,924,221,570,450

As Table 4-4 shows, the passwords based on the set of 95 characters are certainly stronger than ones based only on 32. However, a 6-character password from the 95-character set is weaker than an 8-character password from the 32-character set. Realizing that difficulty dealing with complexity is often a human weakness, many people would probably have a simpler time remembering an 8-character password composed of a small set of commonly used characters than a 6-character password composed of characters they hardly ever use. These numbers can be used to develop an appropriate strategy for different people, depending on how they think. However, it is clear from this data that if we can simply get users to use longer passwords, we can solve a lot of password-related problems.

Fundamentally, passwords are a perfectly acceptable, very convenient, comprehensible, and simple-to-implement authentication mechanism. The only flaw in the equation is that people are not good at remembering passwords. If we could only remove the people that use them from the problem, passwords are probably the best way to authenticate to a system. Fortunately, we can.

Managing Passwords

Left to their own devices, people will not pick very good passwords. Yet we need them to pick longer ones to protect us. To reconcile that dilemma, we need to rethink some old concepts that many hold as truth.

Use Other Authenticators

First, a password that the user does not know is better than one the user does know. If you use smart cards and configure the system to require smart card logon, every account will still have a password, but it will be a long and random password. Its hash can still be stolen from any computer that the user logs on to, providing that malware running as the operating system is present on that computer, but the password, for all practical purposes, can never be guessed.

Record Passwords, Safely

For those of us who cannot require smart cards, however, we need to live with the fact that the user must know the password. To help them remember their passwords, the Chinese invented this marvelous technology, called paper, in the second century CE. Users should record their passwords. Currently most organizations have a password policy that requires 8-character passwords, and they must have three different character sets in them. The result? Users pick passwords like "Seattle1", which, if you check it, complies with the policy. "Test1234" complies as well, as does "Password1", "Passw0rd" and "Pa$$word". If you were given the choice, wouldn't you rather have a user carry a little piece of paper in her wallet with the words "Get a skinny tall latte before work!" on it? If a bad guy got hold of that note, the user would know pretty quickly and could take appropriate steps to reset the password (assuming you have told her how to do that), and what exactly would the bad guy do with the note? Which system does password belong to? Is it even a password, or is it a shopping list? A password the user can write down is far easier to manage than one she has to memorize after typing twice. And, for all the other passwords we use every day, you can use an electronic password management tool, such as Password Safe (*http://passwordsafe.sourceforge.net*). Which is really worse: a weak password that the user can remember after typing it twice, or a very strong one that is securely recorded? What exact exposures are we worried about here?

Now imagine that you told your users that they could keep the password on a note until they remembered them, and after that they had to put the notes in the secret disposal bin, or eat them, whichever they preferred. If you do that, your users may even let you set the password policy to require 10 characters and live to tell the tale.

Stop Thinking About Words

Notice that in the preceding discussion the imaginary password was "Get a skinny tall latte before work!" That is not a password. It is a passphrase. Nothing says that passwords have to be words any more. The very term—password—is wrong. Windows will happily accept up to 127 characters, chosen from the entire keyboard (including the space bar) in a password. Recall also that we concluded earlier in the chapter that the longer the password is the stronger it is. Using a passphrase is the perfect way to add length to your password. Passphrases are long and therefore strong. They are simpler to type and easier to remember than contorted strong passwords, such as hG%'3m.^. Simply put, passphrases just work the way people are used to working already. People are used to thinking about words. I have seen seven-year-old children use passphrases successfully. In addition, a phrase such as the latte one is far, far longer and many orders of magnitude stronger than the contorted strong password. If we assume a worst-case scenario, in which the attacker knows that we use passphrases, knows that this one is seven words long, and even knows the dictionary of words it was composed from, it could still take millions of years to guess—even if the attacker uses an attack tool that permutes words as opposed to characters. The set of possibilities is so many times larger than the set of characters on a keyboard. If you wanted to improve the strength a little, do one of the common substitutions somewhere. For example, replace an "a" with "@", or an "l" with a "1", or an "e" with a "3", or an "o" with a "0". In our 8-character password we'll be lucky to get one of those substitutions, merely doubling the possibilities. In the case of the passphrase, we get 12 possible substitutions just with those 4 substitution options, increasing the total search space 4,096 times! Passphrases are immensely powerful as an authenticator.

Set Password Policies

Finally, you should of course have password policies. You need both written organizational policies and technically enforced policies. The written policies are beyond the scope of this book, but should include policies that are realistic—in other words, don't ban writing passwords down. You should also have an implementation guideline that helps people understand how to pick passwords.

Technical policies should be enforced domain-wide, and also on member computers if you use local accounts on member computers. They should require complexity, long passwords (10 or more characters are highly preferable) and should cycle the passwords regularly. However, tie the policies together logically. If you require 10-character passwords, it is almost certainly acceptable to keep them for 6 months or a year. With 8 characters you should change them every 3-6 months. With anything fewer than 8 characters, you should consider changing passwords monthly.

Policies can be managed with Group Policy (GP). Figure 4-14 shows where in GP the settings are.

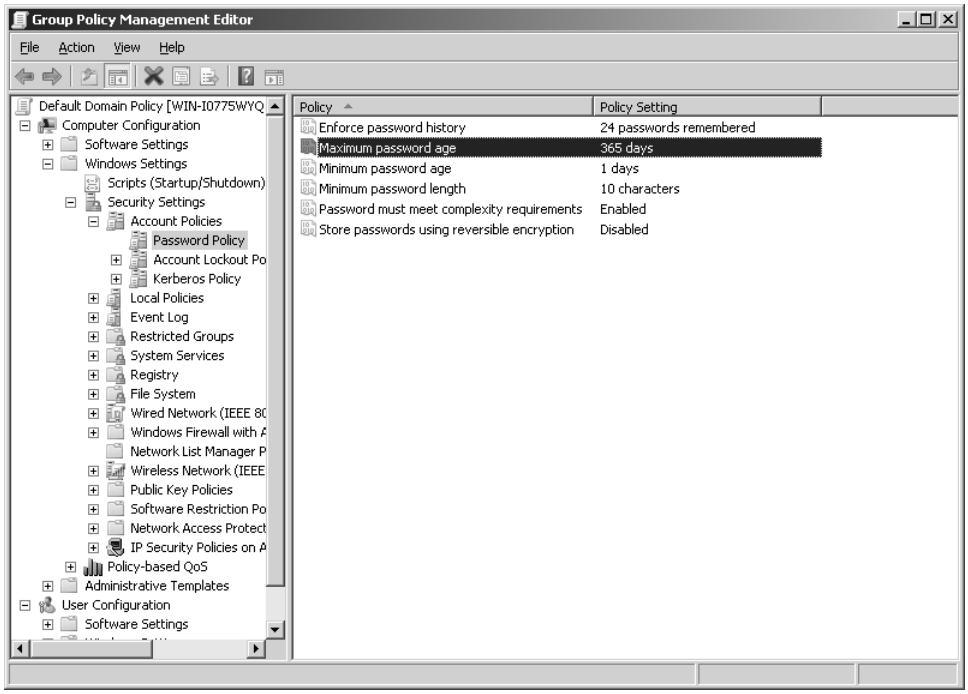

Figure 4-14 Password policies can be managed with Group Policy.

Password policies applied with domain scope apply to domain accounts when the password is set from a workstation. They do not apply to accounts whose passwords are reset in Active Directory Users And Computers (ADUC). This is done so that administrators can reset a user password there without having to create incredibly complex passwords that will be changed in a few minutes anyway. Password policies applied to an organizational unit scope apply to local accounts on all member computers in that OU.

Fine-Grained Password Policies

A persistent request from customers has been the ability to manage password policies so that different users in the domain have different password policies. In Windows Server 2008 this is finally possible, with *fine-grained* password policies. Fine-grained password policies are available in all editions of Windows Server 2008, but only if the domain functional level is Windows Server 2008. In other words, you must first upgrade all your domain controllers to Windows Server 2008 before you can use it.

The primary purpose of fine-grained password policies is to apply stricter settings to privileged accounts and less strict settings to the accounts of the normal users. In other cases you might want to apply a special password policy for accounts whose passwords are synchronized with other data sources.

Fine-grained password policies apply only to user objects, or inetOrgPerson objects if they are used instead of user objects and global security groups. Fine-grained password policies are implemented using a password settings container (PSC) under the System container of the domain. (See Chapter 9, "Designing Active Directory Domain Services for Security," for more details on Active Directory.) The PSC stores one or more password settings objects PSOs that hold the actual policies. Figure 4-15 shows a PSC with two PSOs.

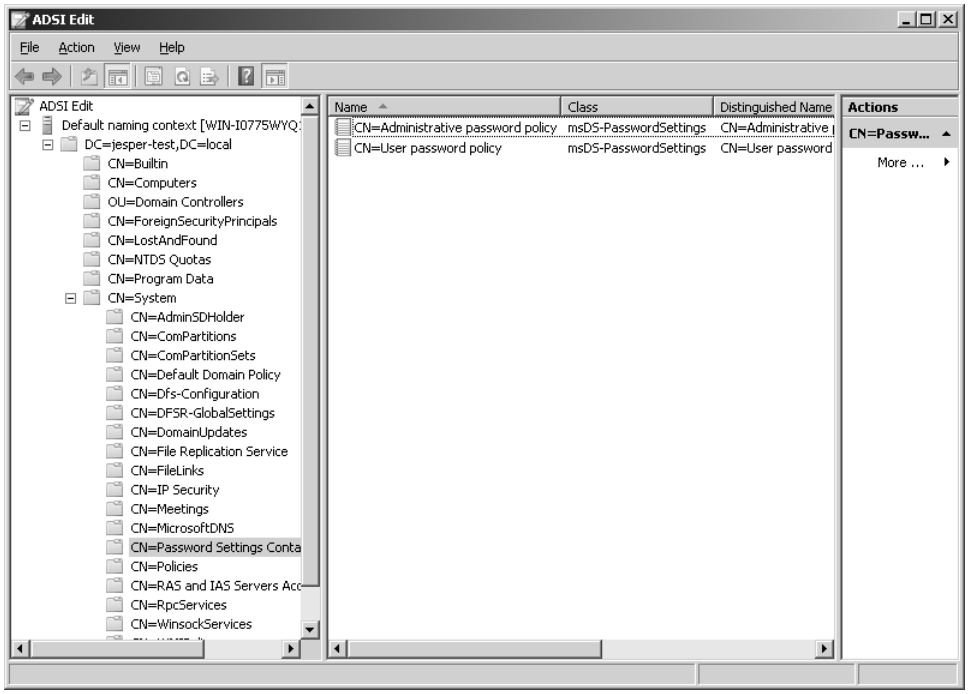

Figure 4-15 This domain uses one password policy for administrators and another for users.

