CHAPTER 12

KERBEROS AND WINDOWS 2000

"That is plain enough," said Gimli. "If you are a friend, speak the password, and the doors will open, and you can enter."

— J. R. R. Tolkien, *The Fellowship of the Ring*

IN THIS CHAPTER

Kerberos provides a mechanism to authenticate and share temporary secret keys between cooperating processes.

- The concept of a key distribution center
- Needham-Schroeder key distribution protocol
- Kerberos tickets and ticket-granting tickets
- Kerberos and Microsoft Windows 2000

12.1 THE KEY DISTRIBUTION CENTER

In the 1980s, the banking industry began to take full advantage of how encryption with the DES could protect electronic funds transfer traffic. As traffic grew, however, the banks struggled with the problem of handling encryption keys safely and effectively, Drawing on network security work by Dennis Branstad, they developed standards for *key distribution centers* (KDCs). Banks used KDCs to generate temporary encryption keys for messages between pairs of bank offices. This eliminated the risk of using a single encryption key for all messages between "trusted" sites. It also eliminated the enormous expense of having to distribute unique keys for every pair of bank offices. The banks' approach was codified in ANSI Standard X9.17. *see Note 1.*

Temporary encryption keys provide a similar benefit to one-time passwords: they limit the damage attackers can do if they manage to

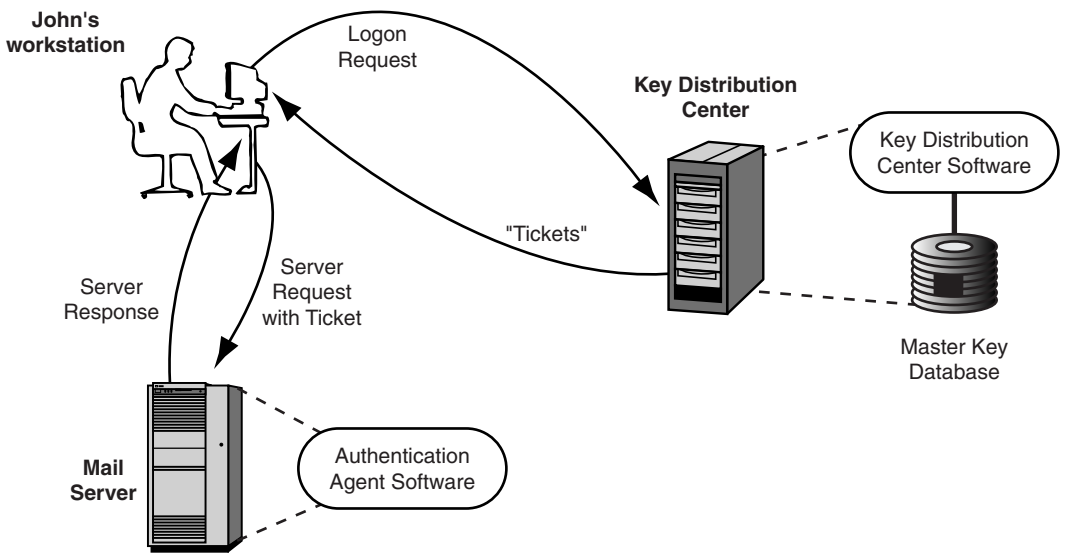

FIGURE 12.1: *Indirect authentication with a key distribution center.* In this approach, two computers mutually authenticate each other by providing a shared secret key that only those two can possibly share. The shared secret key is embedded in a cryptographically protected data item called a "ticket." To use the mail server, John Doe contacts the key distribution center, which gives him two tickets: one encrypted for his own master key and the other encrypted with the mail server's master key. John decrypts his ticket to retrieve the shared secret key and forwards the other to the mail server. The server retrieves its own copy of the shared key from its ticket.

find one of the keys, since each key works only temporarily. Each trusted site has a unique *master key* that it shares with the KDC. The master key allows each site to talk to the KDC safely. In addition, the KDC can cryptographically "package" temporary keys using the master keys so that one site can safely forward the right keys to another site. This reduces the number of network messages by providing a form of indirection.

Perhaps this topic might seem to stray a little from the topic of authentication, but if we think of these shared temporary keys as base secrets, the KDC's role emerges. If two computers need to authenticate each other's messages, the KDC can provide a shared base secret to do this. For example, imagine that John needs to retrieve his e-mail from the mail server. Traditionally, John might provide a reusable secret password to his server, and the server would use it to authenticate John directly or redirect the password to a separate authentication server.

TICKETS

Figure 12.1 shows a third alternative: using a KDC. John starts by contacting the KDC, which provides him with encrypted credentials, called *tickets*. John forwards the appropriate ticket to the mail server, which uses that ticket to authenticate John. (As a matter of historical accuracy, the term *ticket* originated with the Kerberos protocol, so we're really looking at prehistoric tickets here.)

In essence, a ticket is an encrypted copy of a temporary base secret. The ticket is encrypted with the master key known only by the KDC and the ticket's intended recipient. Figure 12.2 shows how this works. The KDC maintains a database of reusable shared secrets, the master keys. Each user or server known to the KDC has its own master key. When John contacts the KDC for access to the mail server, the KDC generates a temporary shared secret for him to use with the mail server. Then the KDC encrypts two copies of the secret, one for John and one for the mail server, each using their respective master key from the database.

Finally, the KDC delivers these tickets to John, who delivers a ticket to the mail server. Upon receipt, the mail server uses the mas-

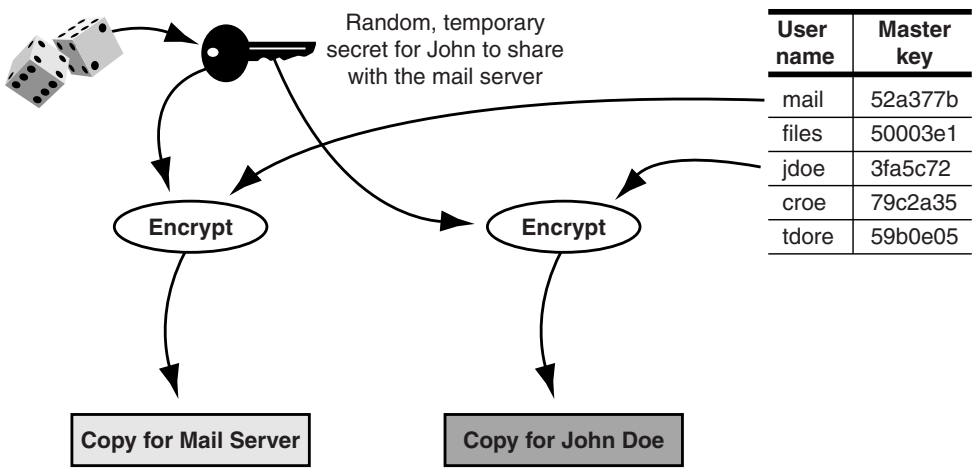

FIGURE 12.2: *Simple "tickets" generated by a KDC.* Fundamentally, a ticket is an encrypted copy of a temporary secret intended to be shared between a particular pair of computers. In this case, John Doe has requested a ticket for use with the mail server. In the upper left, the KDC generates a random secret. Then it produces the tickets by encrypting copies of the secret with the recipients' master keys, taken from the KDC's master key database. This ensures that only the recipients can decrypt the shared temporary secret.

ter key to decrypt the ticket and extract the temporary shared secret. Since nobody else knows the master keys shared between individuals, servers, and the KDC, nobody can decrypt tickets except the intended recipients.

Once the server has the temporary shared secret in hand, John can send the server his e-mail request and use the shared secret to authenticate his request. Trivially, the server could present a traditional password prompt and accept the temporary shared secret as the legal password. More likely, the server will use the secret to perform a challenge response handshake with John's workstation, so that the shared secret doesn't need to be transmitted "in the clear."

