Microsoft TechNet ©2008 Microsoft Corporation. All rights reserved.

Windows 2000 Kerberos Authentication

Abstract

The next generation of the Microsoft Windows operating system will adopt Kerberos as the default protocol for network authentication. An emerging standard, Kerberos provides a foundation for interoperability while enhancing the security of enterprise-wide network authentication.

Windows 2000 implements Kerberos version 5 with extensions for public key authentication. The Kerberos client is implemented as a security provider through the Security Support Provider Interface. Initial authentication is integrated with the Winlogon single sign-on architecture. The Kerberos Key Distribution Center (KDC) is integrated with other Windows 2000 security services running on the domain controller and uses the domain's Active Directory as its security account database.

This white paper examines components of the protocol and provides detail on its implementation.

On This Page

Introduction

This paper provides a technical introduction to how the Microsoft® Windows® 2000 operating system implements the Kerberos version 5 authentication protocol. The paper includes detailed explanations of important concepts, architectural elements, and features of Kerberos authentication. The first section, "Overview of the Kerberos Protocol," is for anyone unfamiliar with Kerberos authentication. Following this introduction to the protocol are several sections with details of Microsoft's implementation in Windows 2000. The paper concludes with a brief discussion of requirements for interoperability with other implementations.

Authentication in Windows 2000

Windows 2000 supports several protocols for verifying the identities of users who claim to have accounts on the system, including protocols for authenticating dial-up connections and protocols for authenticating external users who access the network over the Internet. But there are only two choices for network authentication within Windows 2000 domains:

- **Kerberos Version 5.** The Kerberos version 5 authentication protocol is the default for network authentication on computers with Windows 2000.
- **Windows NT LAN Manager (NTLM).** The NTLM protocol was the default for network authentication in the Windows NT® 4.0 operating system. It is retained in Windows 2000 for compatibility with downlevel clients and servers. NTLM is also used to authenticate logons to standalone computers with Windows 2000.

Computers with Windows 3.11, Windows 95, Windows 98, or Windows NT 4.0 will use the NTLM protocol for network authentication in Windows 2000 domains. Computers running Windows 2000 will use NTLM when authenticating to servers with Windows NT 4.0 and when accessing resources in Windows NT 4.0 domains. But the protocol of choice in Windows 2000, when there is a choice, is Kerberos version 5.

Benefits of Kerberos Authentication

One of the design goals of Windows 2000 is to enable administrators to turn off NTLM authentication once all network clients are capable of Kerberos authentication. The Kerberos protocol is more flexible and efficient than NTLM, and more secure. The benefits gained by using Kerberos authentication are:

- **Faster connections.** With NTLM authentication, an application server must connect to a domain controller in

order to authenticate each client. With Kerberos authentication, the server does not need to go to a domain controller. It can authenticate the client by examining credentials presented by the client. Clients can obtain credentials for a particular server once and reuse them throughout a network logon session.

- **Mutual authentication.** NTLM allows servers to verify the identities of their clients. It does not allow clients to verify a server's identity, or one server to verify the identity of another. NTLM authentication was designed for a network environment in which servers were assumed to be genuine. The Kerberos protocol makes no such assumption. Parties at both ends of a network connection can know that the party on the other end is who it claims to be.

- **Delegated authentication.** Windows services impersonate clients when accessing resources on their behalf. In many cases, a service can complete its work for the client by accessing resources on the local computer. Both NTLM and Kerberos provide the information that a service needs to impersonate its client locally. However, some distributed applications are designed so that a front-end service must impersonate clients when connecting to back-end services on other computers. The Kerberos protocol has a proxy mechanism that allows a service to impersonate its client when connecting to other services. No equivalent is available with NTLM.

- **Simplified trust management.** One of the benefits of mutual authentication in the Kerberos protocol is that trust between the security authorities for Windows 2000 domains is by default two-way and transitive. Networks with multiple domains no longer require a complex web of explicit, point-to-point trust relationships. Instead, the many domains of a large network can be organized in a tree of transitive, mutual trust. Credentials issued by the security authority for any domain are accepted everywhere in the tree. If the network includes more than one tree, credentials issued by a domain in any tree are accepted throughout the forest.

- **Interoperability.** Microsoft's implementation of the Kerberos protocol is based on standards-track specifications recommended to the Internet Engineering Task Force (IETF). As a result, the implementation of the protocol in Windows 2000 lays a foundation for interoperability with other networks where Kerberos version 5 is used for authentication.

Standards for Kerberos Authentication

The Kerberos protocol originated at MIT more than a decade ago, where it was developed by engineers working on Project Athena[1]. The first public release of the protocol was Kerberos version 4. Following wide industry review of the protocol, its authors developed and released Kerberos version 5. That version is now on a standards track with the IETF. The implementation of the protocol in Windows 2000 closely follows the specification defined in Internet RFC 1510.[2] In addition, the mechanism and format for passing security tokens in Kerberos messages follows the specification defined in Internet RFC 1964.[3]

Extensions to the Kerberos Protocol

Windows 2000 implements extensions to the Kerberos protocol that permit initial authentication using public key certificates rather than conventional shared secret keys. This enhancement allows the protocol to support interactive logon with a smart card. The extensions for public key authentication are based on a draft specification submitted to the IETF working group by a number of third parties with interests in public key technology.[4] Microsoft is participating in the standards process and will continue to support its efforts.

Top of page

Overview of the Kerberos Protocol

The Kerberos authentication protocol provides a mechanism for mutual authentication between a client and a server, or between one server and another, before a network connection is opened between them. The protocol assumes that initial transactions between clients and servers take place on an open network where most computers are not physically secure, and packets traveling along the wire can be monitored and modified at will. The assumed environment, in other words, is very much like today's Internet, where an attacker can easily pose as either a client or a server, and can readily eavesdrop on or tamper with communications between legitimate clients and servers.

Basic Concepts

The Kerberos protocol relies heavily on an authentication technique involving shared secrets. The basic concept is quite simple: If a secret is known by only two people, then either person can verify the identity of the other by confirming that the other person knows the secret.

For example, let's suppose that Alice often sends messages to Bob and that Bob needs to be sure that a message from Alice really has come from Alice before he acts on its information. They decide to solve their problem by selecting a password, and they agree not to share this secret with anyone else. If Alice's messages can somehow demonstrate that the sender knows the password, Bob will know that the sender is Alice.

The only question left for Alice and Bob to resolve is *how* Alice will show that she knows the password. She could simply include it somewhere in her messages, perhaps in a signature block at the end—*Alice, Our$ecret*. This would be simple and efficient and might even work if Alice and Bob can be sure that no one else is reading their mail. Unfortunately, that is not the case. Their messages pass over a network used by people like Carol, who has a network analyzer and a hobby of scanning traffic in hope that one day she might spot a password. So it is out of the question for Alice to prove that she knows the secret simply by saying it. To keep the password secret, she must show that she knows it without revealing it.

The Kerberos protocol solves this problem with secret key cryptography. Rather than sharing a password, communication partners share a cryptographic key, and they use knowledge of this key to verify one another's identity. For the technique to work, the shared key must be *symmetric*—a single key must be capable of both encryption and decryption. One party proves knowledge of the key by encrypting a piece of information, the other by decrypting it.

Authenticators

A simple protocol that uses secret key authentication begins when someone is outside a communications door and wants to go in. To gain entry, this person presents an **authenticator** in the form of a piece of information encrypted in the secret key. The information in the authenticator must be different each time the protocol is executed, otherwise an old authenticator could be reused by anyone who happens to overhear the communication. On receiving an authenticator, the person guarding the door decrypts it and knows from what is inside whether decryption was successful. If it was successful, the doorkeeper knows that the person presenting the authenticator has the correct key. Only two people have the correct key; the doorkeeper is one of them, so the person who presented the authenticator must be the other.

If the person outside the door wants mutual authentication, the same protocol can be executed in reverse, with a slight difference. The doorkeeper can extract part of the information from the original authenticator, encrypt it in a new authenticator, and then give the new authenticator to the person waiting outside the door. The person outside the door can then decrypt the doorkeeper's authenticator and compare the result with the original. If there is a match, the person outside the door will know that the doorkeeper was able to decrypt the original, so he must have the correct key.

It will help to walk through an example. Suppose Alice and Bob decide that before transferring any information between their computers, each will use knowledge of a shared secret key to verify the identity of the party at the other end of the connection. In situations where Alice is the wary guest and Bob is the suspicious host, they agree to follow this protocol:

1. Alice sends Bob a message containing her name in plaintext and an authenticator encrypted in the secret key she shares with Bob. In this protocol, the authenticator is a data structure with two fields. One field contains information about Alice. For simplicity, let's say this is another instance of her name. The second field contains the current time on Alice's workstation.

2. Bob receives the message, sees that it is from someone claiming to be Alice, and uses the key he shares with Alice to decrypt the authenticator. He extracts the field that contains the time on Alice's workstation, and evaluates the time.

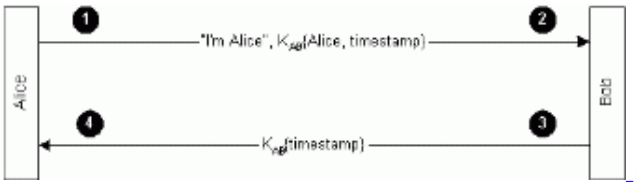
   Bob's task will be easier if his clock is reasonably synchronized with Alice's, so let's suppose both Alice and Bob use a network time service to keep their clocks fairly close. Let's say the time skew is never more than five minutes. This way, Bob can compare the time from the authenticator with the current time on his clock. If the difference is greater than five minutes, he can automatically reject the authenticator.

