CS 261 Fall 2007
Nov 15: Kerberos

Lecture by David Wagner, scribed by John Bethencourt

1 Secure Communication / Secure Channels

Let's try to design a secure channel between a client C and a server S. We want confidentiality and integrity, as if they had a physical wire connected to only them, which no one else could interact with.

Assume they share a preestablished secret key K. Okay, so as a first try at achieving a secure channel we can just use K for a MAC and to encrypt. So if C wanted to send the message M to S, they would send

$$\mathrm{C} \to \mathrm{S} : \{[M]\}_K \ .$$

The braces denote encryption under symmetric key K (this is standard notation in cryptographic protocols), and the square brackets denote the application of a MAC using key K (this is our notation).

However, this simplistic approach doesn't quite get us everything we want.

1. Adversaries can see that we are sending messages and could attempt traffic analysis to make inferences about the content.

2. Adversaries can replay messages from C to S.

3. If the same key is used for messages from S to C, an attacker could take a message sent by C and send it back to C, as if S had sent it. This is called a "reflection attack".

4. An adversary can delete messages if they are situated between S and C on the network.

Now let's try to think of ways to fix some of these weaknesses.

1.1 Replay Attack Defenses

There are at least three possibilities:

(a) Add a timestamp. So a message at time t would be

$$\mathrm{C} \rightarrow \mathrm{S} \ : \ \{[M, t]\}_K \ .$$

(b) Add a nonce N, which could be a 128-bit number selected randomly by the server each time the client wants to send a message.

$$\begin{aligned} \mathrm{C} &\rightarrow \mathrm{S} \ : \text{“request to send”} \\ \mathrm{S} &\rightarrow \mathrm{C} \ : N \\ \mathrm{C} &\rightarrow \mathrm{S} \ : \{[M, N]\}_K \end{aligned}$$

(c) Add a sequence number. So the first three messages from C to S would look like

$$\begin{aligned} \mathrm{C} &\rightarrow \mathrm{S} \ : \{[M, 1]\}_K \\ \mathrm{C} &\rightarrow \mathrm{S} \ : \{[M, 2]\}_K \\ \mathrm{C} &\rightarrow \mathrm{S} \ : \{[M, 3]\}_K \ . \end{aligned}$$

The server would have to keep track of the last sequence number seen and reject messages which do not contain the next one. This state would have to be initialized somehow and kept in sync between the client and server.

Options (a) and (b) have the advantage of being stateless, which is nice.

Option (c) could be made more robust to message losses and reorderings by having the server keep track of the largest sequence number seen so far n and a 32-bit vector $b_1, b_2, \ldots b_{32}$. When a message is received with sequence number i, the server rejects it if $i < n - 32$ or if $i = n$. If $n - 32 \leq i < n$ and $b_{n-i} = 1$, it is also rejected. If $n - 32 \leq i < n$ and $b_{n-i} = 0$, b_{n-i} is set to 1 and the message is accepted. Otherwise, $i > n$, in which case the message is accepted, n is set to i, and the bit vector is shifted appropriately (ensuring the new bit corresponding to the nth message is set).

1.2 Reflection Attack Defenses

Just add a bit in the packet which indicates the direction of the message, or a full “to” and “from” address.

1.3 Deletion Attack Defenses

Of course, if an attacker is capable of cutting the wire between C and S, there is no way to prevent that. However, we can use periodic keep alive messages to inform the other party that this hasn't happened yet. Also, if we use sequence numbers, deleted messages will be detected upon receipt of the next non-deleted message.

2 Session Key Establishment

Suppose there is some chance of accidentally leaking the symmetric key used for encryption each time C and S use it for a period of communication. In that case, they may want to establish a new, temporary session key each time they communicate. To do so, C can pick a random session key SK and send it to S over a channel secured by their long term key, K. Then they switch to a channel secured by SK, and when they are done communicating, SK is discarded. If a session key somehow leaks during communication, this technique will limit the damage.

3 Kerberos

Okay, what if many people all want to communicate with one another? Well, each person could share a separate key with each other person. However, this will require a lot of keys in total (about n^2 for n people) and will be difficult to manage.

3.1 The Key Distribution Center

Kerberos attempts to simplify this situation by having a trusted key distribution center KDC with which each person shares a key. When Alice wants to establish a secure channel with Bob, she asks the KDC to pick a session key for them.

One weakness of this approach is that the KDC is a single point of failure. If the KDC gets compromised, the attacker can decrypt all past traffic and spoof and decrypt anything from that point on.