This basic approach to key distribution is, of course, too simple to use safely. The fundamental problem is that John has no way of knowing if the key he receives for the mail server is in fact for the mail server. For example, an attacker (someone named Henry, for example) might have intercepted John's request. Henry could then have substituted his own name for that of the mail server. If John uses the resulting keys, Henry will be able to intercept John's messages and read them. To combat such problems, the X9.17 protocol incorporated extra data in key distribution messages, notably message authentication codes, time stamps, and the names of senders and recipients. This followed a number of recommendations produced by computer scientists intrigued by the challenges of key management, starting with Roger Needham and Michael Schroeder.

A-75

NEEDHAM-SCHROEDER

In 1978, Needham and Schroeder published a simple protocol to efficiently address the forgery problems faced by the KDC. This Needham-Schroeder protocol incorporates nonces and a challenge response to detect forged or replayed messages. We can look at the protocol in two parts: first the KDC portion, and then the challenge response portion.

D-75

see Note 2.

Figure 12.3 illustrates the KDC portion of the protocol. John sends the KDC a request that identifies who it's from (John), John's destination (the mail server), and a randomly generated nonce. Like before, the KDC generates a shared key. However, the ticket for the mail server now contains John's user name as well as the shared key. Then the KDC combines the ticket, the nonce, the shared key,

and the mail server's name into a single response to John, all encrypted with John's key.

Upon receipt, John decrypts the data from the KDC. John can verify that the response isn't caused by a replay of an earlier request by checking that the nonce value is correct. Likewise, he can verify that an attacker didn't change his request, say, to point to a different recipient, by verifying that the ticket is intended for the mail server. If either of these had happened, the reply's encrypted contents would reflect it.

An important assumption here is that the KDC doesn't use a stream cipher or other encryption technique that's vulnerable to a rewrite attack (A-72 in Section 11.3). Otherwise the attacker could make systematic changes to the reply, like replace one nonce with another, or modify the stated recipient. The KDC must use a cipher that diffuses plaintext bit values across numerous other bits: a

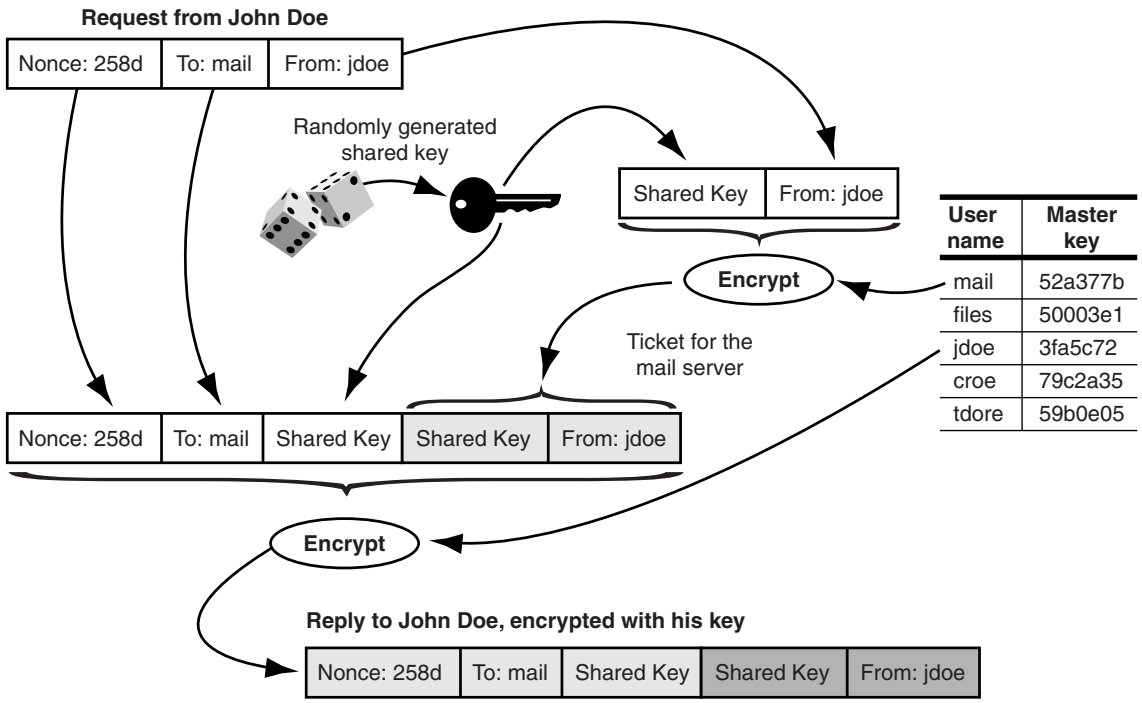

FIGURE 12.3: *The KDC portion of the Needham-Schroeder protocol.* The KDC constructs a ticket that contains a randomly chosen shared secret and the name of the client requesting the key. The KDC then sends back a response encrypted with the client's secret key that includes the client's nonce, the shared key, and the name of the server for whom the ticket was created.

block cipher. Moreover, the block cipher should operate in a chaining mode that makes different parts of the ciphertext depend on one another. This will cause changes to the ciphertext to yield unexpected or invalid results when decrypted.

It might seem that John can finish the key exchange by simply transmitting the ticket to the mail server. John is reasonably certain that he's not using a "stale" message from the KDC that has been replayed by an attacker. On the other hand, the mail server can't be certain that an attacker isn't simply replaying one of John's earlier tickets. If an attacker tricks the mail server into accepting an earlier ticket, the attacker could then replay some earlier session John had with the mail server. For example, the attacker might cause the mail server to accept a duplicate copy of some earlier e-mail message that, perhaps, performs some valuable transaction (like paying money to the attacker an additional time).

To prevent this, the Needham-Schroeder protocol includes a challenge response portion (Figure 12.4). This verifies that the ticket came from a system that actually possesses a copy of the ticket's shared key. Once the mail server receives the ticket from John, it uses the key it shares with the KDC to decrypt the ticket. It verifies that John's user name is in the ticket. Then the server generates a nonce, encrypts it with the key from the ticket, and sends the nonce to John. Upon receipt, John decrypts the nonce, subtracts one from it, and sends it back to the mail server. This allows the mail server to verify that John does indeed have the other key, since otherwise John could not have decremented the nonce.

The main reason for using temporary session keys is that there's always a risk that attackers will occasionally get a hold of a session key. They may do it through brute force cracking or perhaps by penetrating one of the computers using that key. Following that logic, the researchers Dorothy Denning and Giovanni Sacco realized that an attacker with a session key could reuse the corresponding ticket with impunity. For example, imagine that the attacker had intercepted one of the session keys shared between John and the mail server, and also had a copy of the corresponding ticket. The attacker could return to the mail server any number of times with that ticket and convince the server to use the ticket again. The mail server would have no way of knowing that the ticket wasn't being used by

John, since the attacker could use the corresponding session key to handle the challenge response.

see Note 3.

To solve this problem, Denning and Sacco proposed an alternative to the Needham-Schroeder protocol that used *time stamps* instead of nonces. In John's case, his computer would insert the current time in the request being sent to the KDC. The response from the KDC would contain the same time stamp instead of the nonce. In addition, the KDC would include the time stamp in the mail server's ticket. When the mail server decrypts the ticket from John, it can now check that the ticket is from John and that he produced the ticket recently. This greatly reduces the risk of a replay attack, and the technique found widespread use in the Kerberos protocol.