   If the time is within the allowable skew, it's probable that the authenticator came from Alice, but Bob still does not have proof that the authenticator actually came from her. Another person might have been watching network traffic and might now be replaying an earlier attempt by Alice to establish a connection with Bob. However, if Bob has recorded the times of authenticators received from Alice during the past five minutes, he can defeat attempts to replay earlier messages by rejecting any message with a time that is the same as or earlier than the time of the last authenticator. If this authenticator yields a time later than the time of the last authenticator from Alice, then this message must be from Alice.

3. Bob uses the key he shares with Alice to encrypt the time taken from Alice's message and sends the result back to her.

Note that Bob does not send back all of the information taken from Alice's authenticator, just the time. If he sent back everything, Alice would have no way of knowing whether someone posing as Bob had simply copied the authenticator from her original message and sent it back to her unchanged. He sends just a piece of the information in order to demonstrate that he was able to decrypt the authenticator and manipulate the information inside. He chooses the time because that is the one piece of information that is sure to be unique in Alice's message to him.

Alice receives Bob's reply, decrypts it, and compares the result with the time in her original authenticator. If the times match, she can be confident that her authenticator reached someone who knows the secret key needed to decrypt it and extract the time. She shares that key only with Bob, so it must be Bob who received her message and replied.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs01_big(en-us,TechNet.10).gif]

Figure 1: Mutual authentication (Alice-Bob)

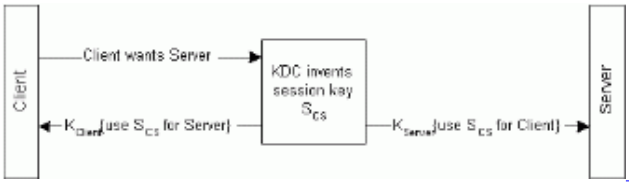
Key Distribution

One problem with the simple protocol described in the preceding section is that it does not explain how or where Alice and Bob got a secret key to use in sessions with each other. If they are people, Alice and Bob could meet, perhaps in an alley, and agree on a secret key. But that method will not work if Alice is a client program that is running on a workstation and Bob is a service that is running on a network server. There is also the further problem that the client, Alice, will want to talk to many servers and will need keys for each of them. Likewise, the service, Bob, will talk to many clients and will need keys for each of them as well. If each client needs to have a key for every service, and each service needs one for every client, key distribution could quickly become a tough problem to solve. And the need to store and protect so many keys on so many computers would present an enormous security risk.

The name of the Kerberos protocol suggests how it solves the problem of key distribution. Kerberos (or *Cerberus*) was a figure in classical Greek mythology, a fierce, three-headed dog who guarded the gates of the Underworld. Like Kerberos the guard, Kerberos the protocol has three heads: a client, a server, and a trusted third party to mediate between them. The trusted intermediary in the protocol is known as the **Key Distribution Center** (KDC).

The KDC is a service that runs on a physically secure server. It maintains a database with account information for all security principals in its *realm,* the Kerberos equivalent of a Windows 2000 domain. (We will continue to call them domains.) Along with other information about each security principal, the KDC stores a cryptographic key known only to the security principal and the KDC. This key is used in exchanges between the security principal and the KDC and is known as a **long-term key**. In most implementations of the protocol, it is derived from a user's logon password.

When a client wants to talk to a server, the client sends a request to the KDC, and the KDC distributes a unique, short-term **session key** for the two parties to use when they authenticate each other. The server's copy of the session key is encrypted in the server's long-term key. The client's copy of the session key is encrypted in the client's long-term key.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs02_big(en-us,TechNet.10).gif]

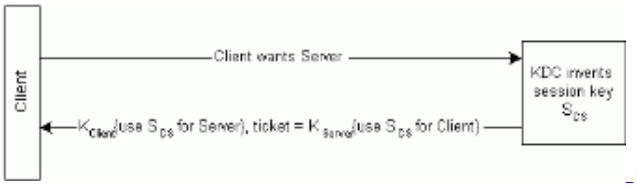
Figure 2: Key Distribution (in theory)

In theory, the KDC could fulfill its role as a trusted intermediary by sending the session key directly to each of the security principals involved, as illustrated above. But, in practice, that procedure would be extremely difficult to implement. For one thing, it would mean that the server would have to retain its copy of the session key in memory while it waited for the client to call. Moreover, the server would need to remember a key not just for this client but

for every client who might ask for service. Key management would consume considerable resources on the server and would thus limit its scalability. In addition, given the vagaries of network traffic, a client's request for service might reach the server before the KDC's message arrived there with the session key. The server would have to suspend its reply to the client while it waited to hear from the KDC. This would require the server to save state, imposing still another burden on the server's resources. What actually happens in the Kerberos protocol is considerably more efficient.

Session Tickets

The KDC responds to the client's request to talk to a server by sending both copies of the session key to the client, as shown in Figure 3. The client's copy of the session key is encrypted with the key that the KDC shares with the client. The server's copy of the session key is embedded, along with information about the client, in a data structure called a **session ticket**. The entire structure is then encrypted with the key that the KDC shares with the server. The ticket—with the server's copy of the session key safely inside—becomes the client's responsibility to manage until it contacts the server.

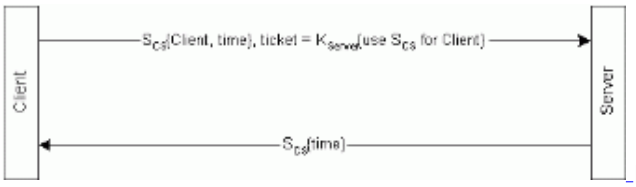


[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs03_big(en-us,TechNet.10).gif]

Figure 3: Key Distribution (in practice)

Note that the KDC is simply providing a ticket-granting service. It does not keep track of its messages to make sure they reach the intended address. No harm will be done if the KDC's messages fall into the wrong hands. Only someone who knows the client's secret key can decrypt the client's copy of the session key. Only someone who knows the server's secret key can read what is inside the ticket.

When the client receives the KDC's reply, it extracts the ticket and the client's copy of the session key, putting both aside in a secure cache (located in volatile memory, not on disk). When the client wants admission to the server, it sends the server a message that consists of the ticket, which is still encrypted with the server's secret key, and an authenticator, which is encrypted with the session key. The ticket and authenticator together are the client's credentials to the server.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs04_big(en-us,TechNet.10).gif]

Figure 4: Mutual authentication (Client/server)

When the server receives credentials from a client, it decrypts the session ticket with its secret key, extracts the session key, and uses the session key to decrypt the client's authenticator. If everything checks out, the server knows that the client's credentials were issued by a trusted authority, the KDC. If the client has asked for mutual authentication, the server uses its copy of the session key to encrypt the timestamp from the client's authenticator and returns the result to the client as the server's authenticator.

One benefit gained by using session tickets is that the server does not have to store the session key that it uses in communicating with this client. It is the client's responsibility to hold a ticket for the server in its credentials cache and present the ticket each time it wants access to the server. Whenever the server receives a session ticket from a client, it can use its secret key to decrypt the ticket and extract the session key. When the server no longer needs the session key, it can discard it.

Another benefit is that the client does not need to go back to the KDC each time it wants access to this particular server. Session tickets can be reused. As a precaution against the possibility that someone might steal a copy of a ticket, session tickets have an expiration time, specified by the KDC in the ticket's data structure. How long a ticket is valid depends on Kerberos policy for the domain. Typically, tickets are good for no longer than eight hours, about the length of a normal logon session. When the user logs off, the credentials cache is flushed and all session tickets—as well as all session keys—are destroyed.

Ticket-Granting Tickets

A user's long-term key is derived from a password. When Alice logs on, for example, the Kerberos client on her workstation accepts her password and then converts it to a cryptographic key by passing the text of the password through a one-way hashing function. (All implementations of Kerberos version 5 must support DES-CBC-MD5. Other algorithms are permissible.) The result is Alice's long-term key.

The KDC gets its copy of Alice's long-term key from her record in its account database. When it receives a request from the Kerberos client on Alice's workstation, the KDC searches its database for Alice, pulls up her account record, and takes her long-term key from a field in the record.

This process—computing one copy of the key from a password, fetching another copy of the key from a database—actually takes place only once, when a user initially logs on to the network. Immediately after accepting the user's password and deriving the user's long-term key, the Kerberos client on the workstation requests a session ticket and session key that it can use in subsequent transactions with the KDC during this logon session.

The KDC responds to the client's request by returning a session ticket for itself. This special session ticket is called a **ticket-granting ticket** (TGT). Like an ordinary session ticket, a TGT contains a copy of the session key that the service (in this case the KDC) will use in communicating with the client. The message that returns the TGT to the client also includes a copy of the session key that the client can use in communicating with the KDC. The TGT is encrypted in the KDC's long-term key. The client's copy of the session key is encrypted in the user's long-term key.

When the client receives the KDC's reply to its initial request, it uses its cached copy of the user's long-term key to decrypt its copy of the session key. It can then discard the long-term key derived from the user's password, for it is no longer needed. In all subsequent exchanges with the KDC, the client uses the session key. Like any other session key, this key is temporary, valid only until the TGT expires or the user logs off. For that reason, it is called a **logon session key**.

From the client's point of view, a TGT is just another ticket. Before it attempts to connect to any service, the client first checks its credentials cache for a session ticket to that service. If it does not have one, it checks the cache again for a TGT. If it finds a TGT, the client fetches the corresponding logon session key from the cache, uses this key to prepare an authenticator, and sends both the authenticator and the TGT to the KDC, along with a request for a session ticket for the service. In other words, gaining admission to the KDC is no different from gaining admission to any other service in the domain—it requires a session key, an authenticator, and a ticket (in this case, a TGT).

From the KDC's point of view, TGTs allow it to shave a few nanoseconds off the turnaround time for ticket requests. The KDC looks up the user's long-term key only once, when it grants an initial TGT. For all other exchanges with this client, the KDC can decrypt the TGT with its own long-term key, extract the logon session key, and use that to validate the client's authenticator.

