

## Chapter 3: Authentication and Resource Protection in Windows 2000

Because this is a user administration book, not a security book, I'm not going to go into a huge amount of detail about all the new security features in Windows 2000. There are security improvements in many parts of the product: Kerberos for authentication, an encrypting file system, support for certificates, and in networking, IPSEC. Although this is a chapter on security, we aren't going to discuss all of Windows 2000's new security features. I am going to try and keep the discussion to the security features that impact your user or desktop administration duties. In this chapter, we'll discuss how Windows 2000 handles authentication, authorization, access control and auditing. There are other security functions we will discuss in later chapters. For instance, we won't discuss how to control visibility and permissions on directory objects until Chapter 4, "Understanding Organizational Units: The Building Blocks of the Active Directory." Some of the security features in Windows 2000 are implemented as policies—IPSEC is an example. We will discuss those security functions in Chapter 9, "IntelliMirror Features for Client Management," when we are talking about creating and deploying policies.

In the real world, you really should at least have a passing familiarity with what Windows 2000 security can do for you as an administrator, so in this chapter, we will have a high-level discussion of some of the new security features, talk about some basic security concepts, and finally discuss how to secure your file system resources and implement auditing on those resources. As I mentioned in Chapter 1, "Inside Windows 2000 Overview," the permission and inheritance model has changed quite a bit in Windows 2000, so we will spend a considerable amount of time on that at the end of the chapter to make sure you fully understand the security implications in administering the system. Let's start with some general security terms and concepts, and see how Windows 2000 implements these functions.

### Security Perimeters

When I was first learning about computer security, I was always told that there are four major security perimeters in any computer system:

- Physical security
- System security
- Application security
- Data Security

I was also told that these four layers could be thought of as being arranged in a bull's-eye pattern. As you move toward the center of the eye, each ring you pass through provides a higher level of security for the resources contained in that ring.

In [Figure 3.1](#), you can see that the outer ring is the physical security ring. The fence around the installation, locked doors, and bulletproof glass would be considered physical security measures. Moving toward the center of the bull's-eye, the next ring is the system security perimeter. The idea of

a network logon to an NT domain is an example of system security. After the system ring, there is the application ring and finally the data ring. Often we use the system logon to determine access to the innermost rings by using pass-through authentication to determine access levels for a particular user.

### **Figure 3.1**

Security bullseye.

## **Security Activities**

Any decent security system has to allow for the implementation of certain basic security functions. Naturally, every system implements these differently. Five of the basic security activities are identification, authentication, authorization, access control, and auditing. How strong a security system is depends on how strong and trustworthy each of these implementations is.

### **Identification**

Identification is the concept of identifying yourself to the system. Depending on the perimeter, the method of identification differs. In physical security systems, this may be a badge. In system, application, and data security, you are typically identified by your username: your logon ID.

Windows 2000 can use two methods of identification for local and network logons. One is the traditional username and another is the Smart Card.

### **Authentication**

When you authenticate to a system, you are proving that you are who you say you are. That picture on your badge is an example of authentication in the physical layer. In the system security ring, your password is a typical authentication mechanism. The certificates on a Smart Card logon are also examples of authentication.

In order to achieve authentication, a network must know how to authenticate. What methods are followed? What input is acceptable as an authenticator? How do we securely pass those authenticators to a machine that can validate them? How do we let a client know they have been authenticated? All these questions are answered by the choice of an authentication protocol. Windows 2000 supports three authentication protocols: NTLM, Kerberos, and SSL/TLS. We will discuss these protocols in detail a little later in the chapter.

---

### **Smart Cards**

There is a lot of talk in the Windows 2000 documentation about Smart Cards as an identification and authentication mechanism. Exactly what is a Smart Card? A smart card is a credit-card-sized device that is inserted into a smart card reader, which is either installed internally in your computer or connected externally to your computer. The combination of the card (something you have) and a PIN that you enter (something you know) becomes the identification and authentication mechanism. To provide further security in a public key infrastructure environment, certificate information can also be placed on the card. As you probably already know, a certificate is an encrypted set of

authentication credentials. These credentials are then passed to the system to be used as you access different servers. In order to use certificates in Windows 2000, you must use activate Certificate Services.

---

## Authorization

When the system allows you to cross any given security perimeter, we refer to that as the process of authorization. You are being given the "authority" to cross the perimeter. When the door opens after you swipe that badge, the system has authorized you to enter a secure perimeter. When NT logs you on after you enter your username and password correctly, you have been authenticated at the system security perimeter.

## Access Control

After you have been allowed to cross a given perimeter, there are usually further controls on your activities within that perimeter. After all, just because you have gotten in the gate doesn't mean the house is unlocked too. From a physical security standpoint, that is why you see multiple security levels within a facility, with more locked doors protecting each area. In general, how access control is handled on a system level depends on whether the particular system invokes share-level or user-level security (see sidebar).

---

### Share-Level or User-Level Security?

Share-level security is the type usually offered by less secure peer-to-peer networking systems, such as Windows for Workgroups. Windows 9x can implement either share- or user-level security. With a share-level system, access restriction options are rather limited. Typically, they are Read Only, Full Control, or, at most, the access level can be set depending on the password supplied. There would be one password for Read Only and another password for Full Control. All the Read Only users share the same password to access the share. I call this "secret knock" security. If you know the secret knock, you get in. People being what they are, when a password is out for a share, it typically ends up being given to unauthorized users for one reason or another. Another aspect of this type of security is that there is only one checkpoint—the share. After you get into the sharepoint, all files and folders that exist in the share are visible to you and, if you have full control, alterable by you.

User-level security, on the other hand, requires that each user of a system be uniquely identified. This concept of a unique identifier for users and groups is then used to implement discretionary access control on resources in the system—discretionary because your access depends on your identity and the permissions set on the resource using that identifier. Typically, discretionary access controls are applied at each level of a system's resources. For example, there would be a list governing access to the sharepoint, then to the folder, then to subfolders, and finally to individual files. This allows an administrator to be very specific with permissions on resource levels. If a user wants to use a resource, he must have correct permission entries on all the lists leading to the resource he wants to access, based on either his personal identity or a group he is a

member of. In Windows 2000, these permissions lists are called Discretionary Access Control Lists (DACLS).