Unfortunately, Microsoft did not provide a very good user experience for managing fine-grained password policies in Windows Server 2008. There is a step-by-step walkthrough available, but the steps are somewhat complicated. To configure a separate password policy for administrators, follow these steps:

1. Run the ADSI Edit tool by running adsiedit.msc.
2. Connect to your domain by right-clicking the ADSI Edit node in the left-hand pane and selecting Connect To. Type in the name of the domain.
3. Expand the domain, expand the DC node, navigate down to CN=System, and select CN=Password Settings Container.
4. Right-click CN=Password Settings Container, select New, and then select Object.
5. Select the *msDS-Password Settings* object, as shown in Figure 4-16.

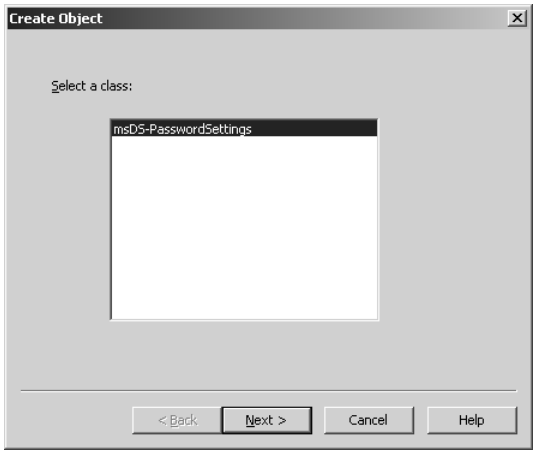

Figure 4-16 To create a fine-grained password policy, you need to create a new *msDS-PasswordSettings* object.

6. Name the new object something memorable, such as **Administrative password policy**.
7. Click Next, and set the precedence value for this object. This value governs which policy takes precedence if two policies apply to the same user. The lowest precedence wins.
8. Walk through the rest of the wizard and set values for all the items. The possible values are listed in Table 4-5.

Table 4-5 Fine-Grained Password Policy Values

Attribute name	Description	Acceptable value range
msDS-PasswordSettingsPrecedence	Defines which policy takes precedence if more than one policy applies to a given user. The policy with the lowest precedence wins.	Greater than 0
msDS-PasswordReversibleEncryptionEnabled	Whether passwords are stored with reversible encryption. False means they are not.	FALSE / TRUE
msDS-PasswordHistoryLength	How many passwords the system remembers for a user. Practically speaking, this means the user cannot reuse a password until he has chosen at least this many different ones.	0 through 1024
msDS-PasswordComplexityEnabled	False if password complexity is not required for these user accounts. True if password complexity is required.	FALSE / TRUE
msDS-MinimumPasswordLength	The minimum length for a password. Note that passwords can be up to 255 characters long, but older systems support entering only 127-character passwords.	0 through 255

Table 4-5 Fine-Grained Password Policy Values

Attribute name	Description	Acceptable value range
msDS-MinimumPasswordAge	The minimum age that a password must be before it can be changed again. Setting this to some reasonable value, such as a day or two, ensures that a user cannot cycle through the password history automatically and change the password back to the one that she just had. This value, (and all other time values) is entered in DAYS:HOURS:MINUTES:SECONDS format. Hence 02:00:00:00 is two days.	(None) 00:00:00:00 through msDS-MaximumPasswordAge value
msDS-MaximumPasswordAge	How old a password can be before it must be changed. To have passwords that never expire, use the value *(Never)*. Otherwise, set a date in the standard time value format, such as 180:00:00:00.	(Never) msDS-MinimumPasswordAge value through (Never) msDS-MaximumPasswordAge cannot be set to zero
msDS-LockoutThreshold	How many tries a user gets at the password before it is locked out. To disable account lockout, set this to 0.	0 through 65535
msDS-LockoutObservationWindow	The time interval used to calculate the number of incorrect password tries. If this value is set to 00:00:30:00, for example, the user gets *msDS-LockoutThreshold* tries in 30 minutes and then the counter is reset.	(None) 00:00:00:01 through msDS-LockoutDuration value
msDS-LockoutDuration	How long the account remains locked out before it is automatically unlocked. To require administrative unlock set it to *(Never)*.	(None) (Never) msDS-LockoutObservationWindow value through (Never)

9. After you configure the lockout duration you will see the screen shown in Figure 4-17. At this point you need to configure which users this PSO applies to. To start that process, click More Attributes.

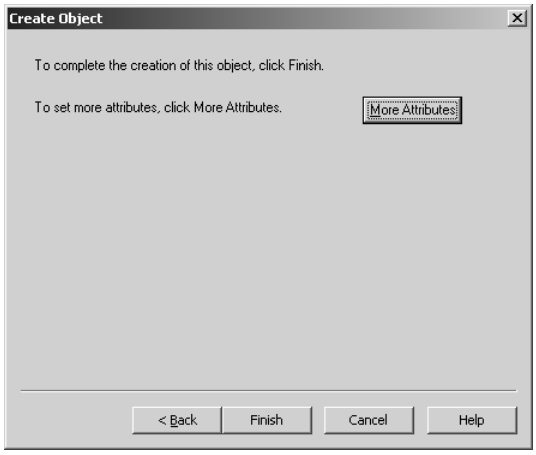

Figure 4-17 When you get to this screen, configure who the object applies to.

10. From the Select A Property To View drop-down list, select msDS-PSOAppliesTo.
11. Type in the distinguished name (DN) of the user or the global security group you want this policy to apply to. For example, to apply it to the Domain Admins group, use the following syntax, replacing the DC attributes with your domain information:
    CN=Domain Admins,CN=Users,DC=jesper-test,DC=local
    The net result is shown in Figure 4-18.

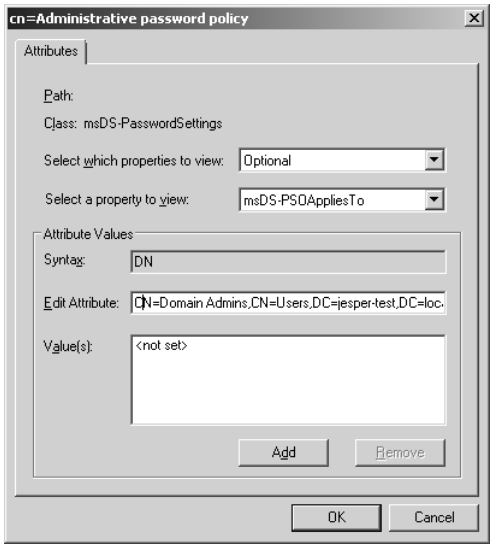

Figure 4-18 You use the *msDS-PSOAppliesTo* attribute to apply the policy to a group or a user.

By now you have probably already figured out that Microsoft kind of ran out of time to build good tools to manage fine-grained password polices. Fortunately, there are some options out there. Joeware has a command-line tool available at:
http://www.joeware.net/freetools/tools/psomgr/index.htm

A GUI tool is available for PowerGUI, based on the PowerShell in Windows Server 2008:

http://powergui.org/entry.jspa?externalID=882&categoryID=46

Another free GUI tool is available from Specopssoft:

http://www.specopssoft.com/wiki/index.php/SpecopsPasswordPolicybasic/SpecopsPasswordPolicybasic/

Precedence and Fine-Grained Password Policies

I mentioned earlier that a precedence value is associated with fine-grained password policies. This value is for resolving conflicts when two PSOs apply to a single user. The policies are not merged, so there must be some way to resolve the conflict. The resolution works as follows:

1. If only one PSO is linked to the user object, that PSO is the resultant PSO. If more than one PSO is linked to the user object, a warning message is logged to the event log and the one with the lowest precedence value is the resultant PSO.
2. If no PSOs are linked to the user object, the system compares the precedence values of all the PSOs linked to groups the user is a member of. The PSO with the lowest precedence value is the resultant PSO.
3. If neither of these methods results in a PSO being the most preferred, the default domain policy applies.

Summary

Additional Resources

1. Burnett, M. Perfect Passwords: Selection, Protection, Authentication. (Syngress, 2005)
2. Johansson, J. M. *Protect Your Windows Network.* (Addison-Wesley, 2005)
3. Johansson, J. M. "The Most Misunderstood Security Setting of All Time." *TechNet Magazine*.
4. Kent, J. "Malaysia car thieves steal finger." http://news.bbc.co.uk/2/hi/asia-pacific/4396831.stm
5. Microsoft Corporation. "Server Core Installation Option of Windows Server 2008 Step-By-Step Guide." http://technet2.microsoft.com/windowsserver2008/en/library/47a23a74-e13c-46de-8d30-ad0afb1eaffc1033.mspx?mfr=true
6. Microsoft Corporation. "Step-by-Step Guide for Fine-Grained Password and Account Lockout Policy Configuration." *http://go.microsoft.com/fwlink/?LinkID=91477*
7. Wagner, M. "The Password Is: Chocolate" http://informationweek.com/story/showArticle.jhtml?articleID=18902123

Part III

Common Security Scenarios

Chapter 14

Managing Security Dependencies to Secure Your Network

Jesper M. Johansson

I am often asked how to protect workstations on a network. More specifically, the question is framed against the latest attack-du-jour that was demonstrated at some conference. For example, many people are extremely concerned about USB Flash Memory—those incredibly handy little finger-sized, solid-state memory devices that are now available in capacities bordering the ludicrous. People are worried about an attack that starts with the attacker inserting a USB Flash Drive into a computer, or causing the user to do so. The USB Flash Drive is laden with malware that either automatically—or with minimal user interaction—executes malware on the computer.