Authentication Across Domain Boundaries

The functions of the KDC are divided into two distinct services: an authentication service whose job is to issue TGTs, and a ticket-granting service whose job is to issue session tickets. This division of labor allows the Kerberos protocol to operate across domain boundaries. A client can get a TGT from the authentication service of one domain and use it to get session tickets from the ticket-granting service of another domain.

To see how cross-domain authentication works, let's first consider the simplest case: a network with only two domains, East and West. If administrators for these domains are members of the same organization, or if for some other reason they are willing to treat the other domain's users as their own, they can enable authentication across domain boundaries simply by sharing an inter-domain key. (In Windows 2000 this happens automatically when two domains establish a trust relationship.) Once this is accomplished, the ticket-granting service of each domain is registered as a security principal with the other domain's KDC. As a result, the ticket-granting service in each domain can treat the ticket-granting service in the other domain as just another service, something for which properly authenticated clients can request and receive session tickets.

When a user with an account in East wants access to a server with an account in West, the Kerberos client on the user's workstation sends a request for a session ticket to the ticket-granting service in the user's account domain, East. The ticket-granting service in East sees that the desired server is not a security principal in its domain, so it replies by sending the client a **referral ticket**. This is simply a TGT encrypted with the inter-domain key that the KDC in East shares with the KDC in West. The client uses the referral ticket to prepare a second request for a session ticket, and this time sends the request to the ticket-granting service in the server's account domain, West. The ticket-granting service in West uses its copy of the inter-domain key to decrypt the referral ticket. If decryption is successful, it sends the client a session ticket to the desired server in its domain.

The referral process is more complicated on networks with more than two domains. In theory, the KDC in each domain could establish a direct link to the KDC in every other domain on the network, in each case sharing a different inter-domain key. In practice, the number and complexity of these relationships could easily become unmanageable, especially on a large network. The Kerberos protocol solves the problem by making direct links unnecessary. A client in one domain can get a ticket to a server in another domain by traveling a referral path through one or more intermediate domains.

For example, consider a network with three domains, East, West, and CorpHQ. The KDC in East does not share an inter-domain key with the KDC in West, but both East and West do share inter-domain keys with CorpHQ. In this case, when a user with an account in East wants access to a server with an account in West, the referral path begins at the KDC for the user's account domain, East, passes through an intermediate domain, CorpHQ, and ends at the KDC for the server's account domain, West. The client must send its request for a session ticket three times, to three different KDCs.

1. The client asks the KDC for East to give it a ticket to the server in West.

   The KDC for East sends the client a referral ticket to the KDC for CorpHQ. This ticket is encrypted in the inter-domain key East shares with CorpHQ.

2. The client asks the KDC for CorpHQ to give it a ticket to the server in West.

   The KDC for CorpHQ sends the client a referral ticket to the KDC for West. This ticket is encrypted in the inter-domain key CorpHQ shares with West.

3. The client asks the KDC for West to give it a ticket to the server in West.

   The KDC for West sends back a ticket for the server.

Subprotocols

The Kerberos protocol is comprised of three subprotocols. The subprotocol in which the KDC gives the client a logon session key and a TGT is known as the Authentication Service (AS) Exchange. The subprotocol in which the KDC distributes a service session key and a session ticket for the service is known as the Ticket-Granting Service (TGS) Exchange. The subprotocol in which the client presents the session ticket for admission to a service is called the Client/Server (CS) Exchange.

To see how the three subprotocols work together, let's look at how Alice, a user at a workstation, gets access to Bob, a service on the network.

AS Exchange

Alice begins by logging on to the network. She types her logon name and her password. The Kerberos client on Alice's workstation converts her password to an encryption key and saves the result in its credentials cache.

The client then sends the KDC's authentication service a Kerberos Authentication Service Request (KRB_AS_REQ). The first part of this message identifies the user, Alice, and the name of the service for which she is requesting credentials, the ticket-granting service. The second part of the message contains **pre-authentication data** that proves Alice knows the password. This is usually a timestamp encrypted with Alice's long-term key, although the protocol permits other forms of pre-authentication data.

[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs05_big(en-us,TechNet.10).gif]

Figure 5: AS Exchange

When the KDC receives KRB_AS_REQ, it looks up the user Alice in its database, gets her long-term key, decrypts the pre-authentication data, and evaluates the timestamp inside. If the timestamp passes the test, the KDC can be assured that the pre-authentication data was encrypted with Alice's long-term key and thus that the client is genuine.

After it has verified Alice's identity, the KDC creates credentials that the Kerberos client on her workstation can present to the ticket-granting service. First, the KDC invents a logon session key and encrypts a copy of it with Alice's long-term key. Second, it embeds another copy of the logon session key in a TGT, along with other information about Alice such as her authorization data. The KDC encrypts the TGT with its own long-term key. Finally, it sends both the encrypted logon session key and the TGT back to the client in a Kerberos Authentication Service Reply (KRB_AS_REP).

When the client receives the message, it uses the key derived from Alice's password to decrypt her logon session key and stores the key in its credentials cache. Then it extracts the TGT from the message and stores that in its credentials cache as well.

TGS Exchange

The Kerberos client on Alice's workstation requests credentials for the service Bob by sending the KDC a Kerberos Ticket-Granting Service Request (KRB_TGS_REQ). This message includes the user's name, an authenticator encrypted with the user's logon session key, the TGT obtained in the AS Exchange, and the name of the service for which the user wants a ticket.

[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs06_big(en-us,TechNet.10).gif]

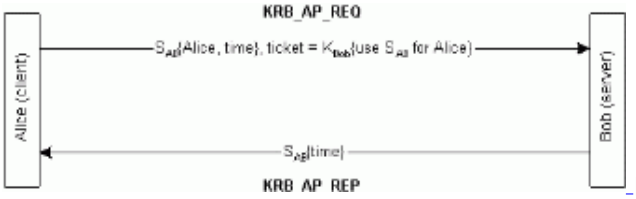
Figure 6: TGS Exchange

When the KDC receives KRB_TGS_REQ, it decrypts the TGT with its own secret key, extracting Alice's logon session key. It uses the logon session key to decrypt the authenticator and evaluates that. If the authenticator passes the test, the KDC extracts Alice's authorization data from the TGT and invents a session key for the client, Alice, to share with the service, Bob. The KDC encrypts one copy of this session key with Alice's logon session key. It embeds another copy of the session key in a ticket, along with Alice's authorization data, and encrypts the ticket with Bob's long-term key. The KDC then sends these credentials back to the client in a Kerberos Ticket-Granting Service Reply (KRB_TGS_REP).

When the client receives the reply, it uses Alice's logon session key to decrypt the session key to use with the service, and stores the key in its credentials cache. Then it extracts the ticket to the service and stores that in its cache.

CS Exchange

The Kerberos client on Alice's workstation requests service from Bob by sending Bob a Kerberos Application Request (KRB_AP_REQ). This message contains an authenticator encrypted with the session key for the service, the ticket obtained in the TGS Exchange, and a flag indicating whether the client wants mutual authentication. (The setting of this flag is one of the options in configuring Kerberos. The user is never asked.)



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs07_big(en-us,TechNet.10).gif]

Figure 7: CS Exchange

The service, Bob, receives KRB_AP_REQ, decrypts the ticket, and extracts Alice's authorization data and the session key. Bob uses the session key to decrypt Alice's authenticator and then evaluates the timestamp inside. If the authenticator passes the test, Bob looks for a mutual authentication flag in the client's request. If the flag is set, he uses the session key to encrypt the time from Alice's authenticator and returns the result in a Kerberos Application Reply (KRB_AP_REP).

When the client on Alice's workstation receives KRB_AP_REP, it decrypts Bob's authenticator with the session key it shares with Bob and compares the time returned by the service with the time in the client's original authenticator. If the times match, the client knows that the service is genuine, and the connection proceeds. During the connection, the session key can be used to encrypt application data, or the client and server can share another key for this purpose.

Tickets

So far we have avoided a detailed description of exactly what is in a ticket, how expiration times are calculated, and how much of a ticket's content is known by the client. All of these details are important to understanding how to

configure Kerberos policy, and so they deserve a closer look.

What Is in a Ticket

For our purpose here, it is enough to list the fields in a ticket and to describe the information they contain. The exact data structures for tickets as well as messages can be found in RFC 1510.

Table 1 Fields of a Ticket

Field Name	Description
The first three fields in a ticket are not encrypted. The information is in plaintext so that the client can use it to manage tickets in its cache.	
tkt-vno	Version number of the ticket format. In Kerberos v.5 it is 5.
Realm	Name of the realm (domain) that issued the ticket. A KDC can issue tickets only for servers in its own realm, so this is also the name of the server's realm.
Sname	Name of the server.
The remaining fields are encrypted with the server's secret key.	
Flags	Ticket options.
Key	Session key.
Crealm	Name of the client's realm (domain).
Cname	Client's name.
Transited	Lists the Kerberos realms that took part in authenticating the client to whom the ticket was issued.
Authtime	Time of initial authentication by the client. The KDC places a timestamp in this field when it issues a TGT. When it issues tickets based on a TGT, the KDC copies the authtime of the TGT to the authtime of the ticket.
Starttime	Time after which the ticket is valid.
Endtime	Ticket's expiration time.
renew-till	(Optional) Maximum endtime that may be set in a ticket with a RENEWABLE flag.
Caddr	(Optional) One or more addresses from which the ticket can be used. If omitted, the ticket can be used from any address.
Authorization-data	(Optional) Privilege attributes for the client. Kerberos does not interpret the contents of this field. Interpretation is left up to the service.

The *flags* field is a bit-field in which options are set by turning a particular bit on (1) or off (0). Although the field is 32 bits long, only nine ticket flags are of interest to Kerberos administrators.