---

Like its NT predecessors, Windows 2000 implements access controls through the use of identifiers that uniquely identify users and groups. It then sets permissions using those identifiers to restrict or allow access using access control lists attached to the resources. As always, the system operates on the model that if there is not an explicit entry on the list for a user, that user gets no access. There is no such thing as an implied permission in the NT world. In addition, permissions setting is still the responsibility of a resource's owner or an administrator who has taken ownership of a resource on a user's behalf. There are a few differences in both the permissions available and the inheritance model that Microsoft has implemented in Windows 2000. We will discuss those differences in detail a little later in the chapter in the section on "Setting Object Permissions and the New Inheritance Model."

## **Auditing**

One requirement of a good security system is the capability to track what has happened on your system. Auditing is the security function that watches and then records security events of interest to the system administrator. Like all the other security activities, this may happen at any and all perimeters as needed. I remember one of our door entry systems was protected by a keypad system where we had to swipe our badges and then enter a PIN for entry. Each and every entry was then logged to a system in the security manager's office for auditing purposes.

Windows 2000 implements auditing of many system and access-related events through the use of an overall security policy that defines what type of security events to audit. In addition, each Active Directory and file system object can be protected by a Security Access Control List (SACL).

## **Secure Authentication Services in Windows 2000**

An authentication protocol determines the method a system uses to verify a user's identity and access levels. In Windows 2000, more authentication methods are available for use than in previous NT systems. Microsoft has accomplished this by including additional authentication protocol support to the security architecture.

For backward compatibility with previous Microsoft systems and standalone logons, Windows 2000 still supports the traditional NTLM authentication protocol. As you have no doubt already heard, Microsoft has also opted to use the Kerberos security system as the default protocol for network authentication. The final authentication protocol that is supported is Secure Sockets Layer/Transport Layer Security (SSL/TLS). Using this protocol and X.509 certificates, Windows 2000 can authenticate Smart Card users and provide protection for unsecured networks, such as the Internet.

### **NTLM Authentication**

NTLM is Microsoft's legacy authentication protocol from NT 4. Prior to Windows NT 4.0 Service Pack 4 (SP4), Windows NT supported two kinds of challenge/response authentication: LAN Manager (LM) challenge/response and Windows NT challenge/response (also known as NTLM challenge/response). In addition, Microsoft has developed a new version of NTLM known as NTLMv2.

A brief explanation of all three is in order so you can make informed decisions about allowing support for these protocols in your Windows 2000 environment.

LM authentication is the weakest of the three protocols because, although the passwords in LM can be longer than 7 characters, the algorithm allows longer passwords to be attacked in 7-character chunks. Password characters can be drawn from the set of uppercase alphabetic, numeric, and punctuation characters, plus 32 special ALT characters.

In contrast, NTLM uses all 14 characters in the password as a single contiguous unit and allows lowercase letters. Basically, increasing the length and character complexity in this manner means that although an eavesdropping hacker can attack in the same way as with the LM authentication protocol, it will take far longer for him to be successful. Naturally, if your users are choosing shorter passwords, whole words, or blank passwords, NTLM won't help!

NTLM on steroids is called NTLMv2 by Microsoft. NTLMv2 improves both the authentication and session security mechanisms of NTLM. In addition, the NTLM Security Service Provider (SSP) now allows clients to control which version of NTLM to use and allows servers to decide which alternatives to accept. Finally, NTLMv2 allows clients and servers to require the negotiation of message confidentiality (encryption), message integrity, 128-bit encryption, and NTLMv2 session security. NTLMv2 is available on NT 4 systems with Service Pack 4 or higher.

In Windows 2000, you have the capability to turn off NTLM support and go with a purely Kerberos model that provides a higher level of security for your network. However, you may find that you need to continue NTLM support on your Windows 2000 system for several reasons.

The first reason to continue NTML support is if you have downlevel clients and servers that use one of the LM/NTLM authentication protocols listed above. Computers with Windows 3.11, Windows 95, Windows 98, or Windows NT 4.0 use the NTLM protocol for network authentication in Windows 2000 domains. Second, your Windows 2000 machines use NTLM when authenticating to servers with Windows NT 4.0 and when accessing resources in Windows NT 4.0 domains. These first two reasons apply if you have a mixed Microsoft network environment.

Other reasons to keep NTLM support have to do with, believe it or not, UNIX compatibility. Even though we think of Kerberos as a UNIX standard, you may also have to consider keeping NTLM support if you have UNIX clients connecting to your Windows 2000 domains. If they are using an SMB client to connect, whether the UNIX clients are configured for Kerberos or not, they need to use NTLM to connect because of the requirements set forth by the SMB client. If the UNIX clients are using standard TCP/IP application protocols, such as Telnet and FTP, exclusively, you can then eliminate NTLM support. Finally, if your Windows 2000 clients are connecting to a UNIX server using an SMB daemon and you want to keep that model, you will still need NTLM. Another option in this situation would be to turn off the SMB daemon and use an NFS client for the Windows 2000 boxes. Microsoft provides that client as part of their UNIX interoperability package, Services for UNIX.

## **Kerberos Authentication**

In Windows 2000, Microsoft is implementing Kerberos version 5 with extensions to support public key authentication as the preferred network authentication protocol between Windows 2000

machines. In this section, we will discuss the advantages of this decision as well as how Microsoft's implementation of Kerberos works in the simplest possible terms. As I mentioned before, an in-depth discussion of Kerberos v5 is way beyond the scope of this book. But if you are a nuts and bolts kind of person and want all the gory details of Kerberos, there is good news. Because Kerberos has been around for more than a decade and is an established industry standard security protocol, there is lots of information out there on the system. To look at Kerberos strictly from a Windows 2000 viewpoint, Microsoft has a white paper titled "Windows 2000 Kerberos Authentication." *Windows NT* magazine has also published a couple of good, understandable articles on Kerberos. You can get those archived articles from their Web site at <http://www.winntmag.com/> if you are a subscriber.

### **Advantages of Using Kerberos**

Kerberos has a few advantages over NTLM as an authentication protocol. The number one advantage may very well be that it is an industry standard protocol. Kerberos v5 is outlined in RFC 1510 from the IETF and is on standards track. In their whitepaper on Kerberos Authentication in Windows 2000, Microsoft says that their implementation of Kerberos "closely follows the specification defined in Internet RFC 1510." That's Microsoftese for "we did it our way." To be fair to Microsoft, though, that's true of all standards implementations. From what we have seen of the product so far, it is generally agreed that cross-system authentication between Windows 2000 and existing Kerberos-based authentication systems should be possible. Let's look at some of the other advantages of Kerberos in the following section.