The problem with this preoccupation with USB Flash Drives is that it is an extremely narrow view of a much larger removable-device problem that also includes CDs, DVDs, FireWire drives, parallel port devices (does anyone still have these?), and just about any other orifice on the computer that can be used to access external content. Too often, people are only worried about workstations and not the rest of the network. I believe that the question is not how you keep workstations from getting hacked, but how you keep the rest of the network from falling like dominos once they do. Let's look at the math. If you have 10,000 end users in a network, what are the chances that you can keep all the workstations secure? Let's assume that each of those workstations is up to date with security updates, fully managed, and operated by users who are savvy enough about security to not run malicious content 99.99 percent of the time. Ignore the complete unrealism of these numbers for a moment and focus on the math. With 10,000 workstations, these numbers mean you have a 37 percent chance of having a secure network at any given time. With 20,000 workstations, your chances are about 13 percent. Add in a more realistic probability of each of your workstations being secure, and you will find that the probability of keeping all of them simultaneously secure asymptotically approaches zero as your network grows in size.

Clearly, it is absolutely critical to protect the network as a whole from the compromise of a single workstation. In fact, as an IT manager, I argue that no single thing you can to do improve the security of your operational environment is more crucial than managing dependencies in your network to isolate exposures. One part of this is to restrict communications within your network; Microsoft calls this Server and Domain Isolation. However, that puzzle has some more pieces.

Direct from the Source: Server and Doman Isolation

Server and Domain Isolation is one of the hidden security gems in Windows Server 2008 that's worth taking a closer look at. Although you had the ability to create virtual networks through end-point authentication in previous releases, the work we've done in Windows Vista and Windows Server 2008 makes this even easier to deploy. By

combining IPsec connection security rules with Windows Firewall filters we have given Windows Server 2008 administrators a powerful tool to increase the security on their networks and better safeguard their important data—and all of this can be done without installing new software.

Chris Black, Program Manager, Windows Networking

Many organizations are still overconfident that their perimeter firewall solves most of the puzzle for them. However, perimeter firewalls are virtually meaningless in today's environment. To understand why, take a moment to try to enumerate the entry points into your network. If you have a medium sized or larger network I am willing to bet a very good dinner that the last audit you had found a few egress points you were never aware of. Every computer on a virtual private network (VPN) is a potential ingress point. Every system that you have not updated is a potential ingress point. Every insecure, custom-written piece of software is a potential ingress point. Every misconfigured router, firewall, VPN device, and wireless access point is a potential ingress point.

> The principle of Defense in Depth simply requires that you put significant effort into reducing the impact of a compromise on your network.

One obvious method for addressing the problem of malicious removable devices is to ban everything with the potential to be malicious. This includes more than just USB Flash Drives. We would need to ban all removable devices, including anything that plugs into any device bus in the computer. I've said before that the best way to handle that problem is to use a giant tube of epoxy to plug up every opening you find on the back, front, sides, top, and bottom of the computer.

This approach has a couple of problems. First, your users might ambush you on the way to your car and perform ritual sacrifice on you if you do. Second, they would be right to do so. Many of the aforementioned devices serve legitimate business needs. For instance, it is pretty much universally accepted that the most secure configuration of the BitLocker full hard-disk encryption technology in Windows Vista is to use an external key on a USB Flash Drive. Doing so would be rather difficult if you filled the USB ports with epoxy. I suppose you could glue the USB Flash Drive into the port, but that would sort of defeat the whole purpose of using it for encryption key storage. The same argument can be said for most ports on a computer these days.

The better option, in all but the most sensitive environments, is probably to attempt to manage the risk and contain the exposure. We need to accept a fundamental truth here. The general statement in Law 3 of the "10 Immutable Laws of Security" (see http://www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.mspx?mfr=true) still holds:

> If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.

If an attacker has—or has ever had—access to your computer, that computer must be considered compromised. This kind of attack can even be perpetrated remotely, if the attacker can get you to run malicious code on your computer. Law 1 from the Immutable Laws states that:

> If a bad guy can persuade you to run his program on your computer, it's not your computer anymore.

If we take it as a fact that the immutable laws still hold—and we probably can because they have proven to be remarkably resilient, and it is unlikely they will be proven invalid in any significant way until we fundamentally change how computers work—we cannot rest with a few registry tweaks to reduce the threat posed by removable drives. Clearly, we must use additional layers of protection. In fact, if we simply make the quite reasonable assumption that many of our client computers are either already compromised, or operated by people who do not always have our best security interests front most in mind (or both), we arrive at the conclusion that we need to mitigate their effects on the remainder of the network. This leads us naturally to understand, analyze, and mitigate security dependencies.

Security Alert: On the Efficacy of Security Guides

For the past 15 years or so an unbelievable amount of effort has been devoted to building security guides. I have taken part in building about half a dozen of these over the years. These guides invariably a list of various security tweaks that— according to the authors—you must make to a standard installation of some software to meet some security requirement. The requirement itself is far too often unstated, and many of the guides are merely listings of every possible tweak that the authors thought might have even the most marginal impact on security; most of the time without considering the functionality your computers need to provide or threat environment your computers face. Often the settings recommended by the guides don't actually work on the software the guide is intended for.

The best of these guides make it very clear what the settings do, what application compatibility impact you can expect from them, and what specific threats they mitigate. Yet even the best of these guides spend scant space on the problem of network security at large. The guides are invariably focused on hardening a single computer against attacks, not fully accounting for the environment that computer is deployed in. The fact of the matter is that once the attacker has a foothold in the network, not a single setting in the security guides matters. The fact that account lockout is set to infinite lockout after three bad guesses—aggravating every user in the process—makes no difference to the attacker that has administrative privileges.

Rather than focus your efforts on which tweaks you need to make to your computers, you will get a lot more mileage out of simply accepting that some portion of your network is, and always will be, untrustworthy. The reality is that Enterprise networks today are semi-hostile at best. Let's accept that sad state as fact and move on. We deal with that problem by protecting the network as a whole from the few bad elements. You cannot secure a society by setting down rules and a sturdy wall around the society. You also need police officers. Police officers are basically a function of society's acceptance that some portion of its members refuse to live within the boundaries that have been set for them. Your network is no different.

Introduction to Security Dependencies

A **security dependency** occurs when the security of one computer is dependent on the security of another. This is quite common, and in many cases desirable. For instance, you might have heard that "if your domain controller (DC) has been hacked, your entire network has been hacked." This is a simplistic way of stating that all domain members are dependent on the DCs for their security. If the domain controller is not kept secure, the member computers cannot possibly be kept secure. An attacker who can change the security configuration of the domain can take over any computer in the domain—for example, by adding new accounts to the Administrators group on a member computer. This explains why any so-called vulnerability that allows a system or network to be compromised by an administrator is not really a legitimate security vulnerability. That's because an administrator, by definition, is supposed to have complete access to the system or network he or she is administering.

Dependencies in computer systems are clearly unavoidable. However, that does not mean that they are all acceptable: Some are acceptable and even desirable, while others are unacceptable. Before we analyze the different types of dependencies and how to mitigate them, we need to understand which types of dependencies are acceptable and which are not.

Acceptable Dependencies

Acceptable dependencies can be summed up by the following statement, from *Protect Your Windows Network*:

A less sensitive system may depend on a more sensitive system for its security.

Computers—and systems in general—can be divided into classes based on their security sensitivity. A system that is more sensitive has higher security requirements. while one that is less sensitive needs less security. The specific set of classes in any particular environment is irrelevant to the general discussion; only the fact that there are inherent classifications is important. For the sake of argument, let us assume that we have two classes of systems: workstations and DCs. The DCs, obviously, are far more sensitive than the workstations. If you control a workstation, theoretically you should have access to only the data used on that workstation. However, if you control a DC, you have the keys to the kingdom—you have complete access to everything in the forest. In that case, it is acceptable for the workstations to depend on the DCs for their security. The DCs class is far more sensitive

than the workstations, and must be correspondingly better protected. This is a form of an acceptable dependency.

The same argument can be made for user accounts. It is acceptable for an administrator to compromise data owned by a user. This is what it means to be an administrator in the first place. Administrators have unfettered (although not always direct and obvious) access to the computer and everything on it. If we understand that and manage the computers appropriately, this is not a problem.

Software can be analyzed the same way. A less sensitive piece of software, such as a Web browser, may use and depend on a more sensitive piece of software for its security, such as the operating system itself. That is acceptable. If the operating system has a bug, the fact that the Web browser is now vulnerable to some new problem is really not surprising and is probably rather low on the list of worries. This also helps us understand where bug fixes go. The bug should be fixed as close to the problem as possible, to have the maximum protective impact. Rather than work around the problem in the Web browser, fix it in the operating system. Alternatively, rewrite the Web browser to reduce its dependencies on functionality in the operating system. This latter approach is appropriate if the functionality in the operating system was never intended to be used in the way it is being used, or if the functionality is not designed to protect against the particular attack the Web browser is suffering from.

Unacceptable Dependencies

Unacceptable dependencies should by now be obvious. Again, quoting from (Johansson, 2005):

A more sensitive system must never depend on a less sensitive system for its security.

If we again think in terms of classes of sensitivity, this statement is easily understood. If a compromise of a workstation means that the domain controller's security has been breached, we have a serious security problem on our hands. As mentioned earlier, it is impossible to protect a network if its aggregate security is dependent on the security of every single computer in that network. The likelihood that the network is secure is inversely exponentially related to the size of the network. A network of any reasonable size is, for all practical purposes, never entirely secure. This makes it paramount that more sensitive systems are protected from less sensitive ones.

This argument can easily be extended to user accounts and software. For example, the new Terminal Services client for Windows permits storage of user names and passwords for virtually transparent Terminal Services logon. Those credentials are stored using the Credential Manager API, protected by the credentials used for the primary logon session.

To see how this can create a security dependency, let us analyze the case of a network administrator logging on to his personal workstation. He uses this workstation for e-mail, Web browsing, and other typical information worker tasks. Naturally, he uses a low-privileged domain account for this purpose. At some point during the day he connects to one of the domain controllers to perform some form of management. He uses the Terminal Services client to do this, and elects to store his password to make future connections easier. This results in at least one, possibly two, unacceptable security

dependencies. The first is that his domain administrative account credentials are now protected by his low-privileged information worker credentials. If his low-privileged user account is compromised, his domain administrative user account is also compromised, and thus the entire domain is compromised.

