6.826 Principles of Computer Systems	Fall Semester, 1992
Lecture 20: November 25, 1992	
Lecturer: Butler Lampson	*Scribe: Frank Ho*

1 Lecture Overview

Today's lecture continues our discussion on authorization schemes used in security systems. The question we shall address is to determine which *principal*, is making the request. In the second half of the lecture, we shall begin a new topic — system management. We shall look at issues of system management and discuss policies for addressing the problems.

2 Authentication Issues

We need to know who is making a request in order to ensure security in a distributed system. It is inadequate to know only about the physical channel which the message comes from; we need to know the principal who generated the message. This is possible if we know *who can speak for whom*, which can be deduced from other knowledge that the system already has. In particular, it will be useful to know that channel C $\Longrightarrow$principal P, that is, if C says S, then P says S. This rule is correct only if P is the sole principal that can send messages on channel C.

In distributed systems, for example, the channels are based on public-key encryption because we can be certain as to who actually sent the message. This is possible because we believe that the encryption algorithm is strong, and that we know who it is that knows the key.

The receiver must make sure that the decryption key K_p that it uses to decrypt messages on C is actually the decryption key for P. The danger is that an adversary A could misinform the receiver by posing its own decryption key K_a as K_p, and then send messages on C using its encryption key K_a^{-1}, which will be accepted by the receiver as coming from P. We discuss below how to establish that $K_p \Longrightarrow$ P.

3 Authentication Model

We could establish that K_{jmw} speaks for *jmw* by either:

1. Physically going to see her and ask her about her key. This is usually inconvenient and maybe infeasible.

2. Asking her to send her key in a message. The problem here is: how can we be sure that the message actually comes from her?

Recall that the Handoff Rule establishes that:

$$If \text{ A}\Longrightarrow\text{B and A says A'}\Longrightarrow\text{B}, \text{ then A'}\Longrightarrow\text{B.}$$

In the single certification scheme, we have a trusted certification authority CA, and it is known that $K_{CA} \Longrightarrow$ CA, and CA $\Longrightarrow$ anyone. Hence if K_{CA} says $K_{jmw} \Longrightarrow$ *jmw* then really $K_{jmw} \Longrightarrow$ *jmw*, which is the same as saying that K_{jmw} is actually the key for *jmw*.

The idea is that if the message can be decrypted with K_{CA}, we can be certain that it comes from the trusted certification server and is therefore reliable. In order to generalize the scheme to make it practical, we need to have more than one certification agency. We want to know which server speaks for which set of entities. This can be done with a decentralized naming scheme:

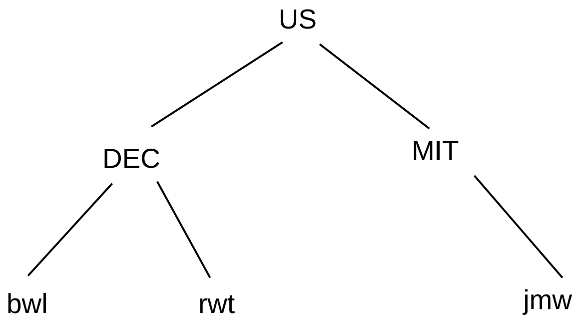

Figure 1: A Decentralized Naming Tree

The question, again, is: how can we believe that $K_{jmw} \Longrightarrow$ *jmw* ?

If we know that MIT speaks for all entities under MIT in the tree, and the root speaks for all entities under it, then it is enough to know only the key of the root. That is,

$$K_{/} \Longrightarrow \text{/*}$$

Now, if:

$$K_{/} \text{ says } K_{MIT} \Longrightarrow \text{/mit/*}$$
$$K_{MIT} \text{ says } K_{jmw} \Longrightarrow \text{/mit/jmw}$$

then we can believe that $K_{jmw} \Longrightarrow$ /mit/jmw.

The problem with this scheme is that we are giving too much power to one entity — the root of the tree. It is more desirable to decentralize the trust, especially if the system is very large. One solution is to trust only those entities that are on the shortest path we can find on the naming tree. It is unnecessary to go up very high on the naming tree to do the authentication. For example, we can start with some known public keys for a set of channels, such as US, MIT, DEC, etc. In order to authenticate /mit/jmw, we only need to go up to the entity MIT. The idea is to avoid trusting something for too much authentication power. A two-way scheme for restricting trust is illustrated by this example:

$$K_{bwl} \Longrightarrow \texttt{/DEC/bwl} \text{ except } \texttt{nil}$$
$$K_{bwl} \text{ says } K_{DEC} \Longrightarrow \texttt{/DEC/*} \text{ except } \texttt{/DEC/bwl}$$
$$K_{DEC} \text{ says } K_{/} \Longrightarrow \texttt{/*} \text{ except } \texttt{/DEC/*}$$
$$K_{/} \text{ says } K_{MIT} \Longrightarrow \texttt{/MIT/*} \text{ except } \texttt{".."}$$
$$K_{MIT} \text{ says } K_{jmw} \Longrightarrow \texttt{/MIT/jmw} \text{ except } \texttt{nil}$$