#### **Quicker Server Connection Time**

As we discussed in Chapter 2, with NTLM, an application server connects to a domain controller to authenticate clients. This happens each time a client establishes a session with the application server. If a client connects to OFFAPPS1, disconnects and reconnects a few minutes later, the machines go through the whole authentication process again. With Kerberos, after the client and the applications server are both authenticated, and the client is approved for access, a set of credentials is issued for a session between the two machines. The client can then reuse those credentials for subsequent conversations with that server. Because no further authentication is necessary, the use of Kerberos speeds up server connection time by eliminating the need to authenticate each time a connection to the same server is made. This can provide a noticeable improvement for users who are coming in over slower WAN connections.

For improved security, these credentials have an expiration time associated with them. You can set that expiration time to match the security levels required in your organization. By default, the time to live on a session ticket in Windows 2000 is eight hours.

#### **Mutual Authentication**

NTLM is a one-way authentication protocol that allows clients to authenticate to a server. How does the client know it is really connecting to the server that it is supposed to be connected to and not some hacker's proxy? With NTLM, you don't know and can't be sure. NTLM was designed in an era where servers were assumed to be genuine. This is an invalid assumption in today's open Internet environments.

Kerberos, in contrast, is based on mutual distrust. The Kerberos server makes all machines in the domain prove their identity before they are allowed to participate in the domain, even the servers. Through the use of shared private keys, Kerberos can guarantee that all the parties involved in the

conversation are who they say they are.

### **Delegated Authentication**

Windows services often have a need to impersonate clients when accessing resources. For example, this must be done when accessing a remote server. NTLM had no capacity to impersonate a client and access a remote service. Kerberos has a proxy function that allows a service to impersonate its client when connecting to other services on remote servers. For example, if a user is accessing an application on the Personnel server and that application requires a resource from the Payroll server, the Personnel server pretends to be the user and passes the user's credentials when requesting authentication on the Payroll server.

### **Simplified Trust Management**

Because Kerberos employs mutual authentication, it can implement a trust model that allows for trust relationships that are two-way and transitive. Mutual authentication guarantees that any machine in a domain truly is a member. Kerberos forces servers to authenticate as well, so that all the domain controllers are also legitimate. In view of that assumption, the domain controllers become trusted security principals in a multidomain network and can be organized in a tree of transitive, mutual trust. Kerberos credentials issued in one domain can then be used and accepted anywhere in the tree or forest.

### **Interoperability**

As mentioned earlier, Microsoft's implementation of the Kerberos protocol is based on IETF standards-track specifications (RFCs 1510 and 1964) from the Internet Engineering Task Force (IETF). Therefore, the implementation of Kerberos in Windows 2000 gives you a basis for interoperability with other networks where Kerberos version 5 is used for authentication.

In Windows 2000, Microsoft has implemented extensions to the Kerberos protocol that permit initial authentication using public key certificates rather than the conventional Kerberos shared secret keys. This enhancement allows the protocol to support interactive logon with a Smart Card.

### **How Kerberos Works**

How does this paranoid security system work? Get yourself a cup of coffee and open your mind—this can be a little hard to follow if you've never seen it before.

Keep in mind two things as we go along. First, remember that Kerberos operates on the concept of symmetric (or shared private key) encryption, meaning you must use the same key to decrypt a message as you used to encrypt the message in the first place. I call this the secret decoder ring method of encryption. If we both have the same secret decoder rings, with identical keys, we can encrypt and decrypt messages we send to each other. However, if you have the Batman decoder ring, and I have the Superman ring, we can't send secret messages to each other. Kerberos works on this same concept by issuing symmetric keys for authentication and work sessions.

Another thing to remember is something I mentioned briefly earlier. Kerberos works on the assumption of mutual distrust. It doesn't believe that anyone is who he say he is until his identity is proven by his ability to decrypt messages encrypted in the private key he shares with the Kerberos server.

Finally, in this shared private key system, there must be a place that keeps track of and distributes these algorithms. In Kerberos, this function is called the key distribution center (KDC). In the Windows 2000 implementation of Kerberos, the KDC is hosted in the Active Directory and is, therefore, resident on each and every domain controller. This gives the KDC all the advantages of the multimaster domain controller replication and fault tolerance. So in this discussion, whenever KDC is mentioned, think "domain controller."

In theory, the functioning of Kerberos is rather simple. A centralized security server (or replicated servers) issues each security principal in their domain a secure key to talk to the Kerberos server. This is often referred to as the long-term key. Long-term keys are typically derived from the user's password and other information through a one-way hashing algorithm and stored in the local credentials cache. The credentials cache is an area of protected volatile memory that is destroyed when the user session is ended. Each security principal has a different key than its neighbor, and the security server is the only other party that knows what each key is. This key is used to get another set of credentials, called a ticket-granting ticket (TGT), that allows a user to request access to other servers.

If one security principal wants to talk to another security principal—Mary Jo wants to access that OFFAPPS1 server, for example—Kerberos has to issue a session key (sometimes called the short-term key) to Mary Jo and the applications server so that those two can send secure messages to each other. Each session key is unique and has an expiration time on it to further enhance its security. Session tickets can be renewed if the administrator chooses to set this up as part of the domain's Kerberos policy. See, simple...in theory! Let's look at the implementation of the details.

### **Logging On**

We'll start with Mary Jo's logon. When Mary Jo logs on to her local machine, the Kerberos client software running on her workstation converts her password to an encryption key and saves the resulting key in its credential cache. Then the client sends an authentication request to the Kerberos server. The first part of the authentication request identifies Mary Jo and the name of the service for which she is requesting credentials. Because this is her logon, she is requesting credentials for the ticket-granting service. As I mentioned, most people don't log on to the domain to talk to the security system, they want to access applications and other resources. When she has logged on successfully to the domain, the KDC will issue her a TGT that she will use when talking to the Kerberos server (during this session) to request access to other servers. The second part of this authentication request message contains pre-authentication data that proves Mary Jo knows the password. This is usually a timestamp encrypted with Mary Jo's long-term key. Other forms of pre-authentication data can be accepted by the protocol.