Table 2 Ticket Flags

Flag	Description
FORWARDABLE	(TGT only) Tells the ticket-granting service that it can issue a new TGT with a different network address based on the presented TGT.
FORWARDED	Indicates either that a TGT has been forwarded or that a ticket was issued from a forwarded TGT.
PROXIABLE	(TGT only) Tells the ticket-granting service that it can issue tickets with a different network address than the one in the TGT.
PROXY	Indicates that the network address in the ticket is different from the one in the TGT used to obtain the ticket.
RENEWABLE	Used in combination with the *endtime* and *renew-till* fields to cause tickets with long life spans to be renewed at the KDC periodically.
INITIAL	(TGT only) Indicates that this is a TGT.

What Clients Know About Tickets

Clients need to know some of the information that is inside tickets and TGTs in order to manage their credentials cache. When the KDC returns a ticket and session key as the result of an AS or TGS Exchange, it packages the client's copy of the session key in a data structure that includes the information in the ticket fields *flags, authtime, starttime, endtime,* and *renew-till.* The entire structure is encrypted in the client's key and returned with KRB_AS_REP or KRB_TGS_REP.

How the KDC Limits a Ticket's Lifetime

Tickets have a start time and an expiration time. At any time after the start time but before the expiration time, a client holding a ticket for a service can present the ticket and gain access to the service, no matter how many times the client has used the ticket before. In order to reduce the risk that a ticket or the corresponding session key may be compromised, administrators can set the maximum lifetime for tickets. This time is an element of Kerberos policy.

When a client asks the KDC for a ticket to a service, it may request a specific start time. If this time is missing from the request or is a time in the past, the KDC sets the ticket's *starttime* field to the current time.

Whether or not clients specify a start time, their requests must include a desired expiration time. The KDC determines the value of a ticket's *endtime* field by adding the maximum ticket life fixed by Kerberos policy to the value of the ticket's *starttime* field. It then compares the result with the requested expiration time. Whichever time is sooner becomes the ticket's *endtime*.

What Happens When Tickets Expire

The KDC does not notify clients when session tickets or TGTs are about to expire. In fact, it makes no effort to keep track of transactions with clients beyond short-term records needed to prevent replay attacks.

If a client presents an expired session ticket when requesting a connection to a server, the server returns an error message. The client must request a new session ticket from the KDC. Once a connection is authenticated, however, it no longer matters whether the session ticket remains valid. Session tickets are used only to authenticate new connections with servers. Ongoing operations are not interrupted if the session ticket used to authenticate the connection expires during the connection.

If a client presents an outdated TGT when requesting a session ticket from the KDC, the KDC responds with an error message. The client must request a new TGT, and to do that it needs the user's long-term key. If the client did not cache the user's long-term key during the initial logon process, the client may have to ask the user for a password and derive the long-term key.

Renewable TGTs

One defense against attacks on session keys is to force them to change often by setting Kerberos policy so that maximum ticket life is relatively short. Another is to allow renewable tickets. When tickets are renewable, session keys are refreshed periodically without issuing a completely new ticket. If Kerberos policy permits renewable tickets, the KDC sets a RENEWABLE flag in every ticket it issues and sets two expiration times in the ticket. One expiration time limits the life of the current instance of the ticket. A second expiration time sets a limit on the cumulative lifetime of all instances of the ticket.

The expiration time for the current instance of the ticket is held in the *endtime* field. As with non-renewable tickets, *endtime* is the value of the *starttime* field plus the maximum ticket life specified by Kerberos policy. A client holding a renewable ticket must send it to the KDC for renewal before the *endtime* is reached, presenting a fresh authenticator as well. When the KDC receives a ticket for renewal, it checks a second expiration time held in the *renew-till* field. This time is set when the ticket is first issued, and the value is the ticket's *starttime* plus the maximum cumulative ticket life specified by Kerberos policy. When the KDC renews the ticket, it checks to see that the *renew-till* time has not yet arrived. If it has not, the KDC issues a new instance of the ticket with a later *endtime* and a new session key.

This means that administrators can set Kerberos policy so that tickets must be renewed at relatively short intervals—every day, perhaps. When tickets are renewed, a new session key is issued, minimizing the value of a compromised key. Administrators can also set cumulative ticket life for a relatively long period—one week, one month, whatever. At the end of that time, the ticket expires and is no longer valid for renewal.

Delegation of Authentication

Multitier client/server applications present a special situation for the Kerberos protocol. In this kind of application, a client may connect to a server that must itself connect to a second server on the back end. For this to happen, the first server must have a ticket to the second. Ideally, the ticket should limit the first server's access on the second server to whatever the client, rather than the server, is authorized to do.

The protocol deals with this situation through a mechanism known as delegation of authentication. Essentially, the client delegates authentication to a server by telling the KDC that the server is authorized to represent the client. The concept is similar to the concept of impersonation in Windows 2000.

Delegation can be done two ways. First, the client can get a ticket for the back-end server and then give it to the front-end server. Tickets obtained in this way—by a client for a proxy—are called proxy tickets. The difficulty with proxy tickets is that the client must know the name of the back-end server. This difficulty is overcome by a second method of delegation that allows the client to give the front-end server a TGT it can use to request tickets as needed. Tickets obtained in this way—with credentials forwarded by a client—are called forwarded tickets. Whether the KDC will allow clients to obtain proxy tickets or forwardable TGTs is a matter for Kerberos policy.

Proxy Tickets

When the KDC issues a TGT to a client, it checks Kerberos policy to see if proxy tickets are allowed. If they are, the KDC sets the PROXIABLE flag in the TGT it issues to the client.

The client obtains a proxy ticket by presenting a TGT to the ticket-granting service and asking for a ticket to the back-end server. The client's request includes a flag signaling that it wants a proxy ticket and also includes the name of the server who will represent the client. When the KDC receives the client's request, it creates a ticket for the back-end server, sets the PROXY flag in the ticket, and sends it back to the client. The client then sends the ticket to the front-end server, which uses the ticket to access the back-end server.

Forwarded Tickets

If a client wants to delegate the task of obtaining tickets for back-end servers to a front-end server, it must ask the KDC for a forwardable TGT. It does this through an AS Exchange, indicating to the KDC the name of the server that will act on the client's behalf. If Kerberos policy permits forwarding, the KDC creates a TGT for the front-end server to use in the client's name, sets the FORWARDABLE flag, and sends it back to the client. The client then forwards the TGT to the front-end server.

When the front-end server requests a ticket to the back-end server, it presents the client's TGT to the KDC. When the KDC issues a ticket, it sees the FORWARDABLE flag in the TGT, sets the FORWARDED flag in the ticket, and returns the ticket to the front-end server.

Top of page

Kerberos Components in Windows 2000

Key Distribution Center

Windows 2000 implements the Key Distribution Center (KDC) as a domain service. It uses the domain's Active Directory as its account database and gets some information about users from the Global Catalog.

As in other implementations of the Kerberos protocol, the KDC is a single process that provides two services:

- **Authentication Service (AS).** This service issues Ticket Granting Tickets (TGTs) good for admission to the

ticket-granting service in its domain. Before network clients can get tickets for services, they must get an initial TGT from the authentication service in the user's account domain.

- **Ticket-Granting Service (TGS)**. This service issues tickets good for admission to other services in its domain or to the ticket-granting service of a trusted domain. When clients want access to a service, they must contact the ticket-granting service in the service's account domain, present a TGT, and ask for a ticket. If the client does not have a TGT good for admission to that ticket-granting service, it must get one through a referral process that begins at the ticket-granting service in the user's account domain and ends at the ticket-granting service in the service's account domain.

The KDC is located on every domain controller, as is the Active Directory service. Both services are started automatically by the domain controller's Local Security Authority (LSA) and run in the process space of the LSA. Neither service can be stopped. Windows 2000 ensures availability of these services by allowing each domain to have several domain controllers, all peers. Any domain controller can accept authentication requests and ticket-granting requests addressed to the domain's KDC.

The security principal name used by the KDC for a Windows 2000 domain is *krbtgt*, as specified by RFC 1510. An account for this security principal is created automatically when a new domain is created. The account cannot be deleted, nor can the name be changed. A password is assigned to the account automatically and is changed on a regular schedule, as are the passwords assigned to domain trust accounts. The password for the KDC's account is used to derive a secret key for encrypting and decrypting the TGTs that it issues. The password for a domain trust account is used to derive an inter-realm key for encrypting referral tickets.

All instances of the KDC within a domain use the domain account for the security principal *krbtgt*. Clients address messages to a domain's KDC by including both the service principal name, *krbtgt*, and the name of the domain. Both items of information are also used in tickets to identify the issuing authority. For information on name forms and addressing conventions, see RFC 1510.

Account Database

The account database that the KDC needs in order to obtain information about security principals is provided by the domain's Active Directory. Each principal is represented by an account object. The encryption key used in communicating with a user, computer, or service is stored as an attribute of that security principal's account object.

Only domain controllers are Active Directory servers. Each domain controller keeps a writeable copy of the directory, so accounts can be created, passwords reset, and group membership modified at any domain controller. Changes made to one replica of the directory are automatically propagated to all other replicas. Windows 2000 does not, however, implement the Kerberos replication protocol. Instead, it replicates the information store for Active Directory using a proprietary multi-master replication protocol over a secure channel between replication partners.

Physical storage of account data is managed by the Directory System Agent (DSA), a protected process integrated with the LSA on the domain controller. Clients of the directory service are never given direct access to the data store. Any client wanting access to directory information must use one of the supported Active Directory Service Interfaces (ADSI) to connect to the DSA and then search for, read, and write directory objects and their attributes.

Requests to access an object or attribute in the directory are subject to validation by Windows 2000 access control mechanisms. Like file and folder objects in the Windows NT File System (NTFS), objects in Active Directory are protected by Access Control Lists (ACLs) that specify who can access the object and in what way. Unlike files and folders, however, Active Directory objects have an ACL for each of their attributes. Thus attributes for sensitive account information can be protected by more restrictive permissions than those granted for other attributes of the account.

