Access Control in Amoeba

Attique Dawood

1. INTRODUCTION TO AMOEBA

Amoeba is a distributed operating system developed at Vrije Universiteit, Amsterdam, in the eighties. Chief designer was Andrew S. Tanenbaum. The architecture can be described as a composition of workstations, terminals, processor pool and servers connected through LAN (figure 1). Servers are responsible for file management, printers, login authentication and database etc.

It was designed to be transparent with user not having to worry about underlying hardware, like number of processors. The user only sees interacting with a single powerful system [1]. Another important design goal was to have an *object-based* resource management [2] where each resource is regarded as an object.

2. OBJECTS AND CAPABILITIES

Object-oriented design is a relatively new concept in computing. The *capability-based* systems support object-oriented computing. Following sections give a brief description of objects and capabilities in Amoeba.

2.1 Objects

An *object* is described by its *data members* and *operations*. In amoeba every resource is treated as an object. An object can be any physical or logical entity. Examples of objects are files, memory segment, processes, terminals, message port, I/O devices etc. Objects are assigned *identifiers* by a naming server.

2.2 Capabilities

All objects are described and protected by *capabilities*. A capability is a token, key or ticket which provides access to a resource or object. Unlike conventional systems, capabilities provide *"a single mechanism to address both primary and secondary memories and a single mechanism to address both software and hardware resources."* [3, p. 3]

3. CAPABILITIES AND ACCESS CONTROL

Capabilities are implemented as a data structure having at least two fields, the *object identifier* and *access rights*. In Amoeba, a capability is a 128 bit data structure with four fields as shown in figure 2.

3.1 Capability Fields

The *server port* field identifies the server on which the object is located. A unique identifier *object ID* is used to locate the requested object such as a file or memory segment. The *permissions* field lists all the operations that can be performed on the object. For example, a user has capability for a file with read–only permissions. That file can only be read by the user in question.

Users, processes or procedures, generally referred as *subjects*, possess a *list of capabilities*. This list defines their rights and permissions for accessing resources available to them. It is important to note that capabilities list exists in *user space*. There is a need to protect against forgery. This is accomplished with the *check field* which is a 48 bit random number. This is used together with the permissions field to provide integrity checks against any tampering.

3.2 `filewrite` Example

A user (or process) cannot access an object unless it has the capability for that object in its capability list with suitable permissions. To specify an object, it would provide the *index* for that capability from its list [3, p. 4]. To write a string "message" onto a file he can use a *filesystem* function as,

```
filewrite (file_capability, "string");
```

The `file_capability` provides the name of the file user is trying to access and whether he has *write* permissions. The *fileserver* (where the file resides) would check against the capability provided by the user with its database and allow/disallow access.

3.3 Reduced Capabilities

When a user creates an object (or resource) on a server, it is provided *owner* capability with all the permission bits set. The user can do anything with that particular object including deleting it altogether. Users with owner permissions often want to give reduced access to others [2, ch. 2], e.g. read–only access to a file.