When the KDC receives this package, it checks its database for Mary Jo's long-term key, decrypts the pre-authentication info, makes sure the timestamp is within allowed parameters, and then issues Mary Jo a TGT that she will use to obtain future session tickets. To accomplish this transaction, the KDC invents a session key for itself and Mary Jo to use for their conversations. Then, it encrypts this logon session key with Mary Jo's long-term key. Next, it embeds another copy of the logon session key and other authorization info about Mary Jo. It encrypts these two things in its own long-term key. This portion that is encrypted in the KDC's long-term key is the actual TGT.

When Mary Jo's station receives this reply, it decrypts the session ticket using Mary Jo's long-term

key and stores the session ticket in its credentials cache. It also extracts the TGT, still encrypted in the KDC's long-term key, and stores it in the credentials cache.

### Accessing Other Servers

Now that Mary Jo has been authenticated onto the domain and has a TGT to request access to other resources, she can ask for permission to talk to the OFFAPPS1 server. From the user standpoint, all this is invisible. Mary Jo probably goes to My Network Places, searches the Active Directory, or clicks the Microsoft Word icon on her desktop. When she does, the Kerberos client on her machine sends a Ticket Granting Service Request message to the KDC that includes Mary Jo's name, an authenticator (again, usually a timestamp) encrypted in the logon session key, the TGT (still encrypted in the KDC's long-term key), and the name of the service for which Mary Jo is requesting a session key (OFFAPPS1, for example).

When the domain controller receives this message, it decrypts the TGT with its long-term key and extracts the logon session key it embedded there. It uses that key to decrypt the authenticator portion of the request. If the authenticator is within bounds, the KDC creates a session key for Mary Jo to use when talking to the applications server. Now, at this point, the KDC could send the session key to Mary Jo and a second message to the applications server with that same session key in it. This method of key distribution would be inefficient because it requires the KDC to send two messages. Problems would also be caused if there was a network disturbance and Mary Jo tried to use her session key before the applications server had received its copy of the key. In order to overcome these potential problems, the KDC's response to Mary Jo's request for access to the applications server looks like this: the response contains two parts. The first part is the session key for the Mary Jo-OFFAPPS1 session, encrypted in the Mary Jo-KDC logon session key. In the second part of the response, the KDC embeds a second copy of the Mary Jo-OFFAPPS1 session key and Mary Jo's authorization data and encrypts this second portion in the OFFAPPS1 server's long-term key.

When Mary Jo's machine receives this response, it decrypts the first part of the message and stores the OFFAPPS1 session key in its credentials cache. It then extracts the second part of the message, the part encrypted in the OFFAPPS1 long-term key, and stores that in its cache as well.

When Mary Jo goes to access the applications server, a message is sent from the Kerberos client on her machine to the Kerberos client on the applications server. This message is called a Kerberos Application Request and contains an authenticator encrypted in the Mary Jo-OFFAPPS1 session key and the ticket that was sent back to Mary Jo's machine encrypted in the applications server's long-term key.

Upon receipt of the Application Request, the applications server decrypts the session key and Mary Jo's authorization data using its own long-term key and then uses the session key to test the authenticator that is encrypted in the session key. If the authenticator checks out, it sends a response to Mary Jo's machine, and the conversation between Mary Jo and the applications server can proceed.

I warned you! Is that coffee cold yet? And that was the executive overview version of logging on within a single domain! When you are working within an Active Directory structure that has more than one domain, Kerberos accepts authentication credentials from other domains in the tree or forest. The way that those are forwarded from one domain to another is a little more complicated than what we have discussed here. Again, if you want the fine details, go to the Microsoft white paper on Kerberos authentication. I think this discussion was enough for you get the idea, though. I'm sure you

can see why Kerberos is considered a good authentication protocol.

Remember, Kerberos is just that, an *authentication* protocol. It does very little to help with implementing access controls after the initial logon to a machine or service. All right, in our example, Mary Jo has been allowed to log on to the applications server, but can she run MS Excel? Kerberos plays a small role in determining the answer, but those kinds of permissions questions come under the mantle of discretionary access control.

## Secure Sockets Layer/Transport Layer Security

The final authentication protocol supported in Windows 2000 is SSL/TLS. This protocol is already supported in other Microsoft products, such as Exchange and IIS. It is used primarily to authenticate Smart Card users using a combination of this protocol and X.509 certificates, and to protect connections on unsecured networks. It can be used to secure your email and I'net (Internet and intranet) applications.

## Security Access Tokens

As mentioned earlier, in order to implement discretionary access control, each security principal must be uniquely identified to the system and there must be a mechanism to compare that identifier to a list of approved users for each resource in the system.

In Windows 2000, as it was in NT 4, security principals are uniquely identified by a SID (Security Identifier), a unique alphanumeric value consisting of two parts. The first part identifies the domain in which the SID was issued, and the second part uniquely identifies the account within that domain. Keep in mind that in Microsoftese, a domain can be the individual machine if it is a standalone box or a member of a workgroup. Regardless of the scope of the "domain," Microsoft guarantees that a SID will always be unique and never reused. SIDs are used to identify individual users as well as groups within the domain. These SIDs are used by the Local Security Authority (LSA) on each local machine to build a Security Access Token (SAT) that follows a user's processes throughout the system. The SAT contains the user's individual account SID as well as all the SIDs for any groups he is a member of. Based on those SIDs, a list of system rights is appended to the access token as well—the right to log on interactively to a domain controller, for example, or permitted logon hours.

In standalone machines, workgroups, and domains not using Kerberos for whatever reason, NTLM is used to build SATs just as they were built in previous versions of NT. In a native mode domain where Kerberos authentication is enabled, the KDC prepares authorization data in two steps to assist in the access control process.

Step one takes place when the KDC prepares a TGT for a user. When the TGT is requested, the KDC queries the Active Directory for the user's account. In the attributes of the account there is a field for the user's SID and another attribute field for the SIDs of the security groups the user belongs to. Both of these attributes are returned and embedded into the authorization data field of the TGT. In a multidomain environment, the KDC also queries the Global Catalog for universal groups that include the user's SID or the SIDs of any of the domain security groups of which the user is a member. If those are found, the SIDs for the universal groups are also included in the authorization data field.

Step two happens when the KDC prepares a session ticket for conversations between two parties. When a user requests a session ticket for a particular server, the domain controller (KDC) in the

server's domain copies the authorization field data from the TGT and embeds it in the session ticket's authorization field. If the server is in a different domain than the one where the user account was created, the KDC in the server's domain will check its Active Directory for any domain local groups that contain either the user SID or group SIDs that are already in the authorization field. If it finds any, it adds these to the session ticket's authorization field before it is passed on to the target server. This authorization data is signed by the KDC.

