Right Administration

and

Access Control Mechanism

Gonsu Veronique

1. Introduction

Security is an important aspect of operating system design because it safeguards against access to resources by unauthorised users.
The security mechanism can be broken in two steps : authentication and authorisation. Authentication involves identifying a user, while authorisation ensures that an identified user has access only to resources that has been permitted to use.

Access control mechanism is a mechanism that
1. permits authorized access to a system, such as a communication, computer, and data processing system
2. prevents unauthorized access to the system
3. is considered to have failed when unauthorized access is permitted or when authorized access is prevented.

In the next sections we will talk about the access control, Access Control List (ACL), Capabilities, and about capabilities-based systems like EROS and Amoeba. At the end we will see how the IPC Redirection mechanism can permit a flexible access control.

2. Access Control

Access control is a more general way of talking about controlling access to a resource. Access can be granted or denied based on a wide variety of criteria, such as the network address of the client, the identity of the person who want access, or the browser which the visitor is using.
Access control is analogous to locking the gate at closing time, or only letting people onto the ride who are more than 48 inches tall.
It is important by controlling access to know who has access and who does not have access to a resource. These informations could be stored in an access matrix.

2.1 Access Matrix

The access matrix model is a visualization of access rights, and has several implementations such as access control lists (ACLs) and capabilities. It is used to describe which users have access to what objects. The matrix can be modified only by the owner or the administrator.
The access matrix model has 2 dimensions:

- A list of objects. Objects could be files, processes, or disk drivers
- A list of subjects (processes)

Example of an access matrix with access rights: **r**ead, **w**rite, e**x**ecute, and **d**elete.

	Objects	
Subjects	index.html *file*	Java VM *Virtual Machine*
S1	rwd	x
S2	r	-

The problem with access matrices is that they can become very large. If there are many subjects and objects in a system, there is for each subject-object field combination an entry and some entries remain empty.

As we mentioned above, the two most used implementations of access matrices are access control lists and capabilities.
Access control lists are achieved by placing on each object a list of users and their associated rights to that object.
Capabilities are realized by storing on each subject a list of rights the subject has for every object.
In the next sections, we will examine more closely what access control lists and capabilities are.

3. Access Control List

An access control list (ACL) is a list that tells a to the operating system which access rights each user has to a particular system object, such as a file directory or individual file. The most common rights include the ability to read a file (or all the files in a directory), to write to the file (s), and to execute the file (if it is executable).
Microsoft Windows NT/2000, Novell's NetWare, Digital's OpenVMS, and Unix-based systems are among the operating systems that use access control lists. Each operating system has its own ACL implementation.
For example, in Windows NT/2000, an access control list (ACL) is associated with

each system object. Each ACL has one or more access control entries (ACEs) containing the name of a user or group of users. The user can also be a role name, such as "programmer," or "tester."
Entries in ACLs mustn't be user, but can be a group of users. Each member of the group becomes the rights that are assigned to the group. In most systems that use ACLs, each process has an user-ID and a group-ID. So a user can have two different roles if he is member of two different groups.
By default, the owner of an object, as well as the root user, have access to the object and the authority to change access to the object. A major task in managing access control is the definition of the group membership of users, because these membership determine the users' access rights to the files that they do not own.

3.1 Advantage and disadvantage of ACLs

In access control lists , it is easy to see all subjects that have access rights on an object and it is also easy to revoke access rights.
The disadvantage with ACL's is that the list could be very large. It takes a lot of time to determine for a subject all the objects on which he has access rights. Since one have to read through all ACLs .

3.2 Problems with ACLs

3.3.1 Access Right Delegation by an user who doesn't own the object

Suppose that an user named Fred wants to delegate his access right (e.g. access to a single file that Fred does not own) to an other user. Since Fred doesn't own the object , he cannot delegate access right.
Fred cannot modify the access control list unless he owns the object.

3.3.2 Selective Access Right Delegation

Consider once again the user Fred who runs, this time, a process p1. Fred has access rights on two objects O1 and O2. Since, in access control list systems, all the authorities associated to an user are granted to every program running on behalf of this user, P1 has also access rights on objects O1 and O2.
Supposed now that Fred wishes to create a new process P2, that should have access only to object O1. The new process P2 is created by P1, and inherits the ID of Fred. So the access rights of P2 are identical with those of P1. Consequently, P2 can also access object O2. There is no means to limit these access rights , and thus no possibility to delegate access rights selectively .

In the next section, we will see what capabilities are.

4. Capabilities

The term capability was introduced in 1996 by Dennis and Van Horn.

The basic idea is the following: suppose we design a computer system so that in order to access an object, a program must have a special token. This token designates an object and gives the program the authority to perform a specific set of actions (such as reading or writing) on that object. Such a token is known as a capability.

Systems that use Capabilities are know as "capability systems". In all of these systems, the capabilities are managed by (trusted) kernel software, often with special assistance from the hardware .
Now we will talk about the properties and problems of capabilities.

4.1 Proprieties of Capabilities

Two capabilities can designate the same object but authorize different sets of actions. One program might hold a read-only capability to a file while another holds a read-write capability to the same file.
Capabilities can be delegated, copied, and forged. When a capability is forged, all the capabilities are rescinded . A rescinded capability conveys no authority to do anything at all.