The second dependency results from the fact that he typed a domain administrative credential on a non-domain controller. Unless his personal workstation is protected at least as well as the domain controllers—and that it is hard to believe—we have a dependency situation in which the security of the domain controllers depends on the security of this user's personal workstation. If, for example, a disgruntled employee in the same office has installed a hardware keystroke logger on the network administrator's workstation, the domain administrative credentials are now stored on that keystroke logger. Any time you type a domain administrative credential on a non-domain controller you have exposed to entire domain to any security flaws on the non-domain controller. For instance, if an attacker inserts a removable drive into a computer where a Domain Administrator is currently logged on, or has ever logged on, or will ever log on, that Domain Administrator is compromised, and by extension the entire domain is compromised. It is absolutely imperative that you understand how these dependencies work so that you can avoid letting them compromise your network. It means, for example, that you should be very careful which computers you use to administer sensitive computers in the network.

The foregoing analysis leads us to two very concrete pieces of advice. First, never use a computer to enter, retrieve, process, or store data that is more sensitive than the computer itself. Remember, everything piece of data handled by a computer should be considered accessible to everyone who has ever used that computer, or who will ever use that computer. Saving credentials on a computer whose every user you trust is safe. Saving them on a computer that may be used by untrusted users, or that may have malware installed at some point, is not, for example.

Second, never administer a sensitive computer from a computer that is less sensitive. Practically speaking, this means that you should have dedicated management stations used to administer ultra-sensitive computers, such as domain controllers. Simply using runas, or User Account Control (UAC) does not introduce a sufficient security boundary.

Obviously the same situation can happen with software. For instance, let us say we want to write a very secure Web browser. We want this browser to be far more secure than the built-in browser. In this case we cannot rely on any functionality provided by the built-in browser. In the case of Windows, where the browser implements much of the client-side, Internet-related functionality in the operating system, we cannot use any built-in Uniform Resource Locator (URL) validation functions or any Hypertext Markup Language (HTML) display functionality provided by the operating system, because those are really components of Internet Explorer. If we rely on functionality provided by the built-in browser, we have a security dependency on the built-in browser. Based on our stated objective of being more secure than the built-in browser, this dependency is unacceptable.

Dependency Analysis of an Attack

At this stage, it might be useful to take a quick detour and analyze an attack from a dependency perspective. Earlier we saw what can happen if a malicious removable drive is inserted into a computer. However, it may not be obvious what would happen to the network where that computer lives. Let's assume that the computer in question is domain-joined, as shown in Figure 14-1.

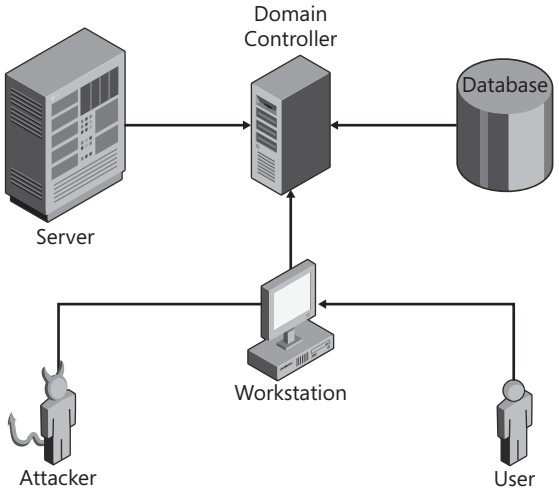

Figure 14-1 Domain dependency graph.

Figure 14-1 shows an ideal dependency graph. The arrows are directional and point in the dependency order: The security of the workstation is dependent on the security of the DC, and the security of the user is dependent on the security of the workstation. The attacker might be able to compromise the workstation, which would compromise any information the user has placed on that workstation, but the compromise would be isolated there.

Let us change the picture a little. Suppose the user logging on to the workstation is a member of the local administrators group on the Server. And suppose that a domain administrator frequently logs on to the server. We now have the dependencies shown with bold arrows in Figure 14-2.

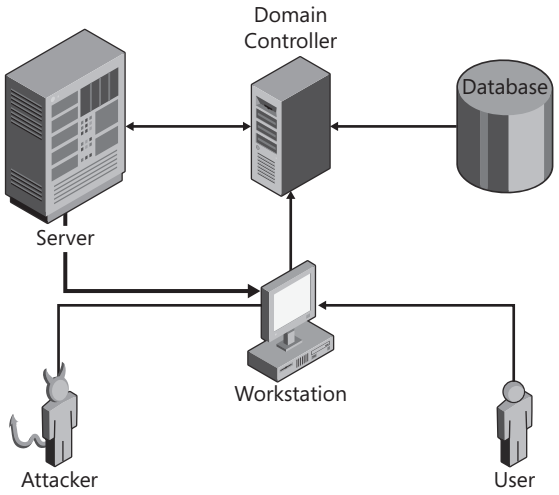

Figure 14-2 Domain dependency graph showing unacceptable dependencies.

As you can tell from Figure 14-2, we can completely violate the security of the entire network by simply changing the assumption of who logs on to which computer. Because a domain administrator logs on to the server, the security of the DC—and hence the domain—is dependent upon the security of the server. This would be acceptable if the server were managed as securely as the DC. However, a user that logs on to the workstation is a member of the Administrators group on the server, making the server dependent on the workstation for its security. Dependencies are *transitive*, which means that the security of the entire domain is now dependent on the security of the workstation, where the user, unfortunately, just ran the attacker's malicious tools. This is why it is so important to manage your dependencies appropriately.

Types of Dependencies

You need to manage many different kinds of dependencies. Some are beyond the scope of this book, such as dependencies inherent in software development. For instance, the security of code on a Web site is dependent upon proper isolation being enforced in the Web browsers that all the visitors use.

However, several different kinds of dependencies are relevant to a network, and in this section I will introduce them and discuss how to mitigate them using standard analysis techniques and actual implementation of these techniques in Windows Server 2008.

Usage Dependencies

The first and simplest kind of dependency is a **usage dependency**. A usage dependency results from usage of computing resources and data in a manner inconsistent with the trust levels of those resources. The first scenario in this chapter—the removable device—is an example of a usage dependency. A user that uses a removable device creates a usage dependency on that device. Whenever a user at one trust level uses a resource at a different trust level there are potential usage dependencies.

There are other kinds of usage dependencies as well. One great example is usage of a single credential in multiple places. For instance, suppose your network is divided into a datacenter forest and a corporate forest. All the users in the datacenter forest also have accounts in the corporate forest. The likelihood that at least one user will have the same user name and password on both of these accounts is extremely high. Yet this violates the entire purpose of having the two forests, which is to ensure that a compromise in one forest does not result in a compromise of another. By using the same password in both places, this particular user has opened a potential pathway between the two. An attacker that breaches a computer in one forest that this user is using can extract the password hash and use it to authenticate to resources in the other forest.

How It Works: Password Hashes Are Plaintext Equivalent

Virtually every computer system in existence today accepts passwords authenticators in at least some situations. On Windows Server 2008—as well as previous server versions of Windows—you can configure a domain to require smart cards for authentication from one or more users. However, as you saw in Chapter 4, "Authenticators and

Authentication Protocols," even when you do so, there will still be a password hash for the user. This hash is transmitted to the client each time the user authenticates to enable automatic access to NTLM-protected resources. This means that an attacker that has access to this hash can access network resources as this user. For more information on this, see Chapter 4.

Access-Based Dependencies

An **access-based dependency** occurs when a user at one trust level accesses a resource in a way that makes the user dependent on the security of that resource. Access-based dependencies result from the access itself, not from usage of a resource or computing construct. Many times they rely on one user or entity trusting another entity that has a security problem.

For example, suppose user Alice accesses a network resource. The network resource is on the server LOKE, which, unknown to Alice, was hacked by Bob earlier that same day. Bob has installed a rootkit on the server that causes authentication to be downgraded to an insecure form of authentication. Alice's computer is running Windows XP, which by default is configured to negotiate authentication to whatever the server and client can agree upon. In doing so, Alice sends a challenge-response sequence that the attacker can replay against Alice's computer, thereby gaining access to her computer with the same privileges she has. To understand this flow, look at Figure 14-3, which shows a normal authentication flow from a client to a server.

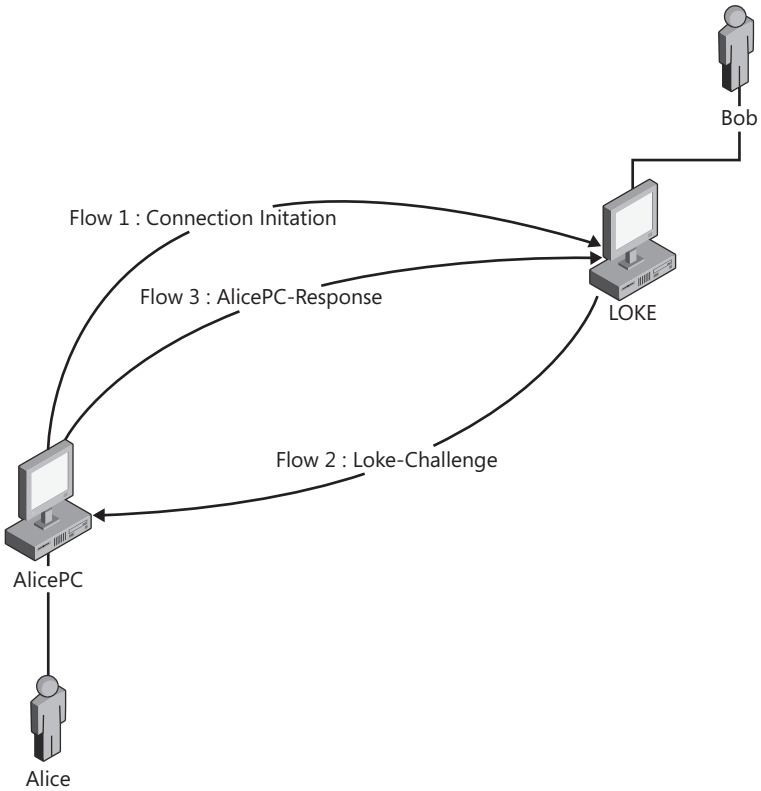

Figure 14-3 A normal challenge response flow from a client to a server.