Wait a minute, what happened to the LSA and the access tokens? They are still in existence. When this session ticket is received by the target server, the LSA process takes the information in the authorization field and creates an access token for that user on the local machine. At that point, the functioning of the system is essentially the same as an NT 4 box.

## Discretionary Access Control Lists (DACLS)

Now that the user is authenticated onto the server, the user's SAT can be checked against the permissions list for the objects she wants to use on that server. The header of every object in the system contains a security descriptor with a list of permissions embedded in it. This list is called the Discretionary Access Control List (DACL) and contains SIDs and permissions levels for each SID listed. If there isn't an explicit entry on the list for a user or one of her groups, it is assumed by Windows 2000 that no access should be allowed. The permissions list is maintained by the owner of the object. Only the owner is allowed to grant and revoke permissions to the resource.

---

### Administrative Ownership of Resources

In the event of an emergency, an administrator can take ownership of a resource and even reset permissions because he is the new owner. This ability is usually exercised when unexpected events happen (like poor old Bob gets hit by a bus at lunch) and it is necessary to transfer ownership to another user. To take ownership, the administrator would go to the Properties screen of the resource and select the Security tab. On the Security screen, select the Advanced button and, finally, the Ownership tab shown in [Figure 3.2](#), and select the new administrative owner. An administrator can elect to set himself as the owner, in which case the resource's size would be counted against his personal disk quota space if disk quotas are being used. He can also elect to set the ownership to the Administrators group. The Administrators group is exempt from disk quotas.

#### **Figure 3.2**

Changing ownership of a resource.

To pass ownership to another user, the administrator can go directly to the Access Control Settings screen by using the Advanced button on the Security tab of the Properties screen. From there, click on the Add button, and select the desired user. Then, the Permission Entry screen will pop up. Now, scroll down to the Take Ownership permission on the Permission Entry screen and put a check in the Allow box to enable the chosen user to take ownership of the resource.

Now, when that user logs on and accesses the Security Properties for that object, he will

have the ability to go to the ownership tab and select himself as the new owner of the object. One interesting thing about the ownership interface: In the list of users and groups that are displayed to select from, only users and groups that have the permission to take ownership will ever show up. As a further security precaution, only the currently logged-on user and his authorized groups will show up in the list. For example, if I were logged on and had been given the permission to take ownership of a resource, the list would show my "Lori" account as well as the "Administrators" group for my machine. Because ownership can only be taken and never given, this makes sense. Otherwise, an unscrupulous administrator would have the ability to take ownership and then give it back without the resource's true owner ever knowing about it. Not a good thing, especially if it's the payroll file or strategic, sensitive corporate information.

---

## Setting Object Permissions and the New Inheritance Model

One of the key things to remember about Windows 2000 and permissions is that there are different types of objects you can set permissions for. It can be a little confusing at first, but if you are used to Novell or Banyan, or are familiar with the idea of shares in the Microsoft world, I think you'll have no problems here.

When we discuss permissions in Windows 2000, we talk about setting permissions on file system objects or directory objects. In this section, we'll discuss setting permissions on file system objects. Those would be limited to files, folders, and shared folders. We'll talk about directory object permissions in the next chapter. The interface for setting permissions on both types of objects, file system and directory, is almost the same. Also, the inheritance model is very similar, so when you get it for file systems, the directory should be a breeze.

### Setting File System Permissions

When setting permissions for file system objects, we are actually going to limit access to the object. As with NT 4, the default permissions mask on the Windows 2000 file system is basically everyone has complete access. If you are coming from the Novell or UNIX world, you are probably breathless with shock right now. Instead of opening access up to an NT/Windows 2000 system like you would in other environments, you must do the opposite. You must shut it down. If you don't believe me, just go have a look at your system volume. By default, it will say Everyone, Full. There are exceptions to this mask in Windows 2000. The %systemroot% directory where Windows 2000 is installed, as well as the Program Files and Documents and Settings directories, has a different default mask that allows average users less access and administrators full access. But, trust me, almost everything else is wide open. And remember, because the guest account is part of the Everyone group, that means whatever Everyone has, Guest has if the account is enabled. If it is enabled, anyone *without* a valid account will be logged on as a guest. It's kind of like putting out a big welcome sign for Hackers, Inc. That's why we're going to spend a little time on setting permissions.

Because Windows 2000 uses discretionary access control, you can think of access to a resource as being similar to passing through several locked doors in a building to get to your office. In order to access a file, the user must have all the right "keys" to get through the file system path that leads to the file, as well as the correct key to open the file. Having the right keys means having the necessary permission entries on the DACLs of the volume, folders, subfolders, and finally the file. If the right permission level is missing at any of these preliminary DACLs, access to the resource may be denied.

If it is a shared resource, and users are coming in over the network to access it, they must also have correct permissions on the sharepoint to proceed with their entry into the file system.

### Standard Permissions Sets

As with NT 4, there is a set of standard permissions for files and folders that you can use to make your life as an administrator easier. You also have the ability to set very specific "special" permissions on a file system resource if necessary. We saw an example of this in the sidebar on taking ownership.

Setting standard permissions can be accomplished using the first screen of the Security Properties dialog box as shown in [Figure 3.3](#). To get to this dialog box, right-click on the desired object, select Properties from the menu, and then select the Security tab.

#### **Figure 3.3**

Security Properties dialog box used for setting standard sets of permissions.

This dialog box is just the first of three that you can use to set permissions on resources. The permissions set shown applies to the highlighted user or group. In this case, our screen shot shows the permissions for the Administrators group. To see the other permissions levels, you would need to move the highlight bar to the appropriate group or user. Adding and removing users from the list is very easy. Simply choose Add or Remove. This part really is intuitive!

Table 3.1 shows the standard permissions sets for files and folders and exactly what those sets mean in Windows 2000.