4 Access Control List

In this model, each object has an *access control list* which contains authorization information related to principals that are allowed to access its resources. A simple-minded method for authentication is to run through the list in order, but this is impractical based on two observations:

1. Public key encryption is slow (1000 times slower than shared key encryption). We therefore need temporary shared keys such as K_{bwl} says $K_S \Longrightarrow K_{bwl}$. (In this example, `bwl` makes up the shared key.) With this indirection method, many levels may be required and the access control list may end up with long chains of information.

2. For any given group, there are many possibilities that a certain name may be associated with. The access control list can be very large.

We therefore need an efficient way to find out what on the access control list to try. One solution is to attach the principal's name (in unencrypted form) to the encrypted message; for example, in the form of a hint that that says that K_{bwl} speaks for `bwl`. The object entity then needs to obtain a certificate that confirms this (*authentication*) and also check with the access control list whether `bwl` is *authorized* to do the operation.

4.1 Push model

In this setup, the sender obtains the certificate and passes it along as part of the request. In such a case, the receiving object need only read the certificate and the access control list. The problem with this approach is that too much burden is put on the sending principal.

4.2 Pull model

In the pull model, the receiver of the request (the object) is responsible for obtaining all necessary authentication information from the certification authority.

The obvious compromise is to use an intermediate approach. It is possible to have the sender supply some information, and the receiver the rest; for example, each can provide certificates for their part of the authentication tree.

5 System Management — An Overview

The remaining part of the lecture concerns the issues of system management. One issue is that when an organization acquires a computer system, it initially establishes and then maintains a small set of information as to who is authorized to use which part of the system. This is then distributed among the users. In practice, however, a large amount of information is initialized and problems can arise. The question is whether something can be done about it.

There are two things we can say about the system at the top level:

1. No matter how perfect the system, it must be told something. For example, security systems require the input of keys and authentication schemes.
2. The primary policy issue concerns resource allocation. Systems are often not properly engineered, and many bugs are nonrecoverable.

In order to get a usable system, we may:

- Build some set of mechanism that uses the basic technology to create a management-free system.
- Let people perform the system management task.

The domain of system management is what is left over when the algorithms have done their best. As a general rule, the more complex the system management scheme, the less management is required by human operators.

6 System Management — Issues

1. **Configuration changes**

   Configuration changes arise from installing and removing parts, and from failures and repairs. There is usually a heavy involvement of people. For example, in a fault tolerant system, we can maintain redundancy for every component of the system. An automatic test can determine if each component is working and calls technical service for help. Most people, however, are not comfortable giving so much autonomy to the system.

2. **Replication**

   Many systems require that data exist in more than one place. An example is a system consisting of many UNIX workstations. Each one has to maintain a password file. A central server may keep the master copy of the file, which is then replicated to each machine. Since the server may crash, it must be replicated. Other examples include software distribution among many workstations, backup for storage components, and filesystems replicated to provide higher availability.

3. **Performance monitoring**

   The general issue here is that we have some expectations as to how the system is behaving, which is derived from our specifications and past experience. We need to build a model for describing how the system behaves, and measure the actual behavior based on the model. We should note that a system may be too complex to measure exactly, as many things can go wrong. It is therefore important that a proper model is used. A correct model also provides the basis for implementing a feedback loop for automating system management.

4. **Dependencies**

   Systems usually have a hierarchy of dependencies. A component may not work unless another component works correctly. Knowledge of dependencies is important in failure analysis and initialization. When a system goes wrong, we may observe some high level components behaving improperly. If we observe a problem with A, it may be caused by B's failure. We also need to know the appropriate order of initialization. The boot server, for example, must work before anything else can work in a system. Specifically, we want to make sure that there are no dependency cycles, and that the load control is known.

7 Access to System Management Parameters

Each component in a distributed system has a set of parameters, which the system manager needs to know in order to control it. Under the Simple Network Management Protocol (SNMP), a table is kept that lists the name of each component and also some standard communication channels to access that component. This is implemented with a Management Information Board (MIB), which is a hierarchical namespace that can be viewed as a board of cells:

Attributes

id1			
id2			
id3			
id4			

Figure 2: The Management Information Board

Two operations, `READ` and `WRITE`, can be used to access information on the table. The system management can make use of these operations to access data for maintaining the system.