In the normal flow a client initiates a connection to the server. The server responds with a challenge. The client creates a response to the challenge by performing a cryptographic operation with the authenticator (typically a password hash) and the challenge and returns this as the response. The server performs the same computation and compares the results. If they match, the authentication succeeds.

Now consider Figure 14-4. In this case the client does not respond as it should.

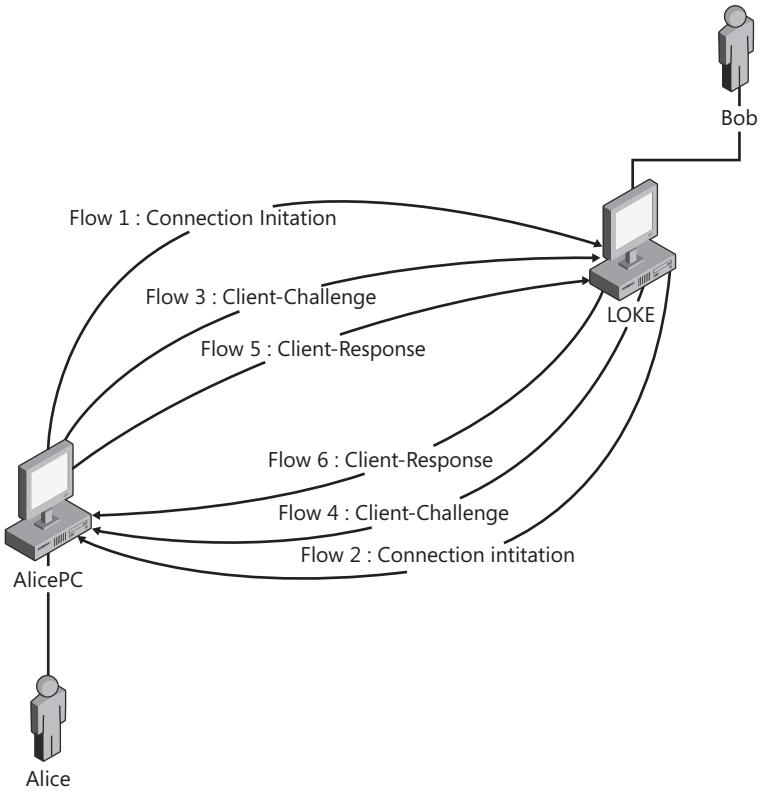

Figure 14-4 Using a reflection attack, the client can "reflect" the server's challenge back to the client to get a valid response.

In Figure 14-4 the client attempts to connect as before. At this point, the server is supposed to send a challenge back. However, the server instead responds with its own connection attempt in flow 2. The client responds to this connection attempt with a challenge (flow 3), which the server subsequently reflects back to the client as the challenge for the connection the client initiated (flow 4). The client now has the same challenge it originally sent back. Unaware that something is amiss, the client computes a valid response to this challenge, which it originally sent, and returns it to the server for the connection the client initiated (flow 5). The server takes this response and returns it as a response to the challenge the client issued for the inbound connection (flow 6). The net result is that we now have two successful connections—one from the client to the server and one from the server to the client. This is known as the **reflection attack**. In Windows Vista and Windows Server 2008 this attack is broken using stateful challenge management. The computer will no longer accept an inbound challenge that matches an outstanding challenge that it sent. In earlier versions the attack can be broken using various security settings.

This attack works because of an access-based dependency. There are other forms of such dependencies. A user might use a public kiosk to access e-mail by using Microsoft Office Outlook Web Access. Public kiosks are among the most malware-infested, untrustworthy, dangerous computers in the world today. Any resource you use on a public kiosk should be considered accessible to any user that has ever accessed that computer in the past or will ever access it the future. There is an access-based dependency between the security of the public kiosk—which you do not control—and any resource you have access to with credentials you use on a public kiosk.

Administrative Dependencies

One of the most common types of dependencies is an **administrative dependency**, which occurs when the same account is used to administer two different resources. For example, when you use a domain administrative account to administer member servers, as shown previously in Figure 14-2, you create an administrative dependency. This may sound a lot like a usage dependency, and it is. However, there is one important difference: administrative dependencies need not be usage-based. Let's say that the Administrators group on Server A includes Teddy, Maggie, and Alex, and the Administrators group on Server B includes Maggie, Jesper, and Jennifer. Maggie might never have logged on to Server A. However, Server B is compromised. When Maggie logs on to Server B, the attacker that compromised Server B has access to Maggie's credentials and can now use them to access Server A.

Service Account Dependencies

Service account dependencies occur when the same identity is used to run a service in multiple places. Suppose you use a network-wide Enterprise Management Solution (EMS). The EMS package includes an agent that runs on all computers to enable remote deployment of software, remote management, and all kinds of other goodness. The agent runs as the _DomainTools account. The _DomainTools account obviously needs to have elevated privileges on all the members to enable this type of remote management. This creates a service account dependency between all the computers where the _DomainTools account has high privileges. If any one of those computers is compromised, all of them are potentially compromised because the attacker now has access to a highly privileged account.

Operational Dependencies

Finally, we have **operational dependencies**. Operational dependencies result from the way a network is operated. For instance, Active Directory creates an ipso facto operational dependency. Any asset within a forest is dependent on the forest for its security. If the forest is compromised, so are all assets within the forest. The forest, in turn, is dependent on all domains in the forest. If a domain is compromised, so is the forest.

> **Security Alert**
>
> The forest, not the domain, is and has always been the security boundary in Active Directory.

Another very common dependency in a network is based on the software distribution system. Very often a single server or a set of Distributed File System (DFS) shares are used to distribute software to computers within the network. If an attacker compromises the server(s) that host the software, all computers that receive software from it are potentially compromised. The operational dependency has created an access-based dependency on the software distribution servers.

Categorizing Dependencies

As you may have noticed by now, the boundaries between the different kinds of dependencies are not always clear, and a single dependency can belong to several categories. For example, a service account used on multiple computers is both a service account dependency and an administrative dependency. The idea behind classifying the dependencies is simply to facilitate thinking about them. One type of dependency is not inherently far worse than another. Individual instances of dependencies might be more or less severe, but that is because of the facts of that instance, not the category of dependency it belongs to. Use the categories as a guideline to help you think about your network, not as a forced framework to plug things into. If you find a different taxonomy to be more useful, use that instead.

The only rule here is that everything you do should improve the security of your network.

Mitigating Dependencies

Finally, many pages into the chapter, we get to the part about how to solve the problem. It has taken this long because the concepts we have discussed so far are barely touched on in the vast majority of security literature, which often does not even mention these issues.

One of the most important techniques for mitigating security dependencies today involves isolating computers that do not need to communicate so that they cannot do so. Microsoft calls this Server and Domain Isolation. To build a strategy to do so is best done in a step-wise process:

1. Define a classification scheme.
2. Model your network.
3. Analyze your network model relative to the classification scheme.
4. Revise the classification scheme as needed and re-analyze.
5. Define an isolation strategy consistent with your risk management strategy.
6. Derive an operational strategy from your isolation strategy.
7. Build a server implementation based on your isolation strategy.

These seven steps are quite a bit more complicated than they might seem. The key is to realize that this is not a single-afternoon project. You really need a far better handle on the structure and usage patterns in your network than what most organizations have. In fact, if you get no further than simply understanding your network better, you have

created significant value. The remainder of this chapter discusses how to use these concepts to design and implement a Server Isolation strategy.

> Before we go any further, it is important to better define the term Server Isolation. When Microsoft first coined the term, it was in conjunction with the term Domain Isolation. **Domain Isolation** simply meant that to communicate with any domain member (with some exceptions) you had to be a domain member. This type of isolation is quite simple and, while valuable, leaves rather large holes by assuming that all domain members are good and nice.
>
> **Server Isolation** is the next step. In Server Isolation each server has its inbound traffic restricted, usually using IPsec, so that only the traffic necessary for the server to fulfill its business purpose is permitted. This provides very good isolation indeed.
>
> When Microsoft and other customers started implementing these isolation mechanisms they discovered that while Domain Isolation was simple in concept, implementing Server Isolation was far easier because IPsec was very difficult to work with in a large network. Therefore, they generally started with Server Isolation.
>
> However, what most observers fail to recognize about Server Isolation is that every Windows-based computer is a server. Every workstation also runs the erver service by default, and if you do not restrict inbound traffic—or even if you use Domain Isolation—you will have a network where every client can attack—I mean communicate—with every other client. Therefore, do not forget to include clients in your Server Isolation strategy.

Step 1: Create a Classification Scheme

The first step in building a server isolation strategy is to classify systems. You can think of network protection mechanisms as residing on a spectrum. Take, for instance, administrative accounts. One extreme of the spectrum is using one account for all purposes, on all computers, by all administrators. On the other extreme you have one account per administrator per task, with the least possible privileges necessary to complete that task per computer. While the former example might be practically possible, it would violate more security principles than we can list. The latter example, while highly secure, is intractable to manage and so cumbersome to use that it will likely be ignored by everyone involved. A similar spectrum exists for all other techniques. For instance, in terms of restricting communications, you can certainly analyze every single computer and restrict access to each one based on exactly what you need to use it for. However, in a network with many thousand computers, this is virtually impossible. You would be hacked long before you completed the analysis.

A far better option is to create a classification scheme. This scheme can be as simple or as complex as you need it to be. The idea is to divide your computers into categories that make sense to your business. Classifications can take many forms. In the military establishment it is common to have a two-dimensional classification scheme, such as that shown in Table 14-1.

Table 14-1 Military Classification Scheme

	Class		
	Unclassified	**Secret**	**Top Secret**
Compartment			Compartment 1
			Compartment 2
			Compartment 3

The military-style classification can be converted to classifying computers quite easily. One variant is shown in Table 14-2.