**Table 3.1 Standard Permissions**

<b>Standard Permission</b>	<b>Specific Permissions Granted as a Result</b>
Read	List Folder/Read Data  Read Attributes  Read Extended Attributes  Read Permissions  Synchronize
Read and Execute	List Folder/Read Data  Read Attributes  Read Extended Attributes  Read Permissions  Synchronize

	<ul style="list-style-type: none"> <li>Traverse Folder/Execute File</li> </ul>
Modify	<ul style="list-style-type: none"> <li>Create Files/Write Data</li> <li>Create Folders/Append Data</li> <li>Delete</li> <li>List Folder/Read Data</li> <li>Read Attributes</li> <li>Read Extended Attributes</li> <li>Read Permissions</li> <li>Synchronize</li> <li>Write Attributes</li> <li>Write Extended Attributes</li> </ul>
Write	<ul style="list-style-type: none"> <li>Create Files/Write Data</li> <li>Create Folders/Append Data</li> <li>Read Permissions</li> <li>Synchronize</li> <li>Write Attributes</li> <li>Write Extended Attributes</li> </ul>
List Folder Contents	<ul style="list-style-type: none"> <li>List Folder/Read Data</li> <li>Read Attributes</li> <li>Read Extended Attributes</li> <li>Read Permissions</li> <li>Synchronize</li> <li>Traverse Folder/Execute File</li> </ul>

Full Control	Change Permissions
	Create Files/Write Data
	Create Folders/Append Data
	Delete
	Delete Subfolders and Files
	List Folder/Read Data
	Read Attributes
	Read Extended Attributes
	Read Permissions
	Synchronize
	Take Ownership
	Write Attributes
	Write Extended Attributes

You might have noticed the new column for "Deny" in [Figure 3.3](#). In NT 4, you had the option to set a "No Access" flag, but you couldn't get specific about what permissions you wanted to deny someone. For example, if you wanted to deny someone the Write permission in NT 4, you had to set all the other permissions to allow access and leave the Write permission check box blank. As you can see from the interface, you can now explicitly deny someone the right to write! Denying Full Control has the same effect as the NT 4 "No Access" flag. All the Deny permission entries in a DACL will be at the top of the list. This improves the efficiency of the permissions checks. If a person or group is denied any one of the requested set of permissions, they are thrown out at the top of the list rather than having to read the whole DACL to determine that they don't have that Write permission.

When an access request comes into the system, the permissions that are requested are considered a set. This means that whether a person is allowed access is an "all or nothing" proposition. Let's say, for example, that I tried to open a document. This would spawn several requests. One request would be to Read and Execute the program's .exe file and its associated DLLs, plus there would probably be a second request for Modify access to the document file. You can see from the standard permissions list in [Table 3.1](#) what Modify means. If I am denied any of those permissions on the document, I will receive an "Access Denied" message. I will not get a message that says, "You can Read and Add Data to this document, but you can't Delete the document." That's what I mean by "all or nothing."

Permissions are cumulative in Windows 2000, as in NT 4. If a user is granted Read permission, but a

group he belongs to is granted Write and Delete, the user's access is Read, Write, and Delete. When trying to determine a user's access level to a resource, there are three ways a permissions check can be terminated. The DACL entries will be checked until one of the following three conditions is met:

- The full set of requested permissions has been satisfied by SID entries for the user or the groups of which the user is a member
- One or more of the requested permissions has been denied for the user or one of the groups of which the user is a member
- The entire DACL has been checked and there was no entry for one or more of the requested permissions; therefore, access is assumed to be denied

If you are troubleshooting a permissions problem and need to know the actual ordering of the DACL, you can view the DACL by using the Advanced button on the Permissions tab. This will bring up the screen in [Figure 3.4](#), which shows the ordering of the entries. From an administrative standpoint, you might find this screen easier to use because it gives you the whole list and the permissions levels without having to move the highlight bar. If you want to know specifically which special permissions have been set, simply highlight the entry in question and click on the View/Edit button. This will take you to the Special Permissions dialog box.

#### **Figure 3.4**

Access Control Settings screen used to view the ordering of the DACL on objects.

### **Setting Special Permissions**

As was mentioned earlier, an administrator or owner of a resource can be very granular about what permissions they want individuals to have. Besides being able to explicitly Allow or Deny a permission, the screen shot in [Figure 3.5](#) shows the special permissions that can be set on folders or files. The interface is a little more self-explanatory than it was in NT 4.

#### **Figure 3.5**

Permission Entry screen used to view the details of a particular DACL entry.

Naturally, whenever possible, you should use the standard permissions and set permissions using groups instead of individuals. Doing these two things and using the capabilities of the new inheritance model should make this part of your job as an administrator much easier. Permission inheritance will be discussed in the next section.

---

### **Shared Folder Permissions**

There are a few things to remember about shared file system resources and permissions. First, a share is nothing more than a folder that has been designated for network access. If you have a folder called DOCS that you share out, a user who logs on locally to your machine will be governed by the DACL on the local file system folder. When you share the folder out, you have the option to set an additional DACL for the share that would govern users coming into the resource over the network. Oh, and by the way, guess what the default permissions mask is on all shares...that's right: Everyone, Full. If there is a

conflict between the share DACL and the folder DACL, the most restrictive permissions set is used. For example, if you have an entry for Bob on the share DACL with Full Access set, and the folder DACL also has an entry for Bob, but with only Read access, Bob will only get Read privileges.

There are fewer choices for standard permissions settings on a share. Basically, you have Read, Change, and Full Control. The capability to set special permissions is not available at all on the share level. You can still set those special permissions at the folder level if you need them, and they will apply for your network users as well. The standard permissions that can be set on a share are shown in Table 3.2.

**Table 3.2 Standard Share Permissions**

<b>Standard Permission</b>	<b>Specific Permissions Granted as a Result</b>
Read	Traversing subfolders  Viewing data in files and running programs  Viewing filenames and subfolder names
Change	Adding files and folders to the shared folder  Changing data in files  Deleting subfolders and files  Traversing subfolder  Viewing data in files and running programs  Viewing filenames and subfolder names
Full Control	Adding files and folders to the shared folder  Changing data in files  Changing permissions (NTFS only)  Deleting subfolders and files  Taking ownership (NTFS only)

Traversing subfolders
Viewing data in files and running programs
Viewing filenames and subfolder names

## Permission Propagation and Inheritance

In NT 4, you could set permissions at the top levels of a file structure and then push that permissions mask down through the structure to subfolders and files, as long as you were the owner of the resources. You had some control over how far you wanted to push the mask, but there weren't a lot of options in the interface. After you set a new mask and chose to push it down, NT 4 had the rather annoying habit of replacing *all* permissions. So, if you had set special permissions somewhere down in the bowels of the file structure, these were erased and replaced with the new mask. There was no way to preserve those explicitly set permissions. Pretty aggravating after you had gone to all the trouble to set them in the first place!