D-77

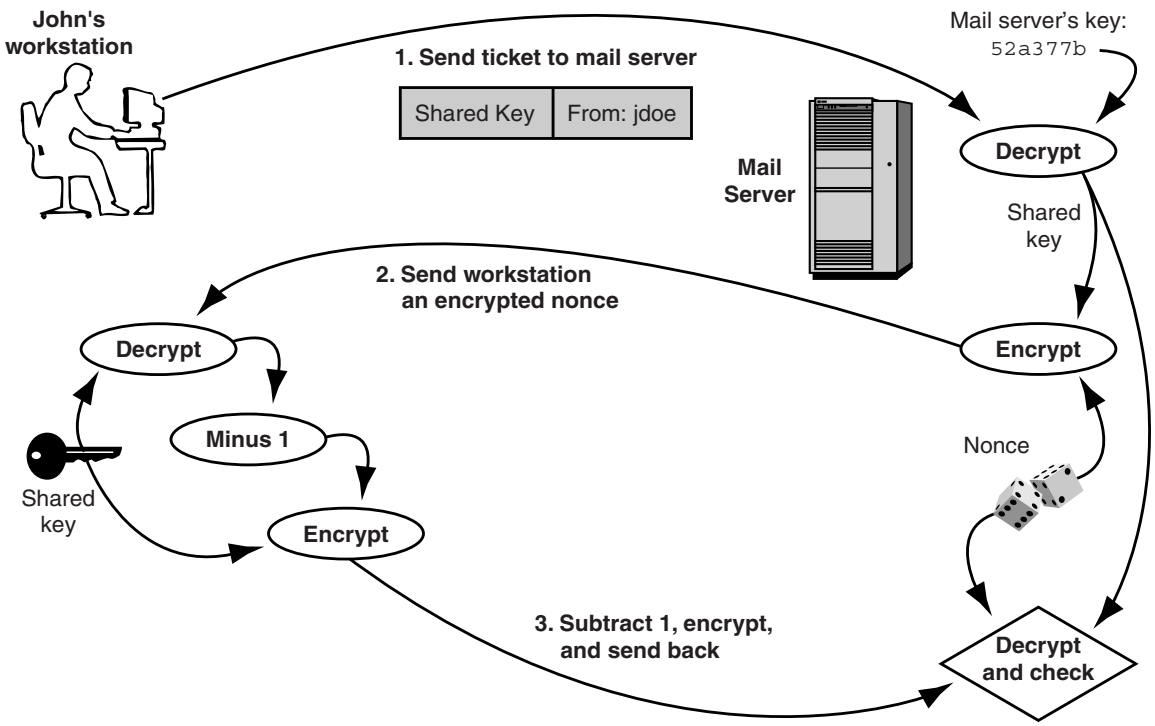

FIGURE 12.4: *The challenge response portion of Needham-Schroeder.* John's workstation has received a mail server ticket from the KDC as described in Figure 12.3. He sends it to the mail server, which extracts the shared key as shown on the right. To ensure that both John and the server are using the same key, the server encrypts a nonce and sends it to John. As shown on the left, John's workstation decrypts the nonce, subtracts 1 from it, encrypts it, and sends it back. The server verifies the response by decrypting it and checking it against the original nonce.

12.2 KERBEROS

In 1983, Project Athena was started at MIT as a model for the envisioned "next generation distributed computing" environment for academic environments. The Project Athena team decided to design their security solution around a KDC based on the Needham-Schroeder protocol but incorporating the work of Denning and Sacco. To make Kerberos work in a large-scale environment, Project Athena had to provide software to handle logons at client workstations and adapt servers to accept the Kerberos protocol. Software adapted to work with Kerberos is generally referred to as having been *Kerberized*. *see Note 4.*

By 1989, Steve Miller and Clifford Neumann had produced four versions of Kerberos with help from others at MIT. Version 4 was the first publicly released version, and is still used to some extent. Version 5, however, is the standardized version in the Internet community, and that is the version we look at here. *see Note 5.*

THE AUTHENTICATION SERVER

The Kerberos KDC contains several separate servers that provide different functions. The "authentication server" provides a protocol most similar to Needham-Schroeder. In theory, this server can issue tickets to talk to any Kerberized service. In practice, most workstations only use the authentication server to issue tickets to the "ticket-granting service," which we will discuss in a later section. Table 12.1 summarizes the essential contents of a Kerberos ticket.

To collect a ticket from the authentication server, John must construct a KRB_AS_REQ message (Figure 12.5). Like the Needham-Schroeder protocol, John provides his own identity, the name of the desired server, and a nonce when requesting a ticket. In addition, the protocol establishes a time period during which the shared key, called the *session key* in Kerberos, should be valid. Kerberos also incorporates a workstation identifier into the keying protocol, so that it can control which workstations are allowed to use a particular ticket. This reduces the likelihood that an attacker can misuse a ticket. The authentication server responds by constructing a KRB_AS_REP message containing the ticket and its associated status information.

TABLE 12.1: *Essential Contents of a Kerberos Ticket*

Field	Purpose
User name	The client who requested the ticket
Server name	The desired service. This ticket is encrypted using this server's master key
Validity period	The time at which the ticket, and its corresponding session key, begin to be valid, and the time at which the key and ticket cease to be valid
Session key	The secret key being shared between this user and server
Workstation	An identifier for the computer (or computers) on which this user may reside

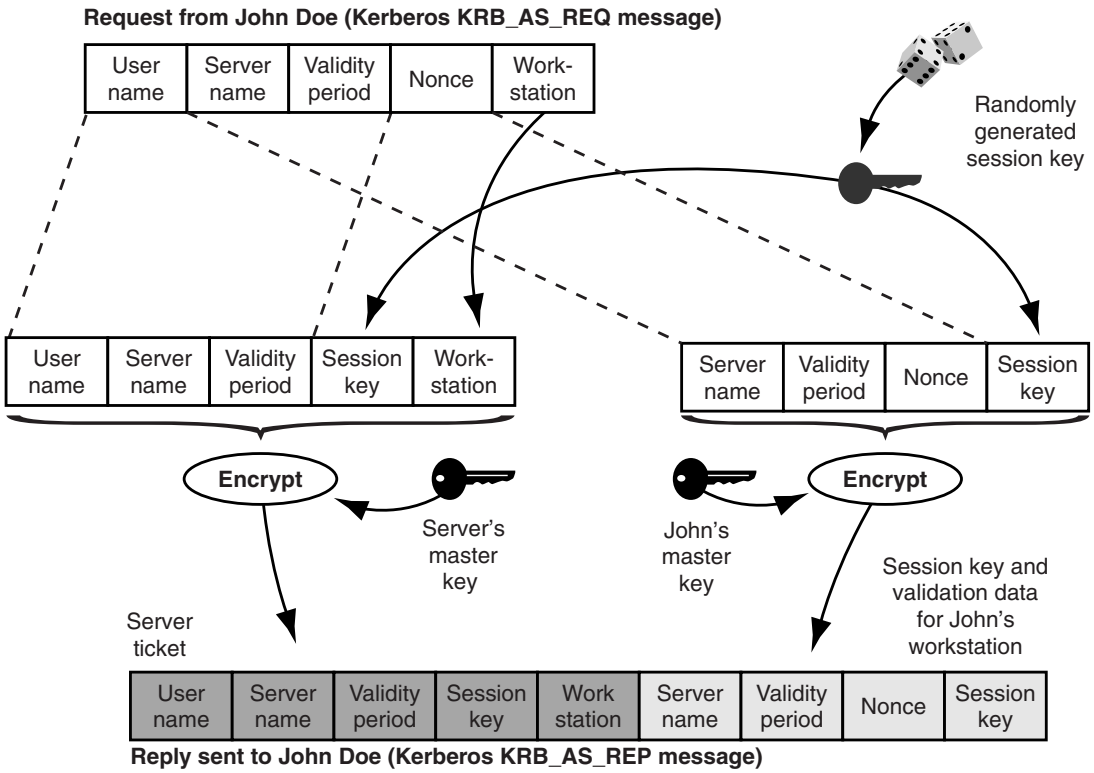

FIGURE 12.5: *The Kerberos authentication server.* Users must go to this server for their first ticket after logging on. The protocol is an improved version of Needham-Schroeder. The validity period establishes when and how long a ticket and its corresponding session key should be used. This addresses the replay problems identified by Denning and Sacco. The workstation identifier establishes which computers may use a particular ticket.