Table 14-2 System Classification by Role

	Class		
	Public	**Workstations**	**Server**
Compartment	Kiosks	Information Worker Workstations	Domain Controllers
	Infrastructure Servers	Developer Workstations	File Servers
		Admin workstations	Web Servers
			Database Servers
			…

Table 14-2 shows a subset of a computer classification based on the role the systems are fulfilling. No matter how you create your classifications, you almost certainly want to base them on the *role* the computer is fulfilling. The more granular you make the classification scheme—that is, the closer to a single role you can get—the more secure the resulting implementation will be. However, don't go overboard with this classification. First, you will probably need to revise it once you start analyzing your network and realize that you missed something and that some roles that do not cleanly make sense. Second, treat this as a risk management effort. If you are designing a classification scheme for an extreme risk environment, you want more granularity. If you are in a low-risk environment, you may be fine with a coarser system.

You may have noticed that one potential problem with using the two-dimensional classification system based on the military scheme is that you cannot neatly take into account the data that a particular computer of a given type is processing as well as the server type. For instance, not all database servers are alike. Some process highly sensitive personal information such as national ID numbers. Others hold public information, such as Web pages, that can be read by all users but written only by a few. Yet others servers may be entirely public and used simply as centralized temp folders. You can add rows to the classification for each computer type, but because many of the parameters you need to apply to computers are similar within a major type, this is not the cleanest method.

One way to accommodate sub-typing of computers a bit more neatly is to use a different modeling method. I like the organizational chart metaphor. It is infinitely extensible and permits easy sub-typing. You can, of course, use a more complicated modeling scheme, but because I find parsimony in your metaphor to be far more valuable than having hundreds of modeling constructs available, I tend to use simple modeling schemes. Using an org chart metaphor, we might come up with a picture such as the one in Figure 14-5.

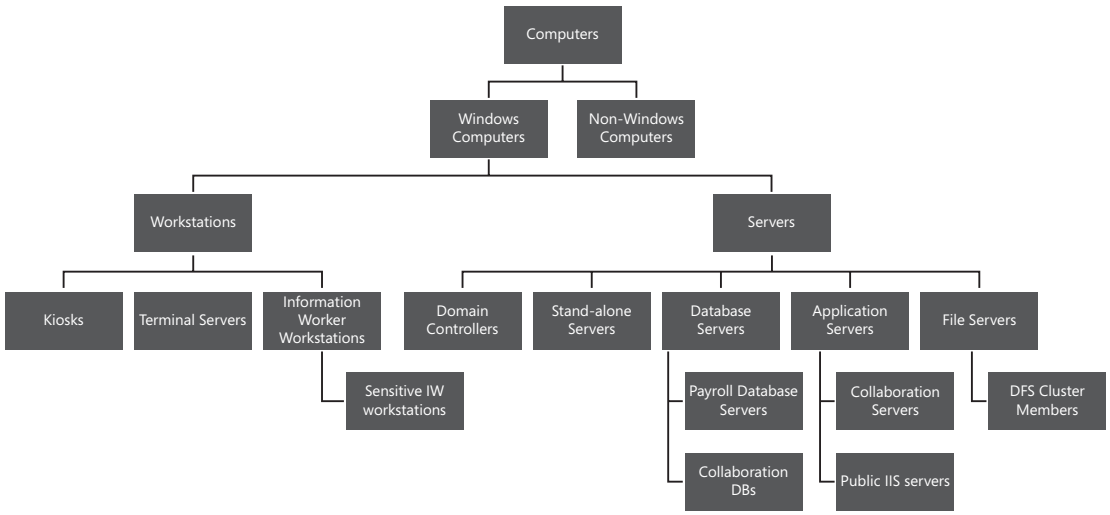

Figure 14-5 An org chart–style classification system is useful for complex environments where a lot of sub-types are needed to adequately express the security needs of the computers.

As Figure 14-5 shows, an org chart–style classification model can get rather large very quickly. Therefore, it might not be right for all environments. Note also that many of the categories are unlikely to have any computers in them. For instance, while Servers is a useful abstract super class, no computers should be assigned to it. All of them should be part of some specialization. However, when discussing server roles, as we did in Chapter 12, "Securing Server Roles," this type of hierarchical designation can be extremely valuable.

Once you have a preliminary classification model to start evaluating for fitness, you can begin your analysis. A useful technique for the analysis portion of the task is Network Threat Modeling, first described in (Johansson, 2005).

Steps 2 and 3: Network Threat Modeling

The next step is to see how well your classification model maps to the actual computers in your network. If you do not already have a map of your network, build one. It should detail everything important on your network, although you may group identical things together. The objective is to have something that lets you understand what your network looks like. Figure 14-6 provides an example.

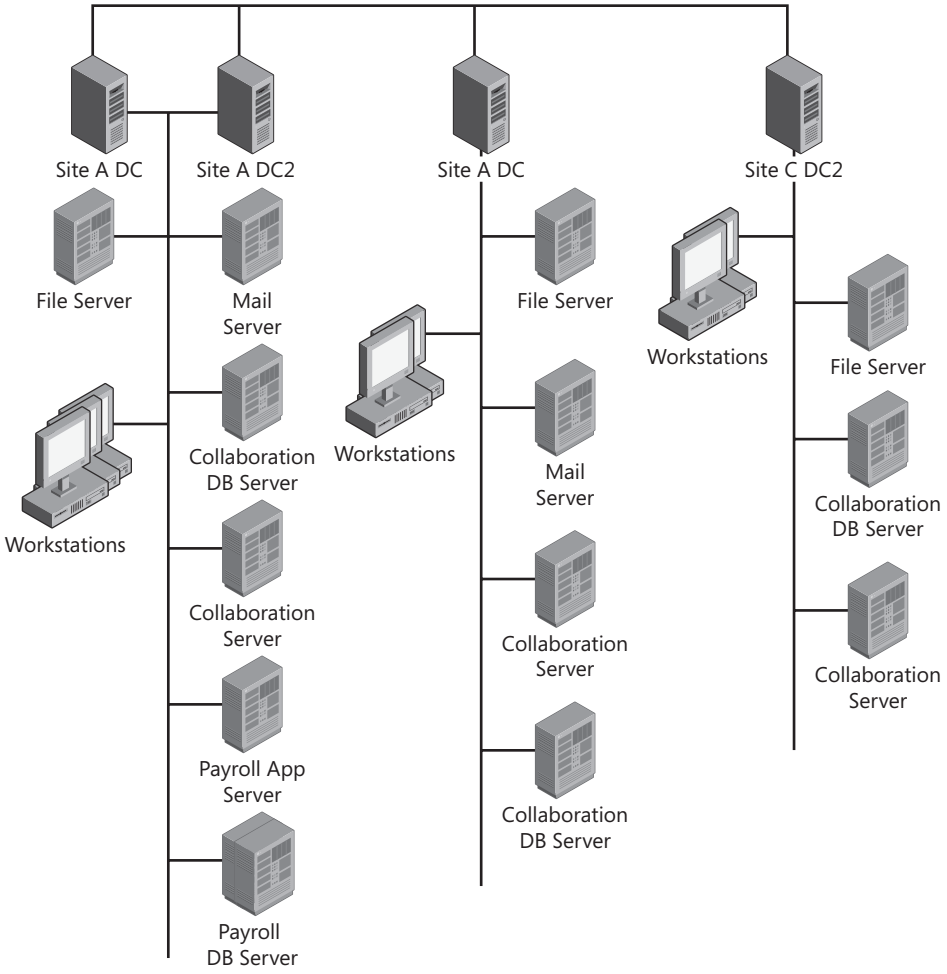

Figure 14-6 Locate or build a map of your network.

The next step is to start applying the classification scheme to the network map. As you have already noticed, Figure 14-6 is based on the physical design of the network, with each site shown separately, and with the same type of server in multiple sites. In Network Threat Modeling we are really not interested in the individual servers. Our objective is to understand the *types* of computers, not the individual computers. To that end, we take our classification scheme and overlay it on our network map. This will probably cause us to lose the distinction between sites. However, if the security needs of similar computer types are the same across sites, we have achieved exactly what we want to achieve. At this stage in the process we are trying to create a higher level of abstraction in our understanding of the network. This should result in a picture similar to Figure 14-7.

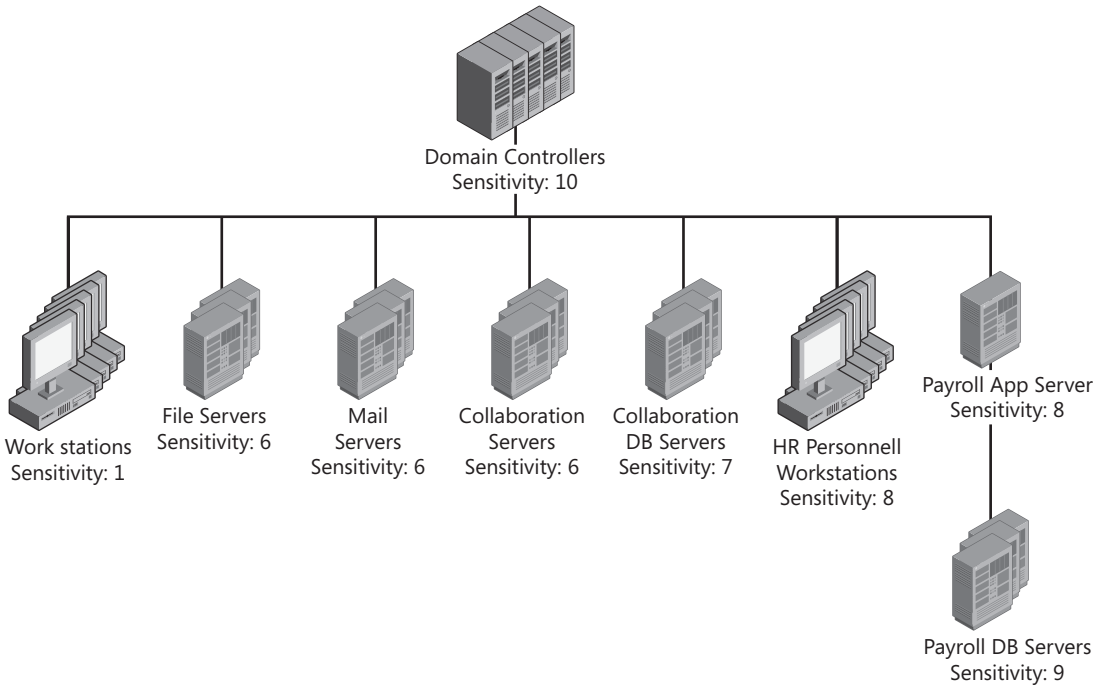

Figure 14-7 Start threat modeling by flattening the network and grouping computers into the classification scheme.