The most sensitive information about an account is of course its password. Although an account object's password attribute stores an encryption key derived from a password, not the password itself, this key is just as useful to an intruder. Therefore, access to an account object's password attribute is granted only to the account holder, never to anyone else, not even administrators. Only processes with Trusted Computer Base privilege—processes running in the security context of the LSA—are allowed to read or change password information.

In order to hinder an off-line attack by someone with access to a domain controller's backup tape, an account object's password attribute is further protected by a second encryption using a *system key*. This encryption key may be stored on removable media so that it can be safeguarded separately, or stored on the domain controller but protected by a dispersal mechanism. Administrators are given the option to choose where the system key is stored and which of several algorithms is used to encrypt password attributes.

Kerberos Policy

In Windows 2000, Kerberos policy is defined at the domain level and implemented by the domain's KDC. Kerberos policy is stored in Active Directory as a subset of the attributes of domain security policy. By default, policy options

can be set only by members of the Domain Administrators group.

Kerberos policy includes these settings:

Note: The defaults given here are not the defaults used in Beta 3.

- **Maximum user ticket lifetime**. A "user ticket" is a TGT. Settings are in hours. The default is ten hours.
- **Maximum lifetime that a user ticket can be renewed.** Settings are in days. The default is seven days.
- **Maximum service ticket lifetime.** A "service ticket" is a session ticket. Settings are in minutes. The setting must be greater than ten minutes and less than the setting for *Maximum user ticket lifetime*. The default is ten hours.
- **Maximum tolerance for synchronization of computer clocks.** Settings are in minutes. The default is five minutes.
- **Enforce user logon restrictions.** When this option is enabled, the KDC validates every request for a session ticket by examining user rights policy on the target computer to verify that the user has the right either to *Log on locally* or to *Access this computer from network*. Verification is optional because the extra step takes time and may slow network access to services. The default is enabled.

Delegation of Authentication

Although Kerberos policy for a domain may permit delegated authentication by allowing forwardable tickets, that aspect of policy need not apply to all users or all computers. An attribute of an individual user account can be set to disable forwarding of that user's credentials by any server. An attribute of an individual computer's account can be set to disable forwarding of credentials from any user. In both cases, delegation can be disabled by creating a policy to apply to all users or all computers in an organizational unit within the domain.

Pre-authentication

By default the KDC requires all accounts to use pre-authentication. This makes offline password-guessing attacks very difficult. However, pre-authentication can be disabled for individual accounts when necessary for compatibility with other implementations of the protocol.

Kerberos Security Support Provider

The Kerberos authentication protocol is implemented as a *security support provider* (SSP), a dynamic-link library supplied with the operating system. Windows 2000 includes an SSP for NTLM authentication as well. By default, both are loaded by the LSA on a Windows 2000 computer when the system boots. Either SSP may be used to authenticate network logons and client/server connections. Which one is used depends on the capabilities of the computer on the other side of the connection. The Kerberos SSP is always the first choice.

After the LSA establishes a security context for an interactive user, another instance of the Kerberos SSP may be loaded by a process running in the user's security context to support the signing and sealing of messages.

System services and transport-level applications access SSPs through the Microsoft Security Support Provider Interface (SSPI). This is a Win32® interface with methods for enumerating the providers available on a system, selecting one, and using it to obtain an authenticated connection. The methods in the SSPI are generic, black-box routines that developers can use without knowing the details of a particular protocol. For example, when a client/server connection is authenticated, the application on the client's side of the connection sends credentials to the server using the SSPI method InitializeSecurityContext. If the Kerberos SSP has been selected, this method generates a KRB_AP_REQ message from the client. The application on the server's side of the connection responds with the SSPI method AcceptSecurityContext, which generates a KRB_AP_REP message from the server. Once the connection has been authenticated, the LSA on the server uses information from the client's ticket to build an access token. The server then invokes the SSPI method ImpersonateSecurityContext to attach the access token to an impersonation thread for the service.

All distributed services in Windows 2000 use the SSPI to access the Kerberos SSP. A partial list of the ways in which the Kerberos protocol is used for authentication includes:

- Print spooler services
- CIFS/SMB remote file access

- LDAP queries to Active Directory
- Distributed file system management and referrals
- IPSec host-to-host security authority authentication
- Reservation requests for network Quality of Service
- Intranet authentication to Internet Information Server
- Remote server or workstation management using authenticated RPC
- Certificate requests to the Microsoft Certificate Server for domain users and computers

Credentials Cache

On computers running Windows 2000, tickets and keys obtained from the KDC are stored in a credentials cache, an area of volatile memory protected by the LSA. The credentials cache is never paged to disk. All objects stored there are destroyed when a security principal logs off or the system is shut down.

The credentials cache is managed by the Kerberos SSP, which runs in the LSA's security context. Whenever tickets and keys need to be obtained or renewed, the LSA calls the Kerberos SSP to accomplish the task.

The LSA also keeps a copy of an interactive user's hashed password. If the user's TGT expires during a logon session, the Kerberos SSP uses the LSA's copy of the hashed password to obtain a new TGT silently, without interrupting the user's logon session. The password is not stored permanently on the computer, and the local copy is destroyed when the user's logon session is destroyed.

Hashed passwords for services and computers are handled differently. As in earlier versions of Windows NT, they are stored in a secure area of the computer's registry. The registry is also used to store hashed passwords for user accounts on the local system, but local accounts are used only for access to computers in standalone mode, never for network access.

DNS Name Resolution

RFC 1510 specifies that IP transport should be used for messages between clients and the KDC. When the Kerberos SSP on a client computer wants to send an initial authentication service request, it needs to find an address for the KDC in the user's account domain. To do that, it needs the Domain Name System (DNS) name of a server where the KDC service is running. If the DNS name can be resolved to an IP address, that is where the Kerberos SSP sends its message. Otherwise, the Kerberos SSP generates an error indicating that it can find no such domain.

In Windows 2000 domains, the KDC service runs on every Windows 2000-based domain controller. In addition to being KDC servers, domain controllers are also Lightweight Directory Access Protocol (LDAP) servers. Both services are registered in DNS service locator records (SRV resource records). Clients can find a domain controller by querying DNS for SRV resource records with the name _ldap._tcp.dc._msdcs.*DnsDomainName*. The KDC service can be located by querying DNS for SRV resource records with the name _kerberos._udp.*DnsDomainName*. Clients that do not support the SRV record type in their DNS resolver can query for a host record (an A resource record) with the name of the domain.

Computers running Windows 2000 can participate in Kerberos realms that are not Windows 2000 domains. In this case, the KDC will not be on a Windows 2000 domain controller, so the DNS names for KDC servers must be stored in the client computer's registry. The Kerberos SSP looks in the registry for the DNS domain name of the user's realm, and then resolves the name to an IP address by querying a DNS server.

Windows 2000 includes a tool for configuring clients to participate in Kerberos realms that are not Windows 2000 domains. Look for ksetup.exe in the Support folder of the Windows 2000 installation CD.

IP Transport

According to RFC 1510, when a client contacts the KDC, it should send a User Datagram Protocol (UDP) datagram to port 88 at the KDC's IP address. The KDC should respond with a reply datagram to the sending port at the sender's IP address.

UDP is a connectionless transport protocol, making it a logical choice where an exchange of messages must precede a connection. UDP is also well suited to applications with one message and one response, like the exchanges between clients and the KDC, so long as each message fits into a single datagram. However, UDP works best when

datagrams are transmitted as single units—when each datagram fits into one frame. The capacity of a frame varies with the medium. The Maximum Transmission Unit (MTU) for an Ethernet frame is 1500 octets. If the physical network is Ethernet, then Kerberos messages sent as UDP datagrams can carry up to 1500 octets of data.

Windows 2000 authorization data can easily total more than 1500 octets. Because this data is needed only by computers with the Windows 2000 operating system, it is omitted from tickets to computers with other operating systems. As a result, those messages are well within the limits of UDP transport, and so that is how they are transmitted. Messages with tickets for computers running Windows 2000 are likely to exceed the limit, and so they are transmitted using the Transport Control Protocol (TCP), which has much greater capacity. The use of TCP transport is consistent with recently proposed revisions to RFC 1510.[5]

Top of page

Authorization Data

Kerberos is a protocol for authentication, not authorization. It verifies that security principals are who they say they are. It does not determine which files and other objects security principals may access, or how they may access them. These decisions are left to whatever access control mechanism may be available on the system. The protocol assists by providing a field for authorization data in Kerberos tickets, but it does not specify the form of the data or how servers should use it.

Access Control in Windows 2000

On some operating systems, applications are required to implement their own mechanisms for determining the level of a user's authorization. Often applications do this by maintaining private lists with the names of users who are authorized access. This kind of access control can be integrated with Kerberos authentication simply by ensuring that the authorization data field of a ticket carries some form of the security principal's name. This is sometimes called *name-based authorization*.

Applications designed for Windows 2000 can implement name-based authorization mechanisms to control access to information internal to the application. Database applications, for example, often maintain private authorization tables to control which fields in a record a particular user can view or change. Unfortunately, private authorization mechanisms are just that—*private*. A database application is powerless to prevent an unauthorized user from running another application and using it to edit the data file.

On Windows 2000, if a file or other resource can be shared by two processes, it is secured against unauthorized access by the operating system's own access control mechanism. The header of every object includes a security descriptor with an access control list (ACL) maintained by the object's owner, who has the discretion to grant or deny access to any security principal, and to define the level of authorization for a security principal who has been granted access. The operating system enforces the owner's decisions by performing an access check whenever an application requests a handle to a protected object. Before returning a handle, the operating system examines the object's ACL to see whether the security principal who is using the application is authorized access. If not, the application is denied access.