John validates the ticket by checking the status information the server provides in the KRB_AS_REP message. In particular, he needs to verify that the reply contains the correct server name, nonce, and validity period. Then he can safely use the ticket and session key to contact the server.

AUTHENTICATING TO A SERVER

Figure 12.6 shows how John uses his Kerberos ticket and the corresponding session key to construct a KRB_AP_REQ message to authenticate himself to the mail server. In addition to the server's ticket, John provides a Kerberos *authenticator*, which is an encrypted data item containing John's user name and a time stamp. John uses the same session key that appears in the ticket to encrypt the authenticator.

Upon receipt, the mail server first decrypts the ticket. Then it extracts the session key and uses it to decrypt the authenticator. The authenticator's user name should match the one in the ticket, and the time stamp should be recent, usually within the past five minutes. If the request passes these tests, the server constructs a KRB_AP_REP message, if the user asked for one. This reply sends back the request's time stamp, encrypted with the session key. As with Needham-Schroeder, this reply is only meaningful if the

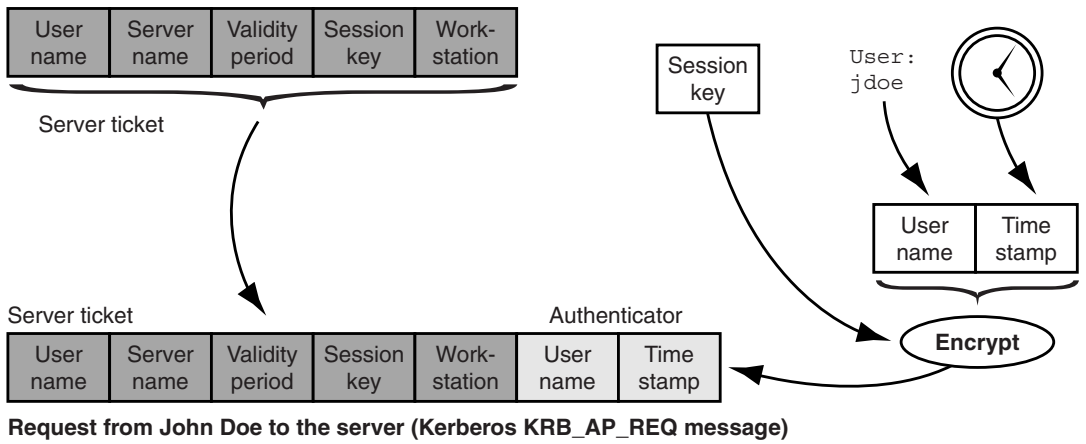

FIGURE 12.6: *Authenticating to a Kerberized server.* When John sends a ticket to the server, he must also provide an authenticator, which contains his user name and a time stamp, both encrypted with the session key.

encryption does not permit precise and reliable rewriting or cut-and-paste attacks.

TICKET-GRANTING SERVICE

Although it's possible to use the Kerberos authentication server to generate tickets to individual service, this poses a problem. Kerberos needs to use one's master key to process messages with the authentication server, and most people tend to use many services while working on a computer. If we store the master key on the workstation while someone is using it, we run the risk that someone else will steal it. To eliminate this problem, we need to use the master key for as short a time as possible and erase it from the workstation as soon as we can. But this presents another problem: if we erase the master key once we finish connecting to one server, we'll have to read it in again when we try to connect to another. Traditionally, Kerberos has used memorized passwords as users' master keys, so this would present people with numerous password prompts. Neither alternative is practical.

So here we are facing the same dilemma that led to session keys: we need a temporary key that we can use for issuing other temporary keys. Indeed, we solve the problem in the same way: instead of leaving the master key on the workstation while John is logged in, we come up with a special session key we can use to issue our tickets. This applies a well-known piece of computing wisdom: you can often fix things by adding another level of indirection.

Kerberos implements this additional temporary key by adding a special server to the KDC called the *ticket-granting server*. This server accepts tickets that are called, of course, *ticket-granting tickets* (TGTs). Users can send their TGTs to this special server to request tickets for other services.

In previous examples, we spoke of John Doe's establishing a connection to the mail server via the KDC. In practice, of course, John won't just connect to the mail server. More likely, he'll also routinely connect to two or more different file servers, a print server or two, and various other services. John's workstation will undoubtedly connect itself to several of these services automatically when he logs on. But there may be other services that he doesn't routinely use or that it doesn't make sense to connect with until needed.

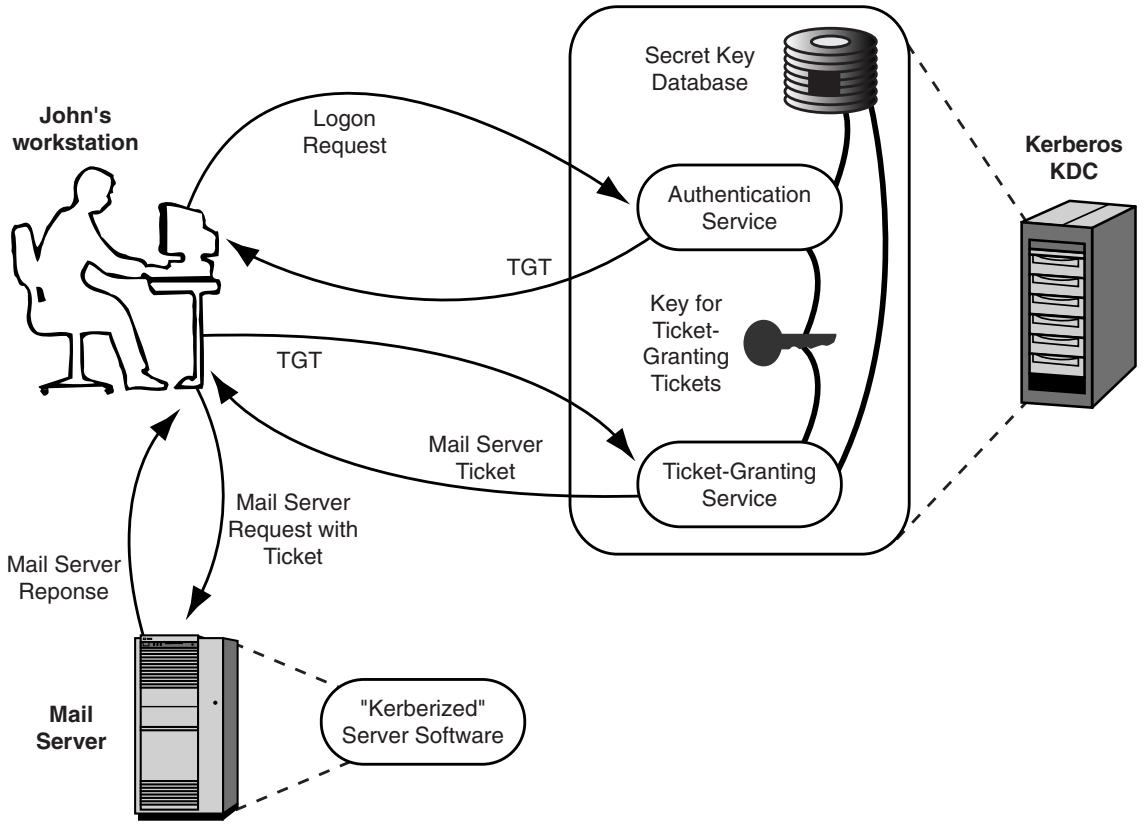

FIGURE 12.7: *Kerberos and ticket-granting tickets.* Kerberos can split the ticketing process into two parts: first, John can retrieve a "ticket-granting ticket" (TGT), and then he uses the TGT with the ticket-granting service to acquire individual tickets to use individual services. He needs to enter his master key only when he requests the TGT, and thereafter it does not need to reside on his workstation and be vulnerable to attack.

