1 Kerberos

Kerberos is an authentication service developed as part of Project Athena at MIT. Kerberos addresses the following problem: Assume a distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like the servers to be able to restrict access to authorized users and to be able to authenticate requests for service.

Kerberos authentication is based entirely on the knowledge of passwords that are stored on the Kerberos Server. Unlike UNIX passwords, which are encrypted with a one-way algorithm that cannot be reversed, Kerberos passwords are stored on the server encrypted with a conventional encryption algorithm (typically DES), so that they can decrypted by the server when needed. A user proves his/her identity to the Kerberos server by demonstrating knowledge of the key.

An overview of Kerberos is given in Figure 1. We describe the steps in more detail below.

1. Logging into a UNIX workstation that is using Kerberos looks the same to a user (Kartik) as logging into a regular UNIX workstation. Sitting at the workstation, the user sees the traditional `login:` and `password:` prompts. In Kerberos 4 as soon as Kartik types his username, the workstation sends a message to the Kerberos authentication server. This message contains Kartik's username and indicates that Kartik is trying to login.

2. The Kerberos server checks its database and, if Kartik is a valid user, sends back a *ticket granting ticket* that is encrypted using Kartik's password. The ticket granting ticket contains the following two pieces of information: a session key K_{ses}, and a ticket for the Kerberos ticket granting ticket T_{tgs}, encrypted with both the session key and the ticket granting service's key K_{tgs}, i.e., $E_{K_{tgs}}E_{K_{ses}}\{T_{tgs}\}$.

3. The workstation then asks Kartik to type in his password, and attempts to decrypt the encrypted ticket using the password that Kartik has supplied. If this is successful then the workstation forgets about Kartik's password and uses the ticket granting ticket exclusively. If decryption fails, the workstation gives Kartik another chance to type the right password.

4. Now, if Kartik wanted to access his files on the file server, the system software on the workstation Kartik is currently on, contacts the ticket