Another important difference from other access control mechanisms is that security principals are not identified by name in Windows 2000, either by the operating system or by entries in ACLs. Instead, each security principal is assigned a unique Security Identifier (SID), an alphanumeric value with a structure similar to a telephone number. Like the country code used in international calling, the first part of a SID identifies the domain where the SID was issued. Like the number for a particular telephone within a country, the second part of a SID identifies an account within the issuing domain. The value for a domain is unique within an enterprise, and the value for an account is unique within a domain. Unlike telephone numbers, SIDs are never reused. There is no possibility that a user could be assigned a SID that once belonged to another user—a problem not easily solved by name-based access control mechanisms.

A third important difference with name-based access control is that authorization is determined not only by user's identity but also by the user's membership in one or more security groups. In fact, the preferred method of controlling resources is to grant access to groups rather than to individuals, adjusting the level of a group's authorization according to the needs of its members. This method makes it easier to keep ACLs up-to-date on networks with thousands of users and millions of objects. Group membership can be managed centrally by administrators, who can change a particular user's level of authorization for many resources simply by adding or removing a member from a group.

Windows 2000 makes resource security still easier to manage by allowing groups to be nested. A group created in one domain can be included in the membership of a group created in another domain, or in the membership of a universal group used throughout a tree of trusted domains. As a result, when employees change jobs, their level of authorization can be changed throughout the enterprise without touching the ACL on a single object.

Like individual security principals, each Windows 2000 security group has a SID. A user's level of authorization is determined, then, by a list of SIDs—one SID for the user and one SID for each security group to which the user

belongs.

How the KDC Prepares Authorization Data

When the Kerberos protocol is used for authentication, a list of SIDs identifying a security principal and the principal's group membership is transported to the local computer in the authorization data field of a session ticket. Authorization data is gathered in two separate steps. The first step takes place when the KDC in a Windows 2000 domain prepares a TGT. The second step is accomplished when the KDC prepares a session ticket for a server in the domain.

When a user requests a TGT, the KDC in the user's account domain queries the domain's Active Directory. The user's account record includes an attribute for the user's SID as well an attribute with SIDs for any domain security groups to which the user belongs. The list of SIDs returned by the KDC's query is placed in the TGT's authorization data field. In a multiple-domain environment, the KDC also queries the Global Catalog for any universal groups that include the user or one of the user's domain security groups. If any are found, their SIDs are added to the list in the TGT's authorization data field.

When the user requests a session ticket for a server, the KDC in the server's domain copies the contents of the TGT's authorization data field to the session ticket's authorization data field. If the server's domain is different from the user's account domain, the KDC queries Active Directory in order to find out whether any security groups in the local domain include the user or one of the user's security groups. If any are found, their SIDs are added to the list in the session ticket's authorization data field.

This use of the authorization data field is consistent with revisions to RFC 1510 submitted to the IETF.[6] The precise format of Windows 2000 authorization data is subject to change until the operating system's release. The Windows 2000 authorization data is in Network Data Representation (NDR) format and is signed by the KDC.

How Services Use Authorization Data

In Windows 2000, services act in their own security contexts only when accessing resources on their own behalf. For the most part, this is just when they are doing their own housekeeping—accessing configuration data stored in registry keys, binding to communications ports, and completing other tasks not related to work for a particular client. When a service does do something for a client, it *impersonates* the client, acting in the client's security context. This means that in addition to identifying clients, Windows 2000 services must also take on some of their characteristics—specifically the client's level of authorization on the system.

When a service sets up housekeeping on a computer running Windows 2000, it calls the SSPI method AcquireCredentialsHandle to gain access to its own credentials—the secret key for the account under which the service runs. The service then binds to a communications port, where it listens for messages from prospective clients.

When a client requests a connection and presents a session ticket, the service asks the Kerberos SSP to verify the client's credentials by calling the SSPI method AcceptSecurityContext, passing the client's session ticket along with a handle to the service's secret key. The Kerberos SSP verifies the ticket's authenticity, opens it, and passes the contents of the authorization data field to its parent process, the LSA. If the data includes a list of SIDs, the LSA uses them to build an access token representing the user on the local system. In addition, the LSA queries its own database to determine if the user or one of the user's security groups is a member of a security group created on the local system. If any are found, the LSA adds those SIDs to the access token. The LSA then confirms to the calling service that the client's identity has been authenticated, enclosing a reference to the client's access token.

On receiving confirmation, the service completes its connection with the client and attaches the client's access token to an impersonation thread by calling ImpersonateSecurityContext. When the impersonation thread needs access to an object, it presents the client's token. The operating system performs an access check by comparing SIDs in the token to SIDs in the object's ACL. If it finds a match, it checks to see that the entry in the ACL grants the level of access requested by the thread. If it does, the thread is allowed access. Otherwise, access is denied.

Why Authorization Data Is Signed

Session tickets are encrypted with the secret key for the account under which the service starts. When a service acquires a handle to its own credentials on the system, it gains access to that key. A rogue service could be installed by an unscrupulous user with a legitimate network account but limited authorization on the local computer. The user could request a session ticket for the service, and it could decrypt the ticket, modify the authorization data by adding the SID for a privileged group, encrypt the altered ticket, and present it to the LSA in a call to AcceptSecurityContext. The result would be to elevate the user's level of authorization on the computer where the service is running.

In order to prevent tampering with authorization data, it is signed by the KDC before it is stored in a session ticket. Any attempt to alter the data will invalidate the signature. The LSA on a Windows 2000 computer always checks the

validity of this signature when session tickets are presented by untrusted services.

The LSA can trust calls to AcceptSecurityContext from services running under the Local System account. This account is used by services installed with the operating system—by the native Server service, for example. Other services can be configured to use the Local System account, but this must be done by a member of the Administrators group. It is assumed that the administrator who installs the service can vouch for its security.

The LSA does not trust services running under any other account. If a session ticket is presented by an application that is not running as Local System, the LSA asks the KDC in its domain to verify the signature on the ticket's authorization data. The question is asked and answered by a Remote Procedure Call (RPC) over Netlogon's secure channel to the domain controller. Requests for verification do not need to travel beyond the local domain because session tickets are always issued—and therefore authorization data is always signed—by the KDC for the target computer's domain.

Top of page

Interactive Logon

Users have a natural tendency to think that logging on to an account in a Windows domain gives them access to the network. That is of course not true. When the Kerberos protocol is used for network authentication, what you actually get when you first log on is access to the domain's authentication service. Specifically, you get a TGT that you can present when requesting session tickets for other services in the domain.

When you log on to a domain account from a computer running Windows 2000, you always need at least one session ticket—a ticket for the computer where you are logging on. Unlike other computers, computers running Windows 2000 have their own accounts in the domain. You can think of them as service accounts. Remote users can access resources on a computer that is running Windows 2000 by submitting requests to its Server service. Interactive users can access resources in the domain by submitting requests to the computer's Workstation service. Before you can gain admission to either of these services, or to any other service running as Local System, you must present a session ticket for the computer.

The Logon Process

When a user with an account in a Windows 2000 domain logs on at the keyboard of a computer running Windows 2000, the user's logon request is processed in three stages:

1. The user asks for admission to the ticket-granting service for the domain.

   This is accomplished by an AS Exchange between the Kerberos SSP on the computer and the KDC for the user's account domain. The result of a successful exchange is a TGT that the user can present in future transactions with the KDC.

2. The user asks for a ticket for the computer.

   This is accomplished by a TGS Exchange between the Kerberos SSP on the computer and the KDC for the computer's account domain. The result is a session ticket that the user can present when requesting access to system services on the computer.

3. The user asks for admission to Local System services on the computer.

   This is accomplished when the Kerberos SSP on the computer presents a session ticket to the LSA on the computer.

If the computer's domain is different from the user's domain, an extra step is involved. Before requesting a session ticket for the computer, the Kerberos SSP must first ask the KDC in the user's account domain for a TGT good for admission to the KDC in the computer's account domain. It can then present the TGT to that KDC and get a session ticket for the computer.

Exactly how the logon process works will depend on how the computer is configured. With standard configurations of Windows 2000, interactive users log on with a password. The operating system also includes the option to log on with a smart card. The basic process is the same for both configurations. We will examine their differences after first stepping through the process for a standard logon with a password.

Password Logon

Suppose Alice has a network account in the domain West. The computer she normally uses, Bob, also has an

account in West. When Alice logs on to the network, she begins by pressing the key combination CTRL+ALT+DEL, the Secure Attention Sequence (SAS) on computers with a standard Windows 2000 configuration.

In response to the SAS, the computer's Winlogon service switches to the logon desktop and dispatches to the Graphical Identification and Authentication (GINA) dynamic-link library, a component loaded in Winlogon's process. The GINA is responsible for collecting logon data from the user, packaging it in a data structure, and sending everything to the LSA for verification. Third parties can develop replacement GINAs, but in this case Winlogon has loaded the standard component supplied with the operating system, MSGINA.DLL. Winlogon dispatches to it, and it displays the standard Logon Information Dialog.

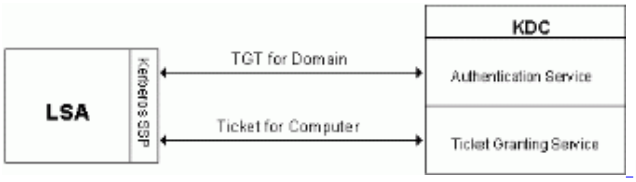
Alice types her user name and password. From the domains in a drop-down list, she selects West. When she clicks OK to dismiss the dialog, MSGINA returns her logon information to Winlogon. It sends the information to the LSA for validation by calling LsaLogonUser.

On receiving a data structure with Alice's logon data, the LSA immediately converts her clear text password to a secret key by passing it through a one-way hashing function. It saves the result in the credentials cache, where it can be retrieved when needed for TGT renewal or for NTLM authentication to servers that are not capable of Kerberos authentication..

In order to validate Alice's logon information and set up her logon session on the computer, the LSA must obtain:

- A TGT good for admission to the ticket-granting service.
- A session ticket good for admission to the computer.