Kerberos provides a relatively safe and user-friendly mechanism for single sign-on by using the ticket-granting server (Figure 12.7). When John signs on to his workstation, it immediately contacts the Kerberos KDC's authentication server and collects a TGT. Then the workstation collects John's password (or other authentication data) and uses it to decrypt the reply from the authentication server. Then the workstation passes the TGT to the ticket-granting server to collect a ticket for every service he immediately needs: his usual file servers, the mail server, the print server. Later on, the workstation can again go to the ticket-granting server for more server tickets if

John needs to use additional servers. The workstation never needs to prompt John again for his password: it simply uses the TGT and its corresponding session key to collect additional tickets.

The ticket-granting server issues tickets through an intricate but fairly straightforward process. Figure 12.8 shows the first part of this process. John sends his request, which contains an authenticator, a TGT, and details about this request: which server (the mail server), when and how long it will be used, and a nonce.

Like all other servers, the ticket-granting server has its own private key, which Kerberos uses to encrypt its tickets. Upon receipt of a TGT, the server decrypts it and extracts the session key being used for issuing John's tickets, called the *ticketing key* here. The server uses the ticketing key to decrypt the authenticator and check its

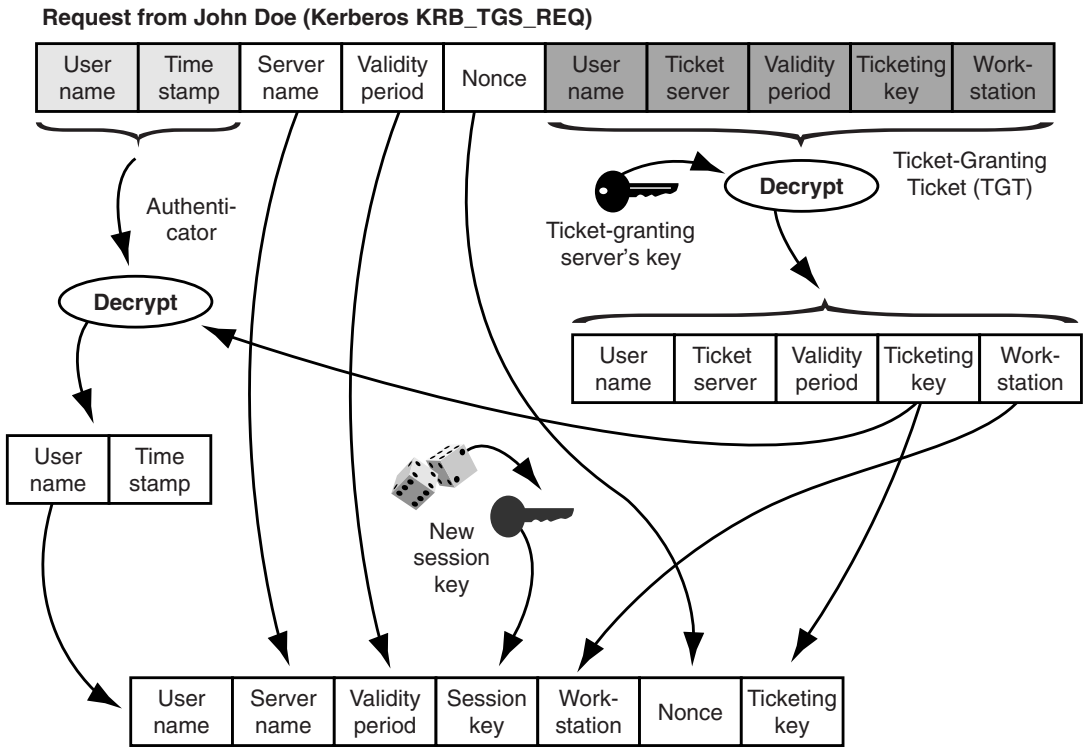

FIGURE 12.8: *Using a TGT request to construct a ticket for a service, part 1.* John sends the ticket granting server a copy of his TGT, an authenticator, and information about the mail server ticket he needs. The ticket-granting server decrypts and validates his request, using their ticketing key. Then it generates a session key for John to share with the mail server.

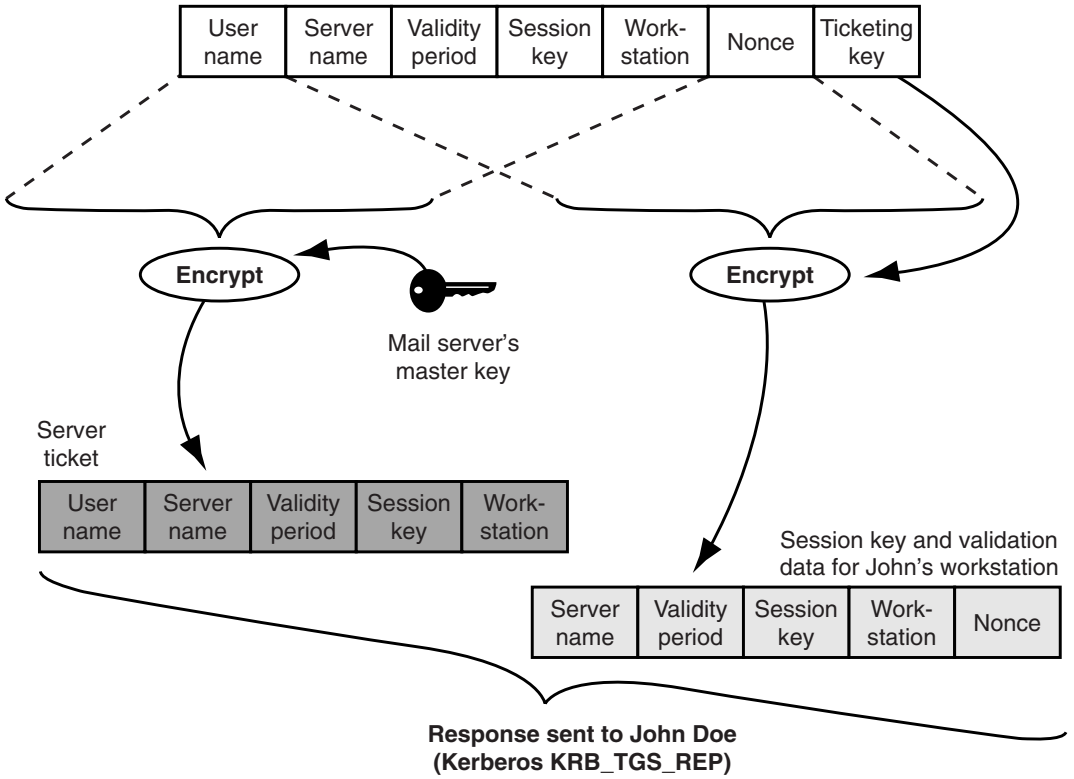

FIGURE 12.9: *Using a TGT request to construct a ticket for a service, part 2.* The ticket-granting server takes the information it has collected and formats it for encryption. The mail server ticket is encrypted with the mail server's master key. The rest of the reply is encrypted with John's ticketing key, taken from the TGT.

validity. The server also checks the validity period of the TGT against the current time and the validity period John's workstation has requested for this new ticket. Next, the server generates a random session key to be used by John's workstation and the mail server.

Figure 12.9 shows how the ticket-granting server takes the data to be returned to John and encrypts it for delivery. First, the server constructs the mail server ticket and encrypts it with the mail server's master key. Next, the server assembles the rest of the data required in its reply, and encrypts that data with John's ticketing key. Both blocks of encrypted data are then combined to produce the reply message that the server sends to John.