Figure 14-7 classifies computers into types based on our classification. Note that we have a new type of computer that did not appear before: the Human Resources (HR) Personnel Workstation. In this enterprise, we decided that because HR personnel have access to sensitive data on every employee, we needed to apply special security to their computers. Only some members of the client operations team that administers clients will have access to these computers. This prevents all client operations employees from having indirect access to personnel Personally Identifiable Information (PII).

When you have a classification scheme you have achieved a large portion of the objective of Network Threat Modeling. You should now be able to assign sensitivity labels to the various computer types. These labels are based on the types of data stored on that computer and the type of access to other computers you have if you successfully attack that computer. I have used numeric labels here, although you can use whatever makes sense.

DCs, obviously, are the most sensitive computers of all. Therefore, they have a sensitivity label of 10. By itself the number means nothing. It is just a way to relate one computer type to another. Workstations, because they are used by the largest proportion of users and at the highest risk, *should* be the least sensitive computers in the network. That does not mean that they are the least likely to be attacked. On the contrary, they are probably the *most* likely to be attacked. Therefore, they should be the least sensitive—in other words, the ones that give you access to the least amount of information in the network.

After you assign labels to all the computers, you should have a good idea of the patterns of operation in the network. This will drive your isolation strategy later. For now, we need to proceed to analyzing the communication patterns in the environment. To do that, we construct a picture similar to Figure 14-8.

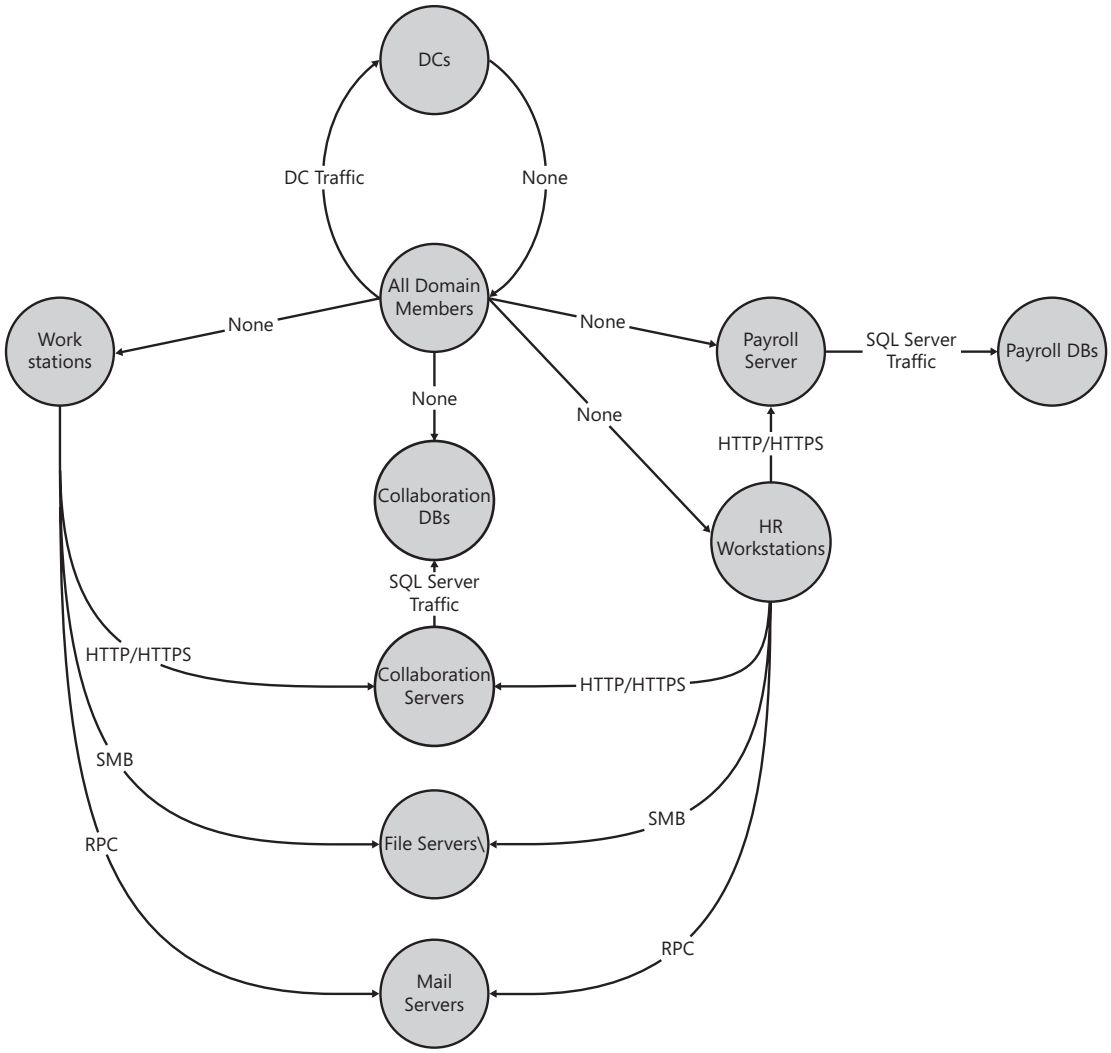

Figure 14-8 After you have grouped the systems, analyze their communication patterns.

Figure 14-8 is a basic Data Flow Diagram (DFD) of the network. The graph shown in Figure 14-7 does not easily lend itself to documenting communication patterns. However, a DFD is tailor-made for that purpose.

We start converting a network diagram to a DFD by simply turning the computer types into processes (the circles you see in Figure 14-8). Even databases are processes because the database server is actually what performs the processing on all database requests. Figure 14-8 also shows a little trick to make the picture far easier to read. Note the process named All Domain Members. It is marked as a duplicate entity with a slash through the corner. It represents all the non-DC computers in the domain. It serves as a very simple placeholder to clean up the diagram, letting us capture any communication pattern that is common to every computer in the entire domain. For example, all computers in the domain need to access the DCs. Instead of drawing separate lines from each computer type to the DCs, we draw just one from All Domain Members. In addition, rather than enumerating all the different types of traffic that domain members need to send to DCs, we use just one vector labeled DC Traffic. With this shortcut technique, what could easily have been 30 separate lines becomes just one.

> To learn more about the types of traffic used to access each type of server, see Knowledge Base article KB 832017. "Service overview and network port requirements for the Windows Server system" found *at http://support.microsoft.com/kb/832017*.

Note also that all the communication vectors are directed. The fact that domain members need to access DCs does not mean that DCs need to access domain members. In fact, they rarely do. If you are diligent about not using your DCs as workstations or management stations, you might not have to access any other computer from them.

Step 4: Analyze, Rinse, Repeat as Needed

While going through the Network Threat Modeling exercise, you might realize that your classification scheme is deficient. Youwill probably have computer types that are not used, and you will almost certainly realize as you go through the exercise that you are missing some types. If you are not, you probably have not adequately considered the security needs of your systems. Keep in mind that two things drive the classification scheme: First, you need to consider communication patterns. A computer that does not need to communicate in a particular way with another computer should not be permitted to do so. Second, computers that have different sensitivities should be managed differently to ensure that if one is compromised, the others do not fall.

One common mistake is failing to consider database servers separately from application servers. With properly written database middleware, which only calls exposed store procedures on the databases—and uses least privilege to do so—application servers are typically less sensitive than database servers. Unrestricted access to a database server means that you have complete access to all the data on it. Unrestricted access to an application server means that you should have access only to what the database will give you.

> You might have noticed that we have made an unstated assumption that there is no difference in the level of access to a particular computer, or rather, that this difference is not relevant to the Network Threat Modeling process. Windows does, in fact, have a reasonable level of isolation mechanisms to prevent someone with mere user access to a computer from taking complete control of that computer. However, Network Threat Modeling is complicated enough as it is. Mixing that in makes the model that much more complex. Instead, we are taking a worst-case scenario approach; we are basing our isolation techniques on what an attacker with complete control over a computer can do with that computer. For example, if an attacker has complete control over a SharePoint server, what access would that give her on the Office SharePoint Server 2007 database servers? The answer depends on how we manage the network (which users log on to the SharePoint servers) and on what traffic is allowed between the two.

If the classification scheme seems inadequate to the task of adequately capturing what your network looks like—or ought to look like—the solution is simple: either modify the classification scheme or change your assumptions. The classification scheme may just not be correct for your risk management strategy. In this case, you may change the classification scheme to better match the risk management strategy. Or, you might decide that although your risk management strategy is sound on paper, it is impossible or undesirable in practice. Many organizations have developed a risk management strategy

that looks great on a Microsoft Office PowerPoint slide in the boardroom, but is impossible to implement in the real world. This is your opportunity to verify how well your strategy really can be implemented. If you do not have a risk management strategy, you probably ought to take this opportunity to think up one.

Step 5: Design the Isolation Strategy

Once you have a network threat model that makes sense, you can start deriving the isolation strategy. The isolation strategy is largely based on the communication patterns identified in Figure 14-8. It should be as restrictive as possible, within reason. You can document the outcome in a table that outlines the server types and the communications patterns. The table includes the source and destination hosts and ports, the protocols, whether the traffic must be authenticated and/or encrypted, and whether the connection can also happen in reverse (mirrored). This is where you really need to get specific. Rather than simply saying "DC traffic," you need to enumerate the ports and protocols. An extraordinarily useful reference at this stage is Microsoft Knowledge Base Article 832017, "Service Overview and Network Port Requirements for the Windows Server System."

The end result of this step in the process should be a table that lists all necessary communications patterns in your network. Your table might look similar to Table 14-3, but will likely be far longer.