The LSA gets these tickets by working through the Kerberos SSP, which exchanges messages directly with the KDC in West.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs08_big(en-us,TechNet.10).gif]

Figure 8: Interactive logon to a domain account

The messages follow this sequence:

1. The LSA sends the message KRB_AS_REQ to the KDC in West.

   The message includes:

   - The user principal name, Alice.
   - The name of the account domain, West.
   - Pre-authentication data encrypted with the secret key derived from Alice's password.

2. If the client's pre-authentication data is correct, the KDC replies with **KRB_AS_REP**.

   The message includes:

   - A session key for Alice to share with the KDC, encrypted with the secret key derived from Alice's password.
   - A TGT for the KDC in West, encrypted with the KDC's secret key.

   The TGT includes:

   - A session key for the KDC to share with Alice.
   - Authorization data for Alice.

   The authorization data includes:

- The SID for Alice's account.
- SIDs for security groups in the domain West that include Alice.
- SIDs for universal groups in the enterprise that include either Alice or one of her domain groups.

3. The LSA sends the message **KRB_TGS_REQ** to the KDC in West.

   The message includes:

   - The name of the target computer, Bob.
   - The name of the target computer's domain, West.
   - Alice's TGT.
   - An authenticator encrypted with the session key Alice shares with the KDC.

4. The KDC replies with **KRB_TGS_REP**.

   The message includes:

   - A session key for Alice to share with Bob, encrypted with the session key Alice shares with the KDC.
   - Alice's session ticket to Bob, encrypted with the secret key Bob shares with the KDC.

   The session ticket includes:

   - A session key for Bob to share with Alice.
   - Authorization data copied from Alice's TGT.

Having received Alice's session ticket, the LSA decrypts it with the computer's secret key and extracts her authorization data. It then queries the local SAM database to discover whether Alice is a member of any security groups local to the computer and whether she has been given any special privileges on the local machine. It adds any SIDs returned by this query to the list taken from the ticket's authorization data. The entire list is then used to build an access token, and a handle to the token is returned to Winlogon, along with an identifier for Alice's logon session and confirmation that her logon information was valid.

Winlogon creates a window station and several desktop objects for Alice, attaches her access token, and starts the shell process she will use to interact with the computer. Alice's access token is subsequently inherited by any application process that she starts during her logon session.

Smart Card Logon

In standard Kerberos logons, users initially prove to the KDC that they are who they say they are by showing that they know a secret known only by the user and the KDC. This shared secret is a cryptographic key derived from the user's password. It is used only during the AS Exchange:

- When the client encrypts pre-authentication data.
- When the KDC decrypts pre-authentication data.
- When the KDC encrypts the logon session key.
- When the client decrypts the logon session key.

The same key is used for both encryption and decryption. For this reason, shared secret keys are said to be *symmetric*.

To support smart card logons, Windows 2000 implements a public key extension to the Kerberos protocol's initial AS Exchange.[7] In contrast to how shared secret keys work, public key cryptography is *asymmetric*. Two different keys are needed, one to encrypt, the other to decrypt. Together, the keys needed to perform both operations make up a

private/public key pair. The private key is known only to the owner of the pair and is never shared. The public key can be made available to anyone with whom the owner wishes to exchange confidential information.

When a smart card is used in place of a password, a private/public key pair stored on the user's smart card is substituted for the shared secret key derived from the user's password. In the public key extension to the Kerberos protocol, the initial AS Exchange is modified so that the KDC encrypts the user's logon session key with the public half of the user's key pair. The client decrypts the logon session key with the private half of the pair.

The logon process begins when the user inserts a smart card into a card reader attached to the computer. When computers with Windows 2000 are configured for smart card logon, a card insertion event signals the SAS, just as the key combination CTRL+ALT+DEL does on computers configured for password logon. In response, Winlogon dispatches to MSGINA, which displays a Logon Information Dialog. In this case, the user types just one item of information, a Personal Identification Number (PIN).

MSGINA sends the user's logon information to the LSA by calling LsaLogonUser, just as it does with a password logon. The LSA uses the PIN to access the smart card, which stores the user's private key and an X.509 v3 certificate containing the public half of the key pair. All cryptographic operations that use these keys take place on the smart card.

The Kerberos SSP on the client computer sends the user's public key certificate to the KDC as pre-authentication data in its initial authentication request, KRB_AS_REQ. The KDC validates the certificate, extracts the public key, and uses it to encrypt a logon session key. It returns the encrypted logon session key, along with a TGT, in its reply to the client, KRB_AS_REP. If the client is in possession of the private half of the key pair, it will be able to use the private key to decrypt the logon session key. Both the client and the KDC then use the logon session key in all further communications with one another. No other deviation from the standard protocol is necessary.

For information on the types of smart cards and readers that will be supported by Windows 2000, see the Windows 2000 Hardware Compatibility List.

Pre-Authentication Data

By default, the Kerberos SSP on a client computer sends the KDC pre-authentication data in the form of an encrypted timestamp. On systems configured for smart card logon, it sends pre-authentication data in the form of a public key certificate.

Top of page

Remote Logon

When you log on at a computer's keyboard, the logon process creates a security context for your actions on the local computer. Your security identity is encapsulated in an access token attached to the process object that represents your logon session. When you start an application, it runs within your logon session's process, and the application process inherits your access token. The application acts in your security context; it can access only resources that you are authorized to access. Something similar happens when an application that you are using becomes the client of a server application on the same computer. The server process spawns a thread of execution, attaches a copy of your access token to the thread, and the thread impersonates you while working on your behalf. The thread acts in your security context and can do only what you are allowed to do on the local computer.

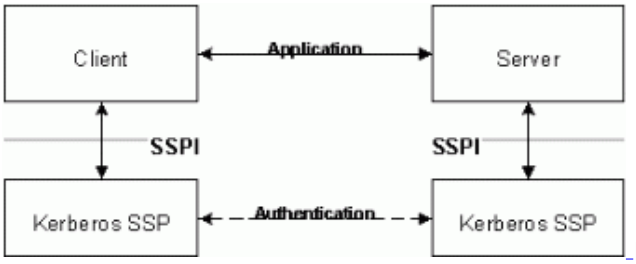
When a client application on your computer connects to a server application on another computer, there is no access token to attach to the server's impersonation thread. You have no security context on the remote computer, and won't have one until you are logged on to that computer. However, unlike logging on to a local computer, logging on to a remote computer is not interactive. You do not need to supply logon information by entering it in a dialog box. Instead, the client application on the local computer sends your logon credentials to the server application on the remote computer, and the server application establishes a security context for your actions on the remote computer.

Security Support Provider Interface

Typically, authentication is built into whatever mechanism the client/server application uses for interprocess communication. Client/server applications designed for Windows 2000 do not need to know the details of a particular authentication protocol, or even which protocols are supported. Instead, both sides of the application can make standard calls to the Security Support Provider Interface (SSPI). The client side calls the SSPI to request a security context for the user. The server side calls the SSPI to create a security context for the user. The SSPI takes care of the details.

In a sense, the SSPI allows the authentication protocol to take place as a subtext to the application protocol, as a kind of conversation within a conversation. On both sides of the client/server conversation, the SSPI works through a security support provider (SSP) that has code needed to implement a particular authentication protocol. The SSP simply hands authentication messages to the application for transport. As far as the application is concerned, these

messages are opaque data. It has no responsibility for knowing what is inside an authentication message that it transports or for knowing how to respond to an authentication message that it receives. All the work of interpreting and responding to authentication messages is done by the SSP.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs09_big(en-us,TechNet.10).gif]

Figure 9: Authentication protocol as subtext to application protocol

Security Support Provider Negotiation

Windows 2000 includes SSPs designed to handle Kerberos authentication, NTLM, and SChannel. Applications can specify a particular SSP when calling the SSPI, or they can use the SSPI's negotiation feature to select the most secure protocol from those available on the computer.[8] When the application requests logon with negotiation, the client proposes an ordered list of authentication protocols, starting with the most secure. The server selects from the list, choosing the most secure protocol available on its side. On systems running Windows 2000, the preference is Kerberos first, then NTLM, then SChannel. If both client and server are running on Windows 2000, the negotiation mechanism will select the Kerberos SSP.

Example: Opening a File on a Remote Server

Suppose that Alice has logged on to her workstation using her domain account. In the course of her work, Alice starts Microsoft Word and opens a document stored on a file server.

To open the document, Microsoft Word submits an I/O request through an API call to the Windows 2000 operating system on the workstation. The operating system determines that the document is a remote resource and passes the request to the file system redirector. The redirector uses the Server Message Block (SMB) file sharing protocol to connect to the Server service on the remote file server, open the file, and read its data. The entire operation is carried out in stages through a series of SMB messages. The first step—and the only one of interest here—is for the redirector to obtain an authenticated connection to the remote service.

Negotiating a Security Support Provider

Before either side of the SMB conversation can verify the other's identity, the two parties must agree on an authentication protocol. Both client and server list the security support providers available on their systems and indicate preferences. In this case, both are running Windows 2000, so the preference on both sides is the Kerberos SSP. That is what they agree to use.

How the Client Initiates Authentication

The authentication protocol begins when the redirector on Alice's workstation calls the SSPI method AcquireCredentialsHandle, specifying Alice as the security principal. This call returns a handle to the credentials Alice obtained when she logged on interactively to her workstation. The redirector then calls the SSPI method InitializeSecurityContext, passing a handle to Alice's TGT and specifying the file server as the target. This call generates the Kerberos message KRB_TGS_REQ, which includes Alice's TGT. The Kerberos SSP sends the message directly to the KDC in Alice's account domain. The KDC returns a ticket for the file server. The Kerberos SSP caches the ticket and returns to the calling process, the redirector, indicating that authentication is not yet complete.