12.3 User and Workstation Authentication

Careful readers may have noticed that the KDC protocols like Kerberos really emphasize the authentication of people to network services. The classic protocols do little or nothing to establish a user's identity for the benefit of a workstation or even the KDC. There is nothing to prevent anyone, including an attacker, from requesting tickets for authenticating someone else to a particular server. Anyone that can communicate with John Doe's KDC can send a message claiming to be John and retrieve a ticket for authenticating John to the mail server. The subsequent protocols simply ensure that attackers can't really use the tickets they receive.

This situation has two interesting implications. First, it means that Kerberos doesn't necessarily treat the workstations themselves as distinct entities that require authentication. Instead, they're assumed to be interchangeable devices, like they might be in an academic lab environment. Second, it means that attackers might be able to collect enough tickets associated with a particular person to try to mount an attack on their master key. Fortunately, these implications aren't cast in stone, and Kerberos implementations can overcome them.

WORKSTATION AUTHENTICATION

A fundamental feature of the KDC protocols we examined in the previous sections is that the protocols rely on a single key, the user's master key, to secure the activities of a workstation. Unlike the Windows NT protocol examined in Section 11.4, the workstation doesn't have a separate key for communicating with the KDC or any other security service. Since Kerberos implements indirect authentication, it doesn't help the workstation itself to authenticate individual users. The workstation simply serves as a vessel by which authorized users can manipulate resources on servers. Anyone with physical access can, in theory, use any workstation since Kerberos itself provides no workstation-specific authentication services.

The Kerberos approach clearly reflects a "thin client" view of distributed computing. If client workstations are simple and fairly uniform across a network, people can use them interchangeably. It doesn't matter which workstation a person uses; the principal objec-

tive of authentication is to ensure that a user like John doesn't retrieve server resources that don't belong to him.

However, this vision doesn't fit the reality in sites dominated by personal computers on individual desks. Each personal computer contains local files belonging to its particular owner or custodian. In many cases it contains software that is individually licensed to its owner. If Kerberos will authenticate anyone on any workstation, then it can't control access to these personal resources. In such cases the workstation will have to provide an additional authentication procedure. This could consist of local authentication or it can use Kerberos-based "preauthentication" described later in this section. Microsoft uses a combination of these techniques in Windows 2000, as described in Section 12.6.

In an ideal world that uses perfectly reliable encryption, it shouldn't matter that attackers can request and receive tickets intended for other users. But in fact we never have as much confidence in our cryptographic algorithms as we might wish. Thus, we generally avoid putting information at risk when we don't have to.

In the case of Kerberos tickets, there is a risk that attackers could collect enough tickets intended for some other important user to mount a successful attack against that user's master key. In a classic Kerberos environment this is a serious threat because personal master keys are based on memorized passwords. For example, imagine that some very important person (the Pope, the President of the United States, Bill Gates, Jodie Foster, or some other notable) uses Kerberos for authentication through a large-scale KDC. This means that anyone else who can contact that KDC can retrieve "papal tickets," or tickets belonging to any other VIP there. If the particular VIP is careless with password selection, then attackers might be able to crack the VIP's password by collecting tickets and mounting a dictionary attack on them.

A-78

PREAUTHENTICATION

Kerberos Version 5 introduced *preauthentication* so that sites could authenticate requests sent to the KDC instead of relying on servers to authenticate the requests later. Administrators can configure Kerberos to require preauthentication on requests for tickets or TGTs so that the KDC only distributes encrypted data to those who

D-78

already know the associated key. Typically, the process is used to authenticate a user when retrieving an initial TGT from the KDC, though the protocol can support a broad range of alternatives. Initial preauthentication should be available as an option in any truly interoperable version of Kerberos.

In traditional Kerberos authentication, the workstation collects keys from the KDC before it needs the user's master key. Since Kerberos typically uses people's passwords as their master key, the workstation can defer the password prompt until it hears from the KDC. With initial preauthentication, the workstation must collect the user's master key first, which usually means collecting the password.

Next, the workstation constructs the initial KDC request, which usually requests a TGT. For preauthentication, the workstation adds a specially formatted, encrypted time stamp. The time stamp includes the current time of day and a one-way hash of the rest of the KDC request, all encrypted with the user's master key.

When the KDC receives the request, it checks to see if preauthentication is required and, if so, rejects the request if it is missing. If the preauthentication time stamp is present, the KDC looks up the user's master key and uses it to decrypt the time stamp. If the time stamp's time of day is reasonable, the KDC computes a one-way hash over the rest of the request and compares it against the hash in the time stamp. If they match, the KDC fulfills the request.

Note that the workstation can treat the KDC's response as confirmation of the user's identity. The KDC will send back a legitimate response message instead of an error message only if the preauthentication succeeds. The workstation can decrypt the KDC's response and verify that it contains the correct nonce, server name, and validity period. If not, then the workstation can conclude that the user is trying (unsuccessfully) to masquerade as someone else.

12.4 Ticket Delegation

As we've described it so far, John's tickets can be used only by John himself to fetch and use his resources. But as distributed systems have evolved in sophistication, we've encountered situations in which servers may need to contact other servers on John's behalf. For example, the mail server undoubtedly needs to read John's mail

queue files, which no doubt reside on a file server somewhere. The e-mail server somehow needs to get permission to read John's queue. Traditionally, administrators grant the mail server the god-like power to read all files, or at least all mail files, but this has also been a source of security problems. If an attacker tricks the mail server into using the wrong identity, the attacker can read the other user's mail. Even worse, if an attacker penetrates the mail server, the attacker could gain free access to everyone's mail file.

The preferred solution is to give the server only as much permission as it needs to perform its authorized activities. If John asks the server to retrieve his mail, then the server should have access only to his mail files and to whatever administrative files are essential for retrieving that mail. The mail server is confident that it speaks to John, because it has a valid ticket from him. But if the mail server turns around and asks the file server for files on John's behalf, how does the file server become confident that the request is honestly on John's behalf? After all, there's always a risk that someone has penetrated the mail server and is making bogus requests. Moreover, Kerberos tickets identify the address of the computer that requested them, and they aren't supposed to be honored if the addresses don't match.

Kerberos Version 5 provides mechanisms so that workstations can delegate their ticketing privileges to servers under limited circumstances. This involves two features added to the TGT. The first feature allows *proxiable tickets*, which permit a TGT's owner to request tickets tied to computers with different network addresses. The second feature allows the TGT to be *forwardable*, which permits the owner to ask for another TGT that is tied to a different network address. Also, Kerberos uses a similar mechanism to authenticate users between different KDCs ("realms") using a special TGT called a *referral ticket*.

PROXIABLE TGT

If John's workstation software wants to use proxy tickets, then it starts by requesting a TGT with the "proxiable" option allowed. When John needs to read his mail, he first collects a mail server ticket, and uses it to authenticate himself to that server as usual. Since the mail server also needs to contact the file server on John's

behalf, John's workstation must also request a *proxy ticket* for the file server. This proxy ticket contains the network address of the mail server. John forwards this ticket to the mail server along with a copy of the associated session key. The session key is encrypted using the key that John shares with the mail server, so that the mail server can retrieve that key. The mail server then forwards the proxy ticket to the file server. Upon receipt, the file server sees that the ticket contains the network address of the mail server, so the ticket is accepted. The file server decrypts the ticket and retrieves the session key. Since the mail server and file server share that key, they can authenticate one another and proceed with the processing of John's mail.

FORWARDABLE TGT

A problem with proxy tickets is that the originating workstation must somehow be able to predict which proxy tickets will be needed. Even worse, the workstation and server could find themselves exchanging numerous messages trying to establish an accurate list of the required proxy tickets. The forwardable TGT eliminates this problem by letting the server ask for the necessary tickets itself.