Table 14-3 Network Communications Patterns

Description	Source	Destination	Source Port	Destination Port	Protocol	Require Authentication	Require Encryption	Mirrored
DC SMB traffic	All Domain Members	All DCs	Any	445	TCP	No	No	No
DC RPC EP Mapper	All Domain Members	All DCs	Any	135	TCP	No	No	No
…								
Appserver DB Access	SharePoint Servers	SharePoint DB Servers	Any	1433	TCP	Yes	No	No
File Server Access	Workstations	File Servers	Any	(139), 445	TCP	Yes	No	No
HR Payroll Access	HR Workstations	Payroll Servers	Any	80	TCP	Yes	Yes	No
…								

As you can tell, the data captured in Table 14-3 can get quite extensive. However, if you have done the job of segmenting the network appropriately, the data should be mostly just tedious to gather. Once you have done so, you have almost completed the IPsec implementation of Server Isolation in your network. Notice that the headings in Table 14-3 actually capture the exact information you need for your IPsec rules. If you want to be really enterprising, you can enter Table 14-3 in a spreadsheet and then use a

macro to convert it into a series of IPsec commands to generate the required IPsec policy. You can configure IPsec on the command line using the **netsh advfirewall consec add rule** command. For more information on IPsec see the Microsoft IPsec site at *http://technet.microsoft.com/en-us/network/bb531150.aspx*.

Note that this kind of analysis will take some time. It is not unusual for a computer to have 50 or so ports open. The more standardized your OS images are, the easier it will be to track down the information you need. Furthermore, once you start doing this kind of analysis you will most definitely realize how valuable it is when the software vendor documents in a conspicuous manner what ports are used for what features.

Step 6: Derive Operational Strategy

The operational strategy is designed around how you are going to manage the various computers in your network. The strategy needs to capture the administrative needs of the computers as well as any services and other steps you take. For example, you probably want some backup strategy for your network. However, if you use a single, centralized backup system for all computers, you have probably defeated a large part of the isolation because you now have a backup server that has access to everything in the network, and it is potentially subject to attacks by every computer in the network. Therefore, you may want to analyze the risk involved in doing so. That analysis might lead you to conclude that the correct way to perform backups is to group computers by sensitivity and then handle backups uniformly within each sensitivity level. You might, for example, decide to use a single backup solution for all computers of sensitivity level 6.

You need to do the same analysis for administration. It would defeat the entire purpose of the exercise if you were to use a single, domain administrative account to access every computer in the domain. By doing that you expose the single administrative account to attacks on every computer. The appropriate decision is to use different administrator accounts for different purposes. You might decide that you have one account per sensitivity level. Or you might assign your sysadmins to different sensitivity levels. Doing so permits you to assign different sysadmins to different computers, rather than allowing all of them to administer every computer. You might even have separate administration stations for each sensitivity level. You have to decide how much pain you are willing to go through to manage your network. That decision will guide the rest of your decision making. With regard to security there is, as always, a tradeoff between how much security you wish to have, and how much inconvenience and work you are willing to put iup with to get it, all while taking into account the resultant functionality you want. The key point of this part of the process is to ensure that you implement the isolation in such a way that you do not expose computers of one sensitivity level to unnecessary attacks by computers at a different sensitivity level.

Step 7: Implement Restrictions

Finally, it is time to implement your strategy. By this stage in the process you should have a complete design for how you want to manage your network as well as for what communication patterns you want to permit within it. The implementation should be relatively straightforward at this point. However, you do want to ensure that several things get done.

Before we go further, if you are like most network managers, you get cold chills thinking about rolling out changes that could disrupt communication. After all, as the old saying goes, nobody ever calls the helpdesk to inform them that everything is working today (and if they do, you have a whole different set of problems to address).

Fortunately, there is a trick. Obviously, you want to eventually require authentication of all or most network connections. However, you may want to start out by *requesting* authentication instead. That way you can test the policies, monitor where IPsec negotiation fails, and adjust as necessary, all while maintaining full connectivity. This permits you to do a safer roll-out that is far less likely to result in events that have an adverse impact on your opportunities from promotion out of network management.

That being said, there are a number of other restrictions that you need to include in your plan.

Minimize Account Scope

First, reduce the scope of your accounts, particularly the highly privileged accounts. Everyone that accesses computers at different sensitivities should, at least if they have high-level permissions on those computers, have different accounts. For example, a highly trusted server administrator might need a domain administrative account for managing the DCs, a level 7 administrative account for managing servers at sensitivity level 7, and an information worker account for e-mail and surfing the Web. An HR employee might need one account for performing HR-related tasks and a different account for reading e-mail and working on presentations. Alternatively, you might decide that based on your risk management philosophy and the fact that both uses are at very low privilege levels, the same account might suffice. However, you should never permit an account that has administrative privileges at one level to access resources at a different level. Administrative accounts at any level must only be used to administer computers at that level.

Organizational Security Policy Changes

Much of the isolation must be done by organizational security policies, not necessarily technical policies. You simply cannot technically enforce many of the isolation decisions. For instance, your domain administrators are omnipotent within the scope of your network. You cannot restrict them from seeing or doing anything within the network. However, you can set rules and guidelines for them to follow, and track those guidelines. You must also have penalties for violating those guidelines. An administrator who refuses to take necessary steps to keep your network protected should be turned into an ex-employee.

Separate Service Accounts

Service accounts are a common problem in Windows. It is a well-known fact that any administrator on any computer has access to the clear-text password of all services—and of all interactive users—on that computer. There is no standard log file where these nuggets are stored, but with commonly available hacking tools it is a simple matter to get them.

For that reason, managing service accounts is crucial. It is still quite common to see services running on many computers in a network under a domain admin account. This

exposes a domain admin account on every computer in the network. For this reason, the scope of service accounts must be limited. A logical way to do this is to only use service accounts within a sensitivity level. For example, as mentioned earlier, the backup service might run in one service account on computers at level 7 and a different account on computers at level 9.

Do You Want To Back Up Workstations?

Do you really want to back up workstations? Many organizations are struggling with that question these days. Users, obviously, are storing data on their workstations, but is that what you really want? Ideally, very little data that does not exist elsewhere in the network should be on workstations. Using techniques such as roaming profiles and folder redirection, the default storage locations for users can be moved to the network. With the Offline Files feature, these files are automatically backed up to the network, and also available offline for roaming users. By using a combination of these strategies you can ensure that the only data available only on workstations is that data which users create while roaming, and that data which they choose to store locally—in possible violation of standard operating procedures. Combine that with a solid imaging strategy using (for example), the Windows Deployment Services, and you might achieve a state where you do not need to back up data on workstations. You might not even need to troubleshoot them. If anything ever goes wrong with a workstation, you could troubleshoot it by using same approach you use for servers, following this simple process:

1. Restart the service, if applicable.
2. Restart the computer.
3. Reimage the computer.
4. Send the computer back to the manufacturer and deploy new hardware.

If you do not need to worry about data being stored on workstations, you can make them disposable.

Note that getting to this stage will take discipline, along with hardware that permits you to implement a strategy like this. However, this is a business decision you need to make. Do you want to greatly simplify your desktop operating procedures, buy hardware and software that lends itself to that, and buy some really big storage servers, or do you want to have complicated and costly desktop operating procedures and spend less up front?

Manage Privileges

You must not forget to manage your privileges properly when you are implementing your isolation strategy. Users with certain privileges can be just as powerful as administrators. For example, a user that has the privilege Impersonate A Client After Authentication is as powerful as any user that connects remotely to the computer. A user that has the Restore Files And Directories privilege can replace any file on the computer. Because this permits that user to control code that is executed by administrators, such a user implicitly has all the rights that administrators do. This is why it makes great sense to separate backup

operators from restore operators. They are two different tasks, at very different sensitivity levels. Privileges are discussed in Chapter 2, "Objects: The Stuff You Want."

Restrict Communications

It should be obvious after our discussion on Network Threat Modeling that we want to restrict communications. In this step we use IPsec and Windows Firewall to restrict inbound traffic to a computer. This will greatly reduce the risk to systems from other systems. Take, for example, a database server accessed by a middleware server. Say there is a SQL Injection flaw in the middleware that permits an attacker to run arbitrary code on the database server. Once the database server has been compromised, what access does the attacker have to the middleware server? If you have set up Windows Firewall on the middleware server to reject all unsolicited inbound traffic (or, rather, to only accept exactly what it must accept) the answer is "none." You can contain the attack right there. Use the table you created listing your communication patterns and design a set of IPsec policies based on it. Deploy these policies using Group Policy or any other means that makes sense in your environment. To learn more about Windows Firewall and IPsec and how to deploy the policies, see Chapter 5, "Windows Firewall(s)."

Restrict Access to Resources

Finally, use the detailed knowledge you have gained, and the isolation strategy you have designed up to this point, to build a data and resource access strategy that enforces the principle of least privilege. You should, at this point, have a fairly detailed user account strategy. You can take advantage of that to prevent access, as well as to enforce the isolation strategy. For example, before you embarked on this project, your HR personnel might have had access to the Exchange servers, all the file servers, the internal SharePoint servers, and the payroll applications—all using the one account you gave them. After you define the isolation strategy, you have the ability to restrict their access to payroll applications to when they are using their HR_Personnel accounts, and possibly even when they are working on a specific HR workstation.

Summary

Few steps that you can take today will have as great an impact on the security of your network and its data as a proper network segmentation and Server Isolation strategy. By going through the process defined in this chapter, you can create a network that does exactly what you want it to do, and nothing else. If you do it right, the network will still be flexible enough to support new applications, with a minimal amount of modifications.

Will this strategy result in additional considerations for your end users and administrators alike? Certainly it will. However, in the world we operate in today, that might be the only way to secure your network. The conventional approach of a completely flat network, where everyone has one account that has access to everything they need, and much more, is simply unsafe in virtually all environments today. How far away from that model you are able to move depends on your risk tolerance, and the security needs you have in your environment.

Additional Resources

- Johansson, J. M. *Protect Your Windows Network*. (Addison-Wesley, 2005).
- Knowledge Base article 832017, "Service overview and network port requirements for the Windows Server system," at *http://support.microsoft.com/kb/832017*.
- "The Immutable Laws of Security," at http://www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.mspx?mfr=true.