Windows 2000 has improved on that model by recognizing two sets of DACLs on a resource: the inherited DACL and the explicit DACL. Now, it is possible to set a default mask at the top level, push it down, and still keep those special permissions you had to set for Bob last week.

Naturally, there are some new rules to know with this new capability. I mentioned previously that Deny permissions would be listed at the front of the DACL to improve access request response times. It's simply more efficient that way. Well, let's complicate that picture just a little. If you have both types of permissions set on a resource, explicit and inherited, the explicit permissions are always checked first. So, the real search order on that resource would be Explicit Denies, Explicit Allows, Inherited Denies, and finally Inherited Allows. [Figure 3.6](#) shows the Access Control Settings screen. I told you earlier that this screen was the easiest one to use to troubleshoot permissions, and you can see why if you look at it closely. I have created a DACL with both inherited and explicit permissions, as well as an Allow and a Deny for each type. They are listed on the interface in exactly the same order they are searched by Windows 2000. If you look closely at the first two entries, you will see that the icons to the left of the entries are nice and clear. These are explicit entries. The icons for the next two entries look a little fuzzy and faded. It's not your eyes; relax. These are the inherited entries. The icons are designed that way to give you a graphical clue to let you know which ones are explicit and which are inherited.

As you can see in that same figure, the explicit deny is listed first, followed by the explicit allow, then the inherited deny, and finally the inherited allow. The DACL will be searched in that order, and the three conditions listed earlier in the chapter will be used to terminate the request. That means *explicit permissions will override inherited permissions*. In our list, Lori will be denied Read access even though she inherits a Full Control permission from the parent folder. This happens because when the request comes in for access, the system will see the explicit deny listed first, terminate the request with an Access Denied message, and never search the list any farther, so it would never even see that Full Control entry. The logic here is that if the owner of the resource went to all the trouble to explicitly set a permission for Lori that is different from the inherited mask, there must be a reason

and therefore the explicit must be the more important permission.

### **Figure 3.6**

Using the Access Control Settings screen to view and troubleshoot file access problems.

If you don't need that much detail, you can get some of the same information on the first Security tab of the resource. Notice that inherited standard permissions are indicated by gray backgrounds in the check boxes, like those in [Figure 3.7](#), while explicitly set permissions have clear backgrounds, as you can see in [Figure 3.8](#). As always, you will have to move the highlight bar around to determine who has what.

### **Figure 3.7**

Using the Security Properties screen to view the inherited permissions set.

### **Figure 3.8**

Viewing explicit permissions using the Security Properties tab.

There is one small glitch in this first interface that I want to make sure you are aware of. Look at [Figures 3.7](#) and [3.8](#) again. Notice that there is only one entry for "Lori" on the first properties page, even though you can clearly see both entries listed on the Access Control Settings screen back in [Figure 3.6](#). Be aware that multiple entries for the same account might not show up. Even worse, the general page shows only that Lori has inherited Full Control permissions. It says nothing about that Explicit Deny for Read access. This could be very deceiving (and aggravating) if you are trying to troubleshoot a problem. This is another argument for just going to the Access Control Settings screen in the first place.

So, if these explicit entries are going to be preserved, what do you do if you mess up the mask and need to reset the permissions to the inherited default? It takes a little work and forethought, but you can still recover from this type of mistake. One option is to simply go in and remove the offending explicit entries and then propagate that mask down to the files and subfolders (I'll tell you how to do that in the next section). If the mask is so messed up that you can't determine for sure which entries are good and which are bad, but you know they are correct above that point in the tree, you can simply move up in the file tree to the level where they are correct and choose to propagate the correct mask from there. If you are at the top level and moving up is not an option, you can do it the old-fashioned way. If there is a correct mask anywhere on the partition, copy the tree structure so that it becomes a child of the resource with the correct mask. Then, check that the permissions are correct and move the structure back to its original spot. Remember the mantra: move retains, copy inherits. It works for permissions and compression states. This method only works within a single partition, however, because moves across partitions behave like copies, and in that case, all permissions will be inherited from the target parent folder.

You can think of propagation of permissions as pushing permissions down the file system tree. Inheritance, on the other hand, has to do with receiving a permissions mask that has been propagated. On the interfaces I have shown you, there are actually a couple of entries that deal with permission propagation and inheritance. Let's look at those two functions and their associated interface entries separately so you get a better understanding of each.

## **Propagation Controls**

There are basically three screens we have discussed that are part of the permissions module in Windows 2000:

- The Security Properties screen, where you see the standard permissions
- The Access Control Settings screen, reached by using the Advanced button on the Security Properties screen
- The Permission Entry screen, which gives the details for each entry on the DACL

The last two screens have entries that control propagation. Earlier we discussed the problem of needing to replace an incorrect permissions mask that contained explicit entries. I mentioned that this is a problem because those explicit settings are preserved by default and checked before the inherited entries in the DACL. If that's the case and you want to replace the whole set of permissions on all subfolders and files, simply go to the Access Control Settings screen and check the box shown with the arrow in [Figure 3.9](#).

### **Figure 3.9**

Using the Access Control Settings screen to replace explicit permissions and propagate the desired mask down through the file tree.

Because this is such a drastic action, you get a warning message telling you that this will remove all explicitly set permissions and turn for inheritance for all subfolders and files. If you are familiar with NT 4, you should know that turning on this check box makes the system act like the NT 4 file system did when you elected to replace permissions on subfolders and files. As you might imagine, the system default is for this check box to be blank.

If, on the other hand, you only want to propagate selected DACL entries or have a need to finely control how far a particular entry should go, use the Permission Entry dialog box. As you can see in [Figure 3.10](#), by using this interface, you can be very selective about just where you want this particular entry to be applied. I think this is pretty intuitive. Just be aware of the granularity you have in this interface. By the way, the default is for the entry to be applied to "This folder, subfolders and files."

### **Figure 3.10**

Using the Permission Entry dialog box to precisely control the application of permissions and limit propagation to the desired level.

At the bottom of that dialog box you will notice another check box. This one says "Apply these permissions to objects and/or containers within this container only." If you turn this feature on, propagation will be limited to the folder and its immediate children. Its children are defined as the subfolders and files that exist within that folder. The grandchildren objects—the files that exist in a subfolder, for example—will not have the mask applied to them.