If John's workstation software wants to use a forwardable TGT, then it starts by requesting a TGT with the "forwardable" option allowed. When John needs to read his mail, he again collects a mail server ticket and uses it in the normal way. Then John contacts the ticket-granting server and requests it to issue him another TGT, this one containing the network address of the mail server. John forwards that ticket to the mail server along with an encrypted copy of the TGT's ticketing key. Now the mail server can contact the ticket-granting server directly to retrieve tickets for whatever server it needs to contact. The ticket-granting server honors the ticket requests on John's behalf because the mail server's network address matches the address in the forwarded TGT. Of course, the request also contains a valid authenticator that the server has encrypted with the TGT's ticketing key.

These different techniques reflect different trade-offs between security and operating convenience. While both eliminate the need to give servers unrestricted access to other servers, both pose their own risks and costs. The proxy approach lets the user's software

decide what server permissions it will delegate to other servers, but this requires the software to know which proxy tickets will be needed. The forwardable TGT approach lets servers collect whatever tickets they need on behalf of a user, but there's no strong mechanism to prevent a subverted server from abusing this power. The only limit is in the ticket's expiration date and, possibly, in permission fields copied into tickets from the TGT. These permission fields are specific to particular software environments and applications. This trade-off depends heavily on the security needs of particular sites and particular application environments, so these delegation features form an optional part of Kerberos. They are only available if the site or application environment allows them to be used.

REALMS AND REFERRAL TICKETS

A proprietor's enterprise can establish two or more Kerberos KDCs if a single KDC can't satisfy the enterprise's needs. The proprietor can establish separate KDCs to reflect organizational or geographical distinctions, or to implement separation of duty. In Kerberos parlance, the community of users, servers, and other entities registered with a particular KDC represent a single *realm*. Entities within the enterprise are assigned names based on their realm, and multiple realms can be structured hierarchically. Authorized users in one realm can be authenticated to servers in other realms using the Kerberos protocols for "cross-realm authentication." This allows sites to handle user administration locally but still authenticate users across a large-scale networking environment.

For example, let's say that John needs to use a special-purpose printer in another realm to print a document for someone at a different location. John can use that printer if he can get a TGT for the printer's KDC. To do this, John asks his own KDC for a *referral ticket* to the printer's KDC; the referral ticket serves as the TGT he needs. To create the referral ticket, John's KDC uses an "interrealm key" that it shares with the printer's KDC. Once John has the referral ticket, he uses it to contact the printer's KDC to retrieve a ticket for visiting that printer.

For one KDC to create a referral ticket for using another KDC, the two must already share an interrealm key. Since this might not be practical in really large organizations with dozens, or hundreds, of

KDCs, a KDC can retrieve a referral ticket through multiple hops if necessary. The KDC can do this by following the branches of the organization's realm hierarchy, as long as every KDC has shared an interrealm key with its parent and children in the hierarchy.

This approach allows anyone in a large-scale Kerberos network to be authenticated to any server within that network. In theory, cross-realm authentication could even span multiple enterprises. However, there remains the question of how the servers make authorization decisions. Once they identify who has asked for service, they need a mechanism to decide if that person should really receive service or not.

12.5 Attacking a Kerberos Network

A fundamental feature of the Kerberos philosophy is to recognize that some computers in its network will be successfully attacked. This should be obvious, since Kerberos relies on reusable passwords as base secrets. The Kerberos design tries to minimize the system-wide implications of intrusions on individual workstations and servers. The overall security of a Kerberos site also relies on the assumption that all clocks are roughly synchronized on all participating computers. Kerberos should meet its security objectives as long as those assumptions remain accurate.

INTRUSION TOLERANCE

Kerberos tries to provide a practical degree of security with minimum reliance on special properties of computers within the network. Successful attacks on individual workstations or servers cannot compromise the overall security of a Kerberos network. This is because neither the workstations nor the servers require any special permissions to work with the KDC.

This is a practical and realistic strategy, especially in an academic setting. Commercial computer systems are notoriously vulnerable to attack. While it's possible to reduce the risk if administrators put a lot of effort into hardening a particular host, this effort isn't practical in a large-scale networking environment. In any case, people with computers on their desks will, over time, disable some security measures either accidentally or intentionally. Moreover, the distribution

of authority in an academic environment does not give a network security administrator a way to reliably install security measures on individual machines.

Kerberos adapts to this situation by tolerating intrusions to a certain extent. If a single computer is compromised somehow, the only possible damage will affect that particular computer and no others. At worst, the attacker might manage to retrieve a master key belonging to the user or server. Most often, however, the only keys available for attack should be the temporary session keys. Loss of those keys should cause only short-term damage. While it is true that attackers can affect large parts of the user population by penetrating a heavily used server, the risk of damage in such a case has more to do with local computing policies than it does with the Kerberos architecture. In such cases, a cautious site can reduce risks by deploying numerous independent servers instead of concentrating users and services on only a handful of computers.

The KDC, however, must successfully resist attack in order to maintain overall network security. If attackers manage to penetrate the KDC's defenses, they can copy the database of master keys and then masquerade as any user. They might even be able to masquerade as the KDC. Cautious Kerberos sites generally assign the KDC to a dedicated computer, and the administrators remove and disable all other services on that computer. This reduces the risk of an attack by reducing the number of entry ways into the computer.

CLOCK SYNCHRONIZATION

Clock synchronization is a crucial requirement of computers operating in a Kerberos environment. Tickets become valid and expire within established time periods; if clocks are wrong, then tickets won't work when they should. Worse, they won't expire when they should.

This provides an opening for an attack. Imagine that an attacker manages to recover the session key corresponding to a particular ticket. The Kerberos approach assumes that such things might happen occasionally, but expects to limit the damage by limiting the validity period of tickets. But what happens if the attacker manipulates a server's clock? There are many ways to do this. An attacker might somehow manage to penetrate far enough into the server to

convince its clock to change. More likely, however, the attacker would simply send messages using the Network Time Protocol that trick the server into changing its clock. Then the attacker can transmit the expired ticket and the server will mistakenly accept it as being legitimate.

A-79

The usual solution is to require authentication on messages from the time server. In practice, though, it can be tricky to provide such authentication at a level of confidence that compares well with Kerberos. Classic strategies for time authentication are based on shared base secrets, like passwords, and suffer from the classic risk of sniffing and replay. A more promising approach would be to use "public key" authentication as discussed in the next chapter. Ultimately, however, accurate clocks will need to rely on vigilance and regular checking. Bellovin and Merritt have noted that an attacker could construct a radio transmitter that generated bogus WWV time signals, and that the transmitter might be able to fool the time server itself into distributing the wrong time.

D-79

see Note 6.

12.6 KERBEROS IN WINDOWS 2000

In Windows 2000, Microsoft replaced the Windows NT domain authentication mechanism described in Section 11.3 with Kerberos. Windows domain logon has become a transaction that retrieves a TGT. File server mapping now involves the exchange of tickets and session keys. Master keys and other security-critical information is stored in the *Active Directory* facility. Although the Windows implementation of Kerberos contains a number of distinctive elements, Microsoft states that the implementation will comply with standard Kerberos interoperability requirements. In particular, non-Windows Kerberized applications should be able to process Windows tickets to yield a single sign-on capability between Windows and non-Windows applications.

see Note 7.

The Kerberos protocol has provided Windows 2000 with three particular benefits in comparison to earlier Windows products. First, it provides faster server authentication, since the server doesn't have to contact the domain controller for indirect authentication. Instead, the server simply processes the ticket. Second, Windows uses the ticket delegation features to pass a user's access rights to a server by way of another server. Third, Windows uses Kerberos protocols

that allow KDCs belonging to different organizations to grant access to each other's users in a controlled manner.

Naturally, Microsoft could not simply substitute new network authentication protocols for old ones, abandoning its existing products and customers. Windows 2000 still supports the older NT domain protocols. If required, Windows 2000 can even support the older, security-challenged LANMAN authentication protocols described in Section 10.4.