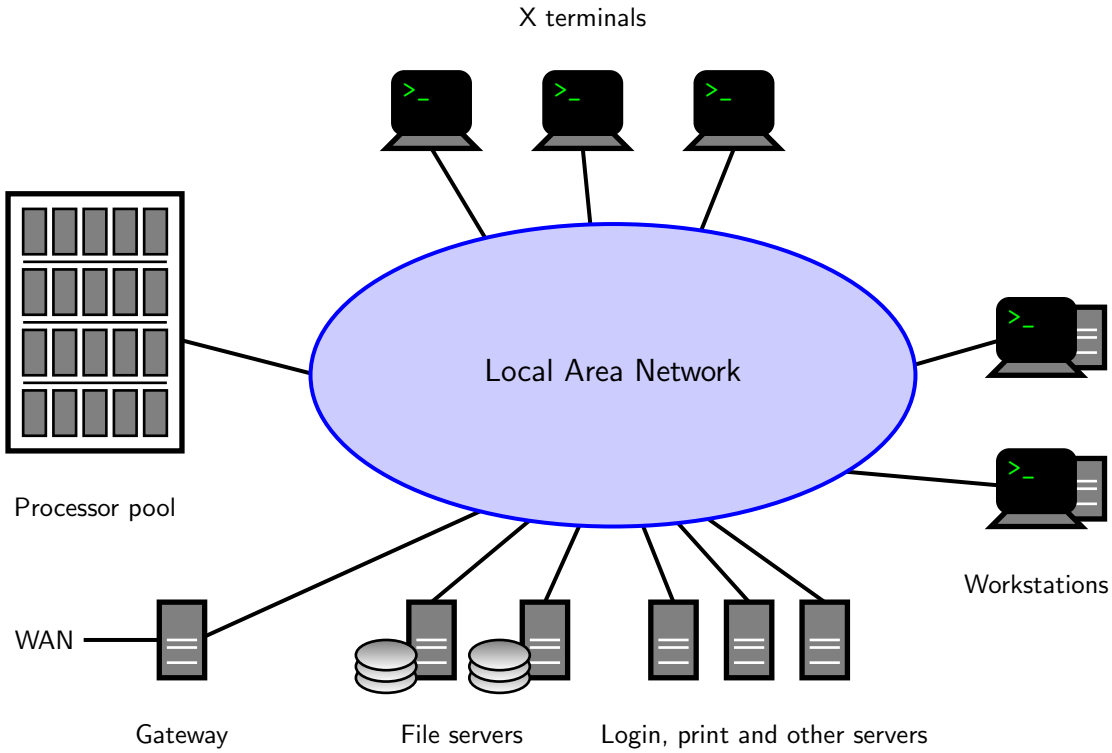

Figure 1: Amoeba architecture.

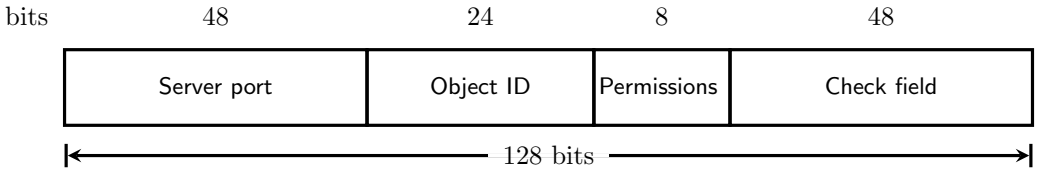

Figure 2: Fields in the capability data structure.

Amoeba provides a one–way function f to compute reduced capability r' from owner's capability check field c. Reduced capability check field is generated by taking $exclusive-OR$ of r' and c and using the result as input to f (figure 3). The resulting reduced capability check field can be expressed as $f(r'\ XOR\ c)$.

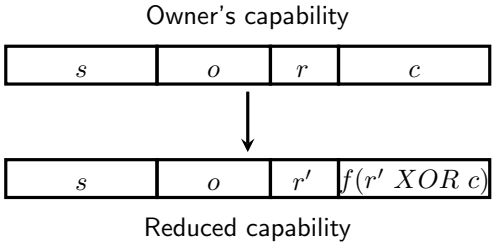

Figure 3: Deriving reduced capability from owner's capability.

4. CAPABILITIES AND ACCESS CONTROL LIST (ACL)

Capabilities–based systems provide object–oriented abstraction and this approach can be applied to many computer design problems. Object–oriented implementation has the inherent advantages of uniformity, data abstraction and information hiding. Maintainability is also much easier as system becomes more sophisticated and size grows. ACLs require authentication at each access compared to capabilities where only one–time authentication is required.

Despite the advantages there are also disadvantages of capabilities. Capabilities cannot prevent man–in–the–middle attacks like eavesdropping and replaying. Compared to ACLs, capabilities cannot be easily retracted [2]. If Alice and Bob have capabilities to access a resource and the owner wants to restrict Bob's access, a new set of capabilities must be issued invalidating the

old capabilities. Alice also needs to be issued the new capability. Also, since capabilities work as keys, Alice can simply give her new capability to Bob and allow access against owner's consent.

5. REFERENCES

[1] `http://fsd-amoeba.sourceforge.net/amoeba.html`.

[2] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems*. Addison Wesley, 2nd ed., 1994.

[3] H. M. Levy, *Capability Based Computer Systems*. Digital Press, 1984.