These two boxes work together. If you do not limit the propagation by checking the box at the bottom of the screen and select to have these permissions applied to "Subfolders only," all the subfolders at all levels below this one will have the permissions reset. So, if you have five levels of subfolders, all will be affected. On the other hand, if you choose to limit the propagation, only this folder and one

level of subfolders will be set.

### **Controlling Inheritance**

Inheritance can be controlled from either the Security tab or the Access Control Settings screen by simply checking the box that says "Allow inheritable permissions from parent to propagate to this object." This box is available on either screen. As with many things in Windows 2000 interfaces, enabling and disabling inheritance in this way is incredibly easy. To block inheritance of upper-level permission masks, simply clear the check box.

What's a little more difficult is understanding what happens to permissions when you change this setting. If that little check box is turned on and you choose to disable the setting, Windows 2000 will ask you what you want to do with the current settings. Your choices are:

- Copy the current set to the resource and make them explicit permissions
- Remove all the permissions and start with a clean slate
- Oops...I don't really want to do this! Also known as the Cancel button

No matter what you choose to do, you always have the option to re-enable inheritance in the future and reapply the inherited DACL. Should you decide to do that, just remember that you will now have two DACLs, an explicit and an inherited, and that the explicit settings will take priority over your inheriteds.

The default setting is that inheritance is enabled on all file system objects, so beware. When you make changes to masks at the top levels, they can propagate to lower levels as long as inheritance is turned on at those lower-level resources. To be honest, I've messed up a whole file system this way. "Gee, all I did was click OK instead of Cancel." Wow, that one hurt! I'm trying to save you from suffering that pain.

## **Auditing Security Access Control Lists**

Now that you've got your permissions set up the way you want them, you might want to keep track of who is trying to get into those resources. This is accomplished through the application of file system auditing.

Turning on the capability to audit a resource is a two-step process in Windows 2000. In the first step, you have to enable Audit Object Access, using the Group Policy snap-in for MMC. If you forget this step, don't worry, Windows 2000 will remind you with an error message when you try to set up auditing on a particular file or folder. It will accept the entry, but until you do turn it on, no auditing will take place. This can be accomplished either through the domain group policies or the Local Computer Policy Editor, if there is no domain group policy being used.

Another thing to remember: In order to enable this setting or audit, you must be logged on with an account that is a member of the local machine's Administrators group, or have been given the right to "Manage Auditing and Security Log."

The second step in setting up auditing is pretty easy. Go to the resource, right-click, and head for that Access Control Settings screen. You'll see that the middle tab is the Auditing tab. There, the interface looks a lot like the interface to add a user to the DACL. Select Add, choose the user or group you want to audit, then choose the event you want to audit and whether you want to audit successes or failures of that event. As you add entries, a list is built. This list is called the Security Access Control List (SACL). This list is kept in addition to the DACL. If you look closely at the interface, you'll notice that the text for the inheritable permissions check boxes has changed. They now mention inheriting and replacing inheritable *auditing* entries, as indicated by the arrows in [Figure 3.11](#). You can also go one step further and use the Permission Entry screen to precisely set an audit setting or limit its propagation in exactly the same way you set and propagated the DACL, except that this time you use the Audit tab instead of the Permissions tab.

---

### Editing Local Group Policy Objects

You won't find the Local Group Policy Editor installed on the menu bars of a Windows 2000 system. In order to run this, you'll need to start an MMC console using Start+Run and typing in `mmc`. When the console starts, select Console and select Add or Remove Snap In from the drop-down menu. Choose Add and select Group Policy from the list. You will get a dialog box that tells you the Group Policy object selected is the Local Computer. Click Finish, and you'll return to the console, which will now have an entry for Local Computer Policy.

To turn on auditing for objects on the local machine, select Computer Configuration and drill down through Windows Settings, Security Settings, Local Policies, and finally Audit Policy. Expand this and double click on Audit Object Access. When the dialog box appears, turn on the check boxes to enable auditing on successful events and/or failed events. Select OK and you will be returned to the main console screen. The Local Setting column entry should have changed from the default of No Auditing to an entry that says Success, Failure. The Effective Setting column might still say No Auditing. Don't worry, this will change when you exit the console and re-enter. You do not have to save your settings as you exit the console to have these changes take effect. When the system asks you if you want to save the console settings, it is referring to the way the console appears; what trees you have open for example, not the changes that you made. Those were applied when you hit the OK button in the dialog box. Now, you have the capability to go to a resource and set auditing for either successful access events or failed events.

---

### **Figure 3.11**

Viewing the SACL and determining SACL propagation limits.

Auditing activities are recorded in the system's security log. You can view the results of your auditing efforts through the Event Viewer. You can get to the Event Viewer several ways, but the easiest is going to the Administrative Tools menu and selecting it from there. Event Viewer hasn't changed much since NT 4. After you're in the security log, you'll see that successful access events are still represented by keys and failures by padlocks. To see the details on an event, double-click on the event. Pretty simple.

One last thing. I always get a kick out of the calls I get from paranoid security folks who have just discovered this auditing function. Typically, they aren't very selective about what they audit. Many just turn on all the settings and audit everything! Not long after they do this, I get a call. Their complaint is usually that their machine has turned into a slug. Yeah, right after they turned on auditing! Please remember that the very act of auditing and producing these log entries carries an overhead in terms of performance. My advice is to be selective about what you choose to audit. Give it some thought and remember that opening one icon can spawn several programs. Teach your users that too, especially those security officers!

## A Few Last Words

In this chapter, we have covered how Windows 2000 handles authentication and how to protect your file system resources. As I mentioned at the start of the chapter, there is a whole lot more to know about Windows 2000 security mechanisms. When Microsoft started delivering the betas for this product, people began predicting that no one would be able to know it all because the product was so huge. They predicted that specialties would develop within the administrative ranks—that there would be desktop/user administrators, network experts, security experts, Active Directory administrators. My focus here has been to help you become more of an expert in desktop/user security issues. As I said, we'll touch on other security issues throughout the rest of the book as they come up in our conversation. If you get done and still want to know more, check out <http://www.microsoft.com/> and browse the Windows 2000 security white papers to get the details. If I were you, though, I'd be sure and get some caffeine first! The papers are well written, but those details are...how should I say this? Well, let's just say they can cure insomnia!

© Copyright Pearson Education. All rights reserved.