MASTER KEYS AND WORKSTATION AUTHENTICATION

It is inevitable that some adjustments take place when a rather mature product line like Windows incorporates an even more mature technology like Kerberos. Microsoft needs to tread a fine line to maintain compatibility with its existing products while reaping at least some of the benefits of Kerberos' security features. We see this when we look at how Microsoft has adapted Kerberos to the Windows workstation logon process.

The workstation user, of course, is supposed to see nothing different. When a user like John Doe tries to log on, he types the usual magic keystroke and sees the password dialog on his screen. He enters his user name, selects his domain, and types his password. Underneath, Windows converts it all into Kerberos transactions.

A significant difference between traditional Kerberos and the Windows implementation is that Windows workstations are treated as distinct entities. Each workstation has its own identity within the realm and its own master key, separate from the master key of a logged-on user. In this example, John has named his workstation *bat*, and that's its name within the domain. The logon process retrieves a ticket to authenticate the user to the workstation in addition to any other tickets that might be needed.

In Windows 2000, the logon process is divided between three major processes: the *Winlogon* process that prompts John for his user name and password, the *Security Support Provider (SSP)* that communicates with Kerberos, and the *Local Security Authority (LSA)* that protects the workstation itself. This is shown in Figure 12.10. The process takes place as follows:

1. Once Winlogon has collected John's authentication data, it passes the data to the LSA.

2. The LSA converts the password into the master key to be used by Kerberos. The LSA does this by hashing the password as described in Section 10.6. Then the LSA invokes the Kerberos SSP, and passes it John's user name and master key.

   Unlike traditional Kerberos, Windows 2000 retains the master key in a cache. This allows the workstation to use Windows NT domain authentication if necessary for talking to older servers. There is an obvious risk in keeping this information in a cache, and the risk is unnecessary if the entire site uses Kerberos authentication.

3. The Kerberos SSP tries to contact its domain controller and retrieve an initial Kerberos TGT in John's name. The request uses preauthentication based on John's master key.

   If the chosen domain is controlled by an older Windows NT server and Kerberos isn't available, the LSA falls back to using an NT version of the SSP that uses the NTLM protocol described in Section 11.3.

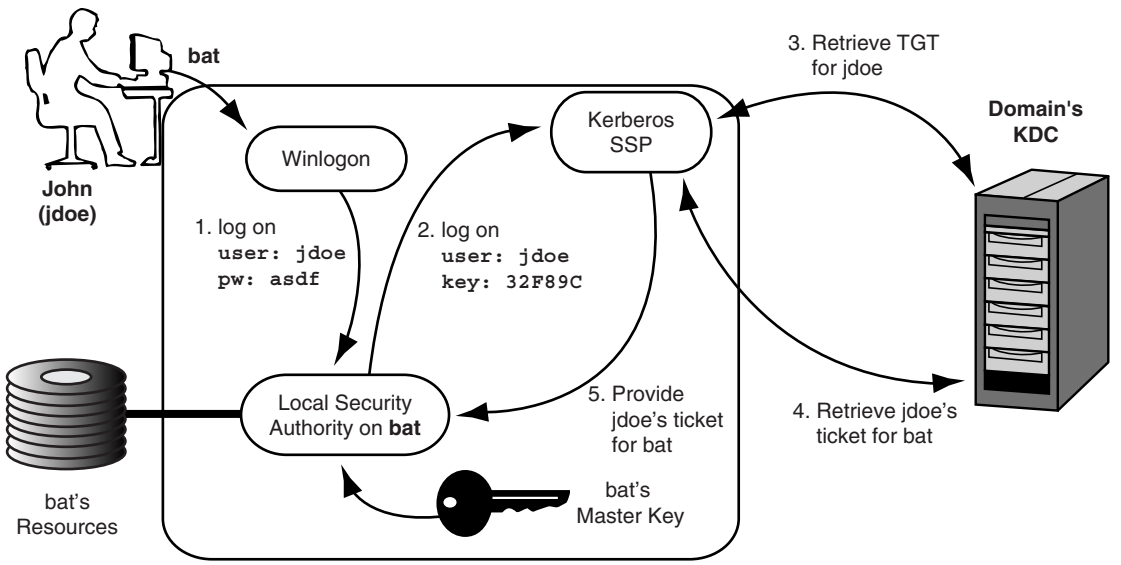

FIGURE 12.10: *Windows 2000 logon with Kerberos.* To log on to his workstation, named "bat," John must collect a ticket for bat from the domain's KDC. John's master key is derived from his memorized password, while bat's master key is a secret value stored in the workstation's Registry. The numbered steps in this diagram are keyed to the discussion in the text.

4. If the SSP receives its Kerberos TGT, it uses the TGT to retrieve a ticket for the workstation named "bat" on John's behalf.
5. Upon receipt of the workstation ticket, the SPP provides the ticket to the LSA, which uses bat's master key to authenticate it. If the key is authentic, the LSA logs John on to bat.

The extra step of collecting a ticket for the workstation itself is an unusual feature compared to other Kerberos environments. The ticket's session key serves no real purpose. Windows 2000 uses this approach because the KDC stores user credentials in each ticket, and this is a relatively clean and consistent way to retrieving a user's credentials from the KDC. Each ticket contains authorization information and other credentials necessary to associate the user with the right resources and access permissions on the workstation. The LSA on a Windows server works the same way: it retrieves credentials from a user's ticket and uses those credentials to run a server process with the identity and permissions of that user. At the workstation, the LSA ensures that applications on the workstation are run in that user's name.

SERVICE AND PROTOCOL SUPPORT

Here is a summary of the services and protocols that use Kerberos authentication in Windows 2000:

- File services, including the usual CIFS/SMB services and distributed file system management
- Printer services
- Web server authentication to the Internet Information Server
- Authenticated Remote Procedure Call (RPC) services for remote server and workstation management
- Queries to the Active Directory using the Lightweight Directory Access Protocol (LDAP)
- Authentication for setting up a host-to-host cryptographic link using the IP Security Protocol (IPSEC)
- Authenticate requests for quality of service levels

12.7 SUMMARY TABLES

TABLE 12.2: *Attack Summary*

Attack 	Security Problem	Prevalence	Attack Description
A-75. KDC request spoof	Recover or modify hidden information	Common	Attacker intercepts a client's KDC request and returns a different set of shared keys whose value is known by the attacker
A-76. Rekey replay	Masquerade as someone else	Common	Attacker sends an earlier ticket to a server and then replays messages sent earlier by the client encrypted with that key
A-77. Off-line cracking and replay	Masquerade as someone else	Common	Attacker cracks a session key off-line and uses this knowledge to reuse that key's ticket
A-78. Off-line master key cracking	Masquerade as someone else	Sophisticated	Attacker requests tickets in the victim's name and uses them to brute force crack the victim's master key
A-79. Forged time change	Masquerade as someone else	Sophisticated	Attacker sends forged time of day messages to a server so that expired tickets are valid

TABLE 12.3: *Defense Summary*

 Defense	Foils Attacks	Description
D-75. Nonce shared with KDC	A-75. KDC request spoof	A nonce is included in requests sent to the KDC and included in responses
D-76. Challenge response by server in KDC protocol	A-76. Rekey replay	Server sends a user a challenge, which requires a response that depends on the user's encrypting data with the session key

TABLE 12.3: *Defense Summary (Continued)*

Defense	Foils Attacks	Description
D-77. Time stamps in KDC protocol	A-77. Off-line cracking and replay	KDC messages include time-of-day information to detect attempts to reuse tickets later
D-78. KDC preauthentication	A-78. Off-line master key cracking	User must provide personal authentication information when requesting a TGT
D-79. Authenticated time messages	A-79. Forged time change	Messages that change a server's time of day must be authenticated