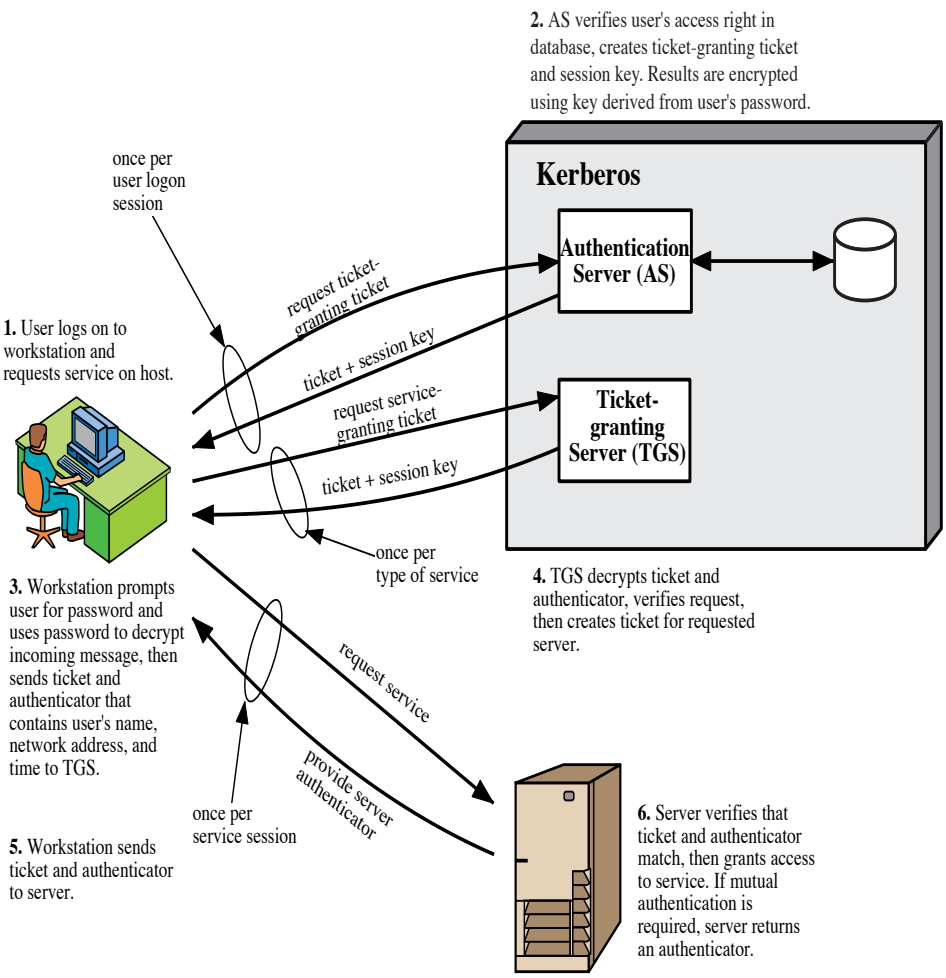

Figure 14.1 Overview of Kerberos

Figure 1: An overview of Kerberos

granting service, and asks for a ticket for the file server. This request is encrypted using the session key K_{ses}. The ticket granting service sends Kartik back another ticket, encrypted with the file server's password, that Kartik's workstation can present to the file server service to request files. The contained ticket also contains Kartik's authenticated username, the expiration time, and the internet address of Kartik's workstation. Kartik's workstation presents this ticket to the file server, which decrypts the ticket using its own password, and grants Kartik access to the file system.

A few points about Kerberos are now in order:

1. Passwords are stored only on the Kerberos server, not on the individual workstations. The user's password is never transmitted on the network– encrypted or otherwise. The Kerberos authentication server is able to authenticate the user's identity because the user knows his own password.

2. The ticket granting service was able to establish Kartik's identity because Kartik's file service ticket request was encrypted using the session key K_{ses}, moreover this message included the ticket granting ticket encrypted with K_{tgs}; the only way Kartik could have obtained this information is by decrypting the original ticket granting ticket obtained from the Kerberos authentication server: to do this Kartik would need to know his own password.

3. The file server service was happy with Kartik's identity since the ticket that it received from Kartik's workstation is encrypted with its own key; this ticket also includes Kartik's username and the IP address of his machine (information that could only be put there by the ticket granting service), and this is good enough for the file server (since the ticket granting service is sure of Kartik's identity).

2 SSL: The Secure Sockets Layer

The world wide web is used for more than just transmitting and navigating static user webpages. Before long, various companies got the idea for using it for financial transactions, such as purchasing merchandise by credit card, online banking, and electronic stock trading. These applications created a demand for secure connections. In 1995, Netscape Communications Corp, introduced a security package called SSL (Secure Sockets Layer) to meet this demand.

SSL is designed to make use of TCP to provide reliable end-to-end secure connection. SSL consists of two subprotocols, one for establishing a secure connection (SSH Handshake Protocol) and one for using it (SSL Record Protocol). Before we proceed, it is useful to distinguish between a connection and a session. A session is an association between a client and a server. Sessions are created by the handshake protocol. A session involves the exchange of cryptographic security parameters, which can be used over multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection. Also, a session consists of multiple connections, each of which uses the record protocol.

We will first discuss the handshake protocol which is used at the beginning of a new session. This protocol is shown in Figure 2. Let us define the two communicating parties as Alice (client) and Bob (server). The protocol begins

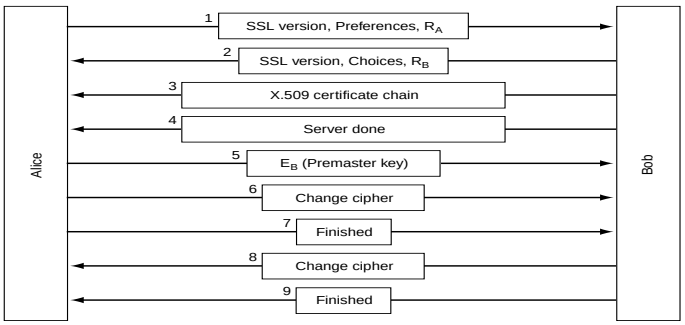

Figure 2: The SSL handshake protocol

with message 1 when Alice sends a request to Bob to establish a connection. The request also specifies the SSL version Alice has and her preferences with respect to compression and cryptographic algorithms. It also contains a nonce (a random number) R_A, to be used later.

In message 2, Bob makes a choice among the various algorithms that Alice can support and sends his own nonce, R_B. Then in message 3, he sends a certificate containing his public key (in SSL it is always important that the server authenticate itself to the client; for example when you buying a book online at amazon.ca this will ensure that you are actually talking to the amazon.ca web server and not to some eavesdropper who masquerades as amazon.ca); moreover just showing your public key to the client is not sufficient since anyone can pull Bob's public key from his webpage (say). So Bob's public key needs to be certified by some well known authority. In this case, Bob's public key is encrypted using the authority's private key; we will assume Alice has access to the public key of the authority and uses this to decrypt Bob's public key. Now what if Alice does not have the authority's public key; in this case Bob sends Alice a chain of X.509 certificates that can be followed back to one. All browsers, including Alice's, come preloaded with about 100 public keys of well known authorities, so if Bob can establish a chain anchored at one of these, Alice will be able to verify Bob's public key. Thus, if Alice has securely obtained the public key of authority X_1, Bob has the public key of authority X_2, and authorities X_1 and X_2 have secured exchanged their public keys, the following procedure will enable Alice to obtain Bob's public key.