The redirector calls InitializeSecurityContext a second time, asking for a continuation of the previous call. This time the Kerberos SSP generates Kerberos message KRB_AP_REQ, which includes a ticket encrypted in the secret key that the server shares with the KDC, and an authenticator encrypted in the session key that Alice shares with the server. The Kerberos SSP returns this message to the redirector. The redirector includes the Kerberos message as an authentication token in an SMB message that it sends to the Server service.

How the Server Responds

When the Server service receives the redirector's SMB message, it creates a local security context for the client by calling the SSPI method AcceptSecurityContext, passing a pointer to the authentication token. The Kerberos SSP on the file server opens KRB_AP_REQ, extracts the ticket, takes out the session key, and uses it to validate Alice's authenticator. If the authenticator is valid, the Kerberos SSP removes Alice's authorization data from the session

ticket, passing it to the LSA on the server. The LSA builds an access token for Alice, establishing a security context for her actions on the system. That done, the Kerberos SSP prepares KRB_AP_REP and returns it, along with a handle to Alice's security context, to the calling process.

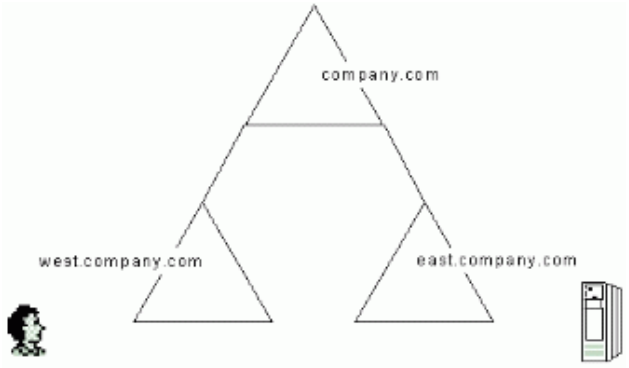
When the call to AcceptSecurityContext returns, the Server service sends KRB_AP_REP as data in an SMB message to the redirector on Alice's workstation. It then uses the handle to Alice's security context to impersonate Alice by calling the SSPI method ImpersonateSecurityContext. As a result of this call, Alice's access token is attached to an impersonation thread in the service process, allowing it to act on Alice's behalf when opening the file. If Alice has been given permission to read the file, the service will be able to respond to the redirector's I/O request.

Mutual Authentication

When the redirector receives the SMB message containing KRB_AP_REP, it hands the data to the Kerberos SSP on Alice's workstation so that it can validate the file server's identity. The Kerberos SSP uses its copy of the session key to decrypt the server's authenticator. It compares the timestamp in the server's authenticator with the timestamp in the authenticator that it sent with KRB_AP_REQ. If the times do not match, the Kerberos SSP returns an error and the redirector breaks the connection with the file server. Otherwise, the connection proceeds. This process is mutual authentication.

Example: Cross-Domain Authentication

The network where Alice works has three domains, a parent, and two children, East and West. Alice has an account in West. She has logged on to her domain account and has collected a TGT for the KDC in West. In the course of her work on the computer, Alice decides that she needs to take a look at a document stored in a public share on Bob, a server in the domain East.



[http://technet.microsoft.com/en-us/library/Bb742431.kerbrs10_big(en-us,TechNet.10).gif]

Figure 10: Alice 's network

The procedure for setting up a secure client/server connection is similar to the one described in the preceding example. However, in this case Alice's domain is different from the file server's domain, so the Kerberos SSP on Alice's workstation must obtain a TGT for the server's domain and then use the TGT to get a ticket for the server. This involves a referral process that proceeds as follows:

1. Alice's workstation sends KRB_TGS_REQ to the KDC in **west.company.com**.

   The message includes:

   - The name of the target computer, Bob.
   - The name of the target computer's domain, east.company.com.
   - A TGT for admission to the KDC in west.company.com.
   - An authenticator encrypted with the session key Alice shares with that KDC.

2. The KDC in **west.company.com** replies with KRB_TGS_REP.

   The message includes:

   - A session key for Alice to share with the KDC in company.com, encrypted with Alice's logon session key.

- A TGT for admission to the KDC in company.com, encrypted with the secret key for the trust relationship between the two domains.

3. Alice's workstation sends KRB_TGS_REQ to the KDC in **company.com**.

   The message includes:

   - The name of the target computer, Bob.
   - The name of the target computer's domain, east.company.com.
   - A TGT for admission to the KDC in company.com.
   - An authenticator encrypted with the session key Alice shares with that KDC.

4. The KDC in **company.com** replies with KRB_TGS_REP.

   The message includes:

   - A TGT for admission to the KDC in east.company.com, encrypted with the secret key for the trust relationship between the two domains.
   - A session key for Alice to share with that KDC, encrypted with the session key Alice shares with the KDC in company.com.

5. Alice's workstation sends KRB_TGS_REQ to the KDC in **east.company.com**.

   The message includes:

   - The name of the target computer, Bob.
   - The name of the target computer's domain, east.company.com.
   - A TGT for admission to the KDC in east.company.com.
   - An authenticator encrypted with the session key Alice shares with that KDC.

6. The KDC in **east.company.com** replies with KRB_TGS_REP.

   The message includes:

   - A session key for Alice to share with Bob, encrypted with the session key Alice shares with the KDC in east.company.com.
   - A session ticket for admission to Bob, encrypted with the secret key Bob shares with the KDC.

   The session ticket includes:

   - A session key for Bob to share with Alice.
   - Authorization data copied from Alice's TGT, plus data for the local domain, east.company.com.

   The authorization data includes:

   - The SID for Alice's account.
   - SIDs for groups in west.company.com that include Alice.
   - SIDs for universal groups that include either Alice or one of her groups in west.company.com.
   - SIDs for groups in east.company.com that include Alice, one of her groups in west.company.com, or one of her universal groups.

7. Alice's workstation sends KRB_AP_REQ to **Bob**.

   The message includes:

   - The user principal name, Alice.
   - A ticket to Bob.
   - An authenticator encrypted with the session key Alice shares with Bob.

8. **Bob** replies with KRB_AP_REP.

   The message includes an authenticator encrypted with the session key Bob shares with Alice.

Top of page

Interoperability

The goal for the implementation of the Kerberos version 5 authentication protocol in Windows 2000 is seamless, out-of-the-box interoperability with the reference implementation of Kerberos published by MIT.

Scenarios

Microsoft has tested interoperability with the MIT reference implementation in the following scenarios:

Windows KDC. Non-windows Kerberos implementations can authenticate to the KDC in a Windows 2000 domain. Non-windows Kerberos users and hosts can authenticate to a domain controller by using **kinit** and DES-CBC-MD5 or DES-CBC-CRC encryption.

Non-windows Kerberos KDC. Systems running Windows 2000 can authenticate to a host serving as the KDC of a Kerberos realm. In addition, a standalone Windows 2000 system can be configured so that local computer accounts map to Kerberos principals. This configuration allows users to log on simultaneously to both the computer and the Kerberos realm.

Windows Client, Non-Windows Kerberos Service. Client applications running on Windows 2000 can authenticate to non-Windows Kerberos services if the services support the Generic Security Service Application Program Interface (GSS-API) defined in RFC 1964.

Windows 2000 does not provide the GSS-API. Applications written for the Windows 2000 operating system should use the SSPI to get support for Kerberos version 5 authentication. The two interfaces are compatible and similar.

Non-Windows Kerberos Client, Windows Service. Client applications running on non-Windows Kerberos systems can authenticate to services running on Windows 2000 if the client applications support the GSS-API as defined in RFC 1964.

Trust Relationships. Windows 2000 domains can be configured to trust non-Windows Kerberos realms. Non-Windows Kerberos realms can be configured to trust Windows domains. The trusts are not complete like domain trusts. For instance, user principals from the Kerberos realm do not have the authorization data that Windows 2000-based services need for access control. In order for this authorization data to be added to the user's ticket, an account mapping mechanism is used. Selected domain user accounts are used to provide identity for Kerberos principals in the trusted realm. These mappings are kept on the altSecurityID property on the user account object. They can be managed through Active Directory Users and Computers.

Configuration

Utilities for configuring Windows 2000 for interoperability with UNIX systems are available in the Support folder of the Windows 2000 installation CD. For instructions, see the Beta 3 Technical Walkthrough, "MIT Kerberos 5 (krb5 1.0) Interoperability."

For More Information

For the latest information on Windows 2000, check out Microsoft TechNet or visit our World Wide Web site at http://www.microsoft.com/windows2000/ [http://www.microsoft.com/windows2000/] .

0599

Top of page

1	Miller, S., Neuman, C., Schiller, J., and J. Saltzer, "Section E.2.1: Kerberos Authentication and Authorization System," MIT Project Athena, Cambridge, MA, December 1987.
2	Kohl, J., and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993.
3	Linn, J., "The Kerberos Version 5 GSS-API Mechanism," RFC 1964, June 1996.
4	Tung, B., Neuman, C., Wray, J., Medvinsky, A., Hur, M., and J. Trostle, "Public Key Cryptography for Initial Authentication in Kerberos," draft-ietf-cat-kerberos-pk-init.
5	Neuman, C., Kohl, J. and T. Ts'o, "The Kerberos Network Authentication Service (V5)," draft-ieft-cat-kerberos-revisions-03, November 1998.
6	Neuman, C., Kohl, J, and T. Ts'o, "The Kerberos Network Authentication Service (V5)," draft-ietf-cat-kerberos-revisions, November 1997.
7	Windows 2000 implements the current IETF CAT draft for PKINIT with the exception that the protocol draft requires encryption of an arbitrary element using a 512-bit RSA public key, which is not possible in an exportable product such as Windows NT. In this particular instance, Windows NT uses the PKCS #7 standard for encrypted data. The issue has been discussed with the authors of the draft.
8	The method used to negotiate a security support provider is based on the SPNEGO protocol described in RFC 2478, "Simple and Protected GSS-API Negotiation Mechanism," December 1998.