To ask the KDC to pick a session key for her and Bob, Alice would send the following message, where $K_{\text{Alice,KDC}}$ is the key shared between Alice and

the KDC.

$$\text{Alice} \rightarrow \text{KDC} : \{[\text{“I, Alice, want a session with Bob, good for time } t_0\text{.”}]\}_{K_{\text{Alice,KDC}}}$$

In the actual packet sent, this might be condensed to

$$\text{Alice} \rightarrow \text{KDC} : \{[\text{Alice}, \text{KDC}, t_0, \text{Bob}]\}_{K_{\text{Alice,KDC}}} \ .$$

In the cryptographic protocol literature, it is customary to write messages in a condensed format like the above. Next, the KDC would pick a random session key K and reply with the following message, given first in full and next in condensed form.

$$\begin{aligned} \text{KDC} \rightarrow \text{Alice} : \{[&\text{“Key } K \text{ requested by Alice is good} \\ &\text{for Alice and Bob for } t_0, \ldots t_{\text{exp}}\text{.”}]\}_{K_{\text{Alice,KDC}}} \\ \text{KDC} \rightarrow \text{Alice} : \{[&\text{Alice}, \text{KDC}, K, \text{Alice}, \text{Bob}, t_0, t_{\text{exp}}]\}_{K_{\text{Alice,KDC}}} \end{aligned}$$

Similarly, the KDC would send the following message to Bob.

$$\begin{aligned} \text{KDC} \rightarrow \text{Bob} : \{[&\text{“Key } K \text{ requested by Alice is good} \\ &\text{for Alice and Bob for } t_0, \ldots t_{\text{exp}}\text{.”}]\}_{K_{\text{Bob,KDC}}} \\ \text{KDC} \rightarrow \text{Bob} : \{[&\text{Alice}, \text{KDC}, K, \text{Alice}, \text{Bob}, t_0, t_{\text{exp}}]\}_{K_{\text{Bob,KDC}}} \end{aligned}$$

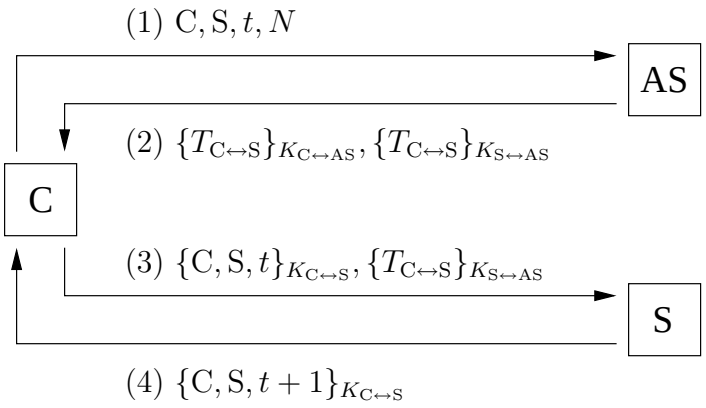
As an optimization, the KDC can instead include this message in its reply to Alice. Alice may then forward it to Bob when she connects to him.

3.2 Basic Kerberos Example

In Figure 1, we show all four messages that would be exchanged when a client C authenticates itself to the authentication server AS and establishes a session key $K_{\text{C}\leftrightarrow\text{S}}$ with a server S (omitting the ticket granting server (TGS)). In the figure, N is a nonce and t is a timestamp.

3.3 Practical Issues with Kerberos

- What is in the trusted computing base? The authentication server and ticket granting server of course. But also the time service, i.e., ntp. Today, most systems use nonces in place of timestamps (at the cost of an additional round trip) to avoid the need for time synchronization.



$$T_{\mathrm{C}\leftrightarrow\mathrm{S}} = (K_{\mathrm{C}\leftrightarrow\mathrm{S}}, \mathrm{C}, \mathrm{S}, \text{ip addr}, t, \text{lifetime}, N)$$

Figure 1: Establishing a session key in Kerberos.

- If you break into the KDC (which, in Kerberos, is split between the AS and TGS), you can decrypt all past and future traffic. There is no forward or backward secrecy. Since everyone has to trust the KDC, Kerberos would normally only be deployed within a single administrative domain.
- The KDC is also a single point of failure from a reliability point of view. If it goes down, you might not be able to print, read your mail, etc. This is easy to mitigate with replication since the AS and TGS are stateless.
- By default, all messages between users and services are authenticated but not encrypted. At the time Kerberos was designed, they felt any unnecessary encryption was too expensive computationally. Now symmetric encryption is almost completely free compared to other costs.
- The original MAC in Kerberos is insecure (it was based on a CRC checksum rather than a cryptographic hash function).
- Kerberos is vulnerable to dictionary attacks. A ticket is encrypted with a user's password, so an attacker can try to crack it offline.

- For a period, the Kerberos implementation used an insecure pseudorandom number generator to generate session keys (it was seeded with the process ID and time of day, which may be easily guessed). It was actually just a placeholder that the developers knew was insecure. They intended to replace it, but someone accidentally checked their code into the wrong repository. Nobody noticed the mistake for years.