1. Alice obtains from her web browser the certificate of X_2 signed by X_1. Since, Alice knows X_1 public key, she can obtain X_2's public key from this certificate.

2. Alice then uses the public key of X_2 on Bob's certificate signed by X_2 to obtain Bob's public key; she can also trust this public key since the authorities X_1 and X_2 are above the law :)

3. The process need to be limited to two authorities; in fact an arbitarily long path of authorities can be followed to produce a chain as descrbed above.

At this point, Bob may send other messages to Alice (such as a request for Alice's public-key certificate; normally servers do not bother to authenticate clients; Alice may well authenticate herself later when she gives her credit card details). When Bob is done, he sends message 4 to tell Alice that it is now her turn.

Alice responds by choosing a random 384 bit *premaster key* and sending it to Bob encrypted with his trusted public key (message 5). The actual session key used for encrypting data is derived from the premaster key combined with both the nonces in a complex way. After message 5 has been received, both Alice and Bob are able to compute the session key.

Alice then computes a hash on a concatenation of the session key and all the handshake messages that have been exchanged so far, and encrypts this with her private key. The data is sent as message 6. This ensures that none of the messages have been computed so far; it also ensures that Bob has computed the correct session key. Alice then sends message 7 indicating that she is finished with the establishment protocol. Bob then acknowledges her (messages 8 and 9).

Thus, we see that the SSL handshake protocol supports multiple cryptographic algorithms. The strongest one uses triple DES with three separate keys for encryption and SHA-1 for message integrity. This combination is relatively slow, so it is mostly used for banking and other applications where the highest security is required. For ordinary e-commerce applications, RC4 (a symmetric-key cryptosystem) is used with a 128 bit key for encryption and MD5 is used for message authentication. In fact, RC4 takes the 128 bit key as a seed and expands it to a much larger number for internal use. The keystream is XORed with the plaintext to produce a classical stream cipher. The export versions also use RC4 with 128-bit keys, but 88 of the bits are actually made public to make the cipher easy to break (all this so that Netscape Communications could actually obtain an export license :)

For the actual transport, a second subprotocol called the SSL record protocol is used. This protocol is illustrated in Figure 3. Messages from the browser are first broken into units of upto 16 KB. If compression is enabled, each unit is then separately compressed. After that, a secret key derived from the two nonces and premaster key in the handshake protocol is concatenated with the compressed text and the result hashed with the agreed-on hashing algorithm (usually MD5). This hash is then appended to each fragment as the MAC. The compressed fragment plus MAC is then encrypted with the agreed-on symmetric encryption algorithm. Finally, a fragment header is added and the fragment is transmitted over the TCP connection.

Supplementary Reading:

1. Chapters 14 of Stallings [1], especially Section 14.1, for a discussion on Kerberos. There is also a nice discussion in Chapter 19 of Garfinkel & Spafford [2].

2. Chapter 17, especially Section 17.2, of Stallings and Section 8.9 of Tanenbaum [3] for a discussion of web security and SSL.

References

[1] W. Stallings, *Cryptography and Network Security*, 3rd edition, Prentice Hall, 2003.

[2] S. Garfinkel and G. Spafford, *Practical Unix & Internet Security*. 2nd edition, O'Reilly & Associates, 1996.

[3] A.S. Tanenbaum, *Computer Networks*, 4th edition, Prentice Hall, 2003.

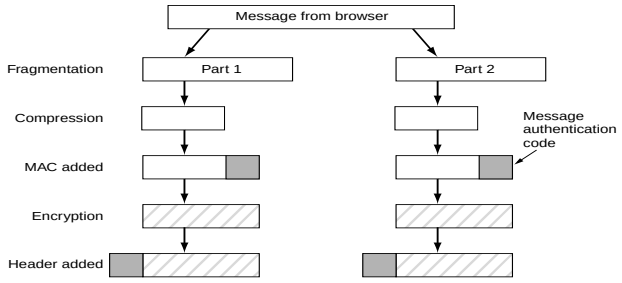

Figure 3: Data transmission using SSL