4.1.1 Least Privilege and Selective Access Right Delegation

We consider the same example as in section 3.3.2. Suppose that Fred creates a new process P2. The new process P2 holds no capabilities and therefor no access rights. But the creating process, P1, is given a capability to the new process including take and grant rights.
The take right allows P1 to read a capability from P2 and the grant right allows P1 to write a capability into P2.
Once a new process is created , P1 has the possibility to give it as many or as few capabilities as desired.
So it is possible with capabilities to construct a process that has the least amount of privilege that is necessary to perform their function, and to selectively delegate access rights.

4.2 Problems with Capabilities

The main problem with capabilities is finding a way to save them to disk so that we can retrieve them . This is one of the main reasons that explains why few capabilities-based systems have been built.

Assume a moment that we runs a program which has a capability that allows him to create a file and write to this file. Suppose that we create a file and while we are writing informations to the file, the system shuts down. We start the system up but we have two problems:

1. how to access the file ? we need a capability to access the file system, but where do the first few programs get their capabilities from ?
2. by which authority do they hold them?

These problems are solved in different way in persistent system and non-persistent system.
In persistent system, processes do not go away until they exit voluntary, and the runtime state ist not lost with system shutdown. This is not the case in non-persistent system, where applications die when the system crashes, and any informations that the applications have not explicitly saved are lost.
In section 5 and 6, we will see two *capability systems*: EROS and Amoeba.

5. Eros

EROS is a new capabilities-based operating system that is persistent and originally implemented at the University of Pennsylvania.
EROS merges some very old ideas in operating systems with some newer ideas about performance and resource management. The result is a small, secure, real-time operating system.
To solve the problems mentioned in section 4.2 , EROS provides a mechanism called *checkpointing.* The programms do not die until they are told to. Once an application is started, it will continue to run until it is canceled.
Bevor we examine the *checkpointing* technique , we will talk , in the next section, about capabilities in EROS.

5.1 Capabilities in EROS

5.1.1 Invocation

To exercise the object or service designated by a capability, the holding process *invokes* that capability. The object is invoked, performs the requested servive, and typically replies to the caller .

I n EROS, capability invocation is the ony means of accessing objects . To enforce object protection, it is therefore sufficient to ensure that a process does not possess (and cannot obtain) a capability conveying inappropriate access rights to the object protected.
Capabilily may be transferred, but only between applications that hold capabilities to each other or to some commonly accessible object capable of holding capabilities.

5.1.2 Protection

The protection of capabilities is accomplished by partitioning capabilities and data into separate spaces. Just as there are operations for manipulating data, there are operations for manipulating capabilities. The system provides a service for copying the representation of a capabilty into data space. The authority to use this service is not widely available.

5.2 Checkpointing

Checkpointing is a technique that consists to create consistent snapshots of everything that is happening (running processes,...) and to whrite them down. The entire state of every process on the machine is stored. The snapshots are taked every 5 minutes and it requires, depending on systems, 100 ms. While the snapshot are being taken , all the applications are stopped.

System recovery is fast. When the power cord is reinserted in the outlet, EROS is up and running in 30 seconds or less , with all applications intact.

5.3 Implications of Checkpointing

With checkpointing many applications do not need to write data into a file. Since the applications don't die there is no need to store informations somewhere else.

6. Amoeba

Amoeba is an object-oriented distributed and non-persistent capability-based operating system. It uses capabilities for naming and protecting objects. Objects are specified by capabilities.

Operations are executed by having exchanges of messages between processes. The messages are generally in the form of request from a client followed later by a reply from a server.

To create an object, a client send a request to the appropriate server specifying that it wants to create an object. The server then creates the object and returns a capability tho the client. On subsequent operations, the client must present the capabilty to identify the object.

6.1 Capabilities in Amoeba

We have mentioned in section 4.2 the main problem with capabilities. In order to solve this problem in Amoeba , some arrangement must be made to store capabilities in a permanent store. To prevent users from forging new capabilities or tampering with existing ones, capabilities are protected cryptographically.
Encrypted capabilities are stored as normal data.

A capability is a long binary number and has following format in Amoeba 5.2:

1. The put-port of the server that manages the object
2. An object number meaningful only to the server managing the object
3. A rights field , for each permitted operation the bit is set to 1
4. A random number, for protecting each object

Server Port	Object	Rights	Check

To perform an operation on an object, the client calls a stub procedure that builds a message containing the object's capability and then sends it to the kernel.

The kernel extracts the *Server port* field to identify the machine on which the server resides and then sends the rest of informations in the capability to the server.
The server would use the *Object* field as an index into its table to locate the object. For UNIX like file server, the object number would be the i-number, which could be used to locate the i-node.
The *Rights* field is a bit map that tells which operations the holder of the capability are allowed to perform. For allowed operations the bit is set to 1.
The *Check* field is used for validating the capability . It prevents user processes from forging capabilities.

6.2 Object protection

There are several object protection systems. In the basic model , the server merely compares the random number in its table (put there by the server when the object was created) to the one contained in the capability. If they are the same , the capability is assumed to be valid , and all operations are permitted. The problem with this model is that the system doesn't distinguish between READ, WRITE, DELETE and other operations . The reason is the following: since all rights bits are initially on in a new capability, all capabilities that are returned to the client have all rights bits on.

To create a restricted capability, a client return a new capability to the sever, along with a bit mask for the new rights (subset of the rights in the capability). The server takes the original *Check* field from its tables , EXCLUSIVE-ORs it with the new rights , and then uses it as an argument of the one-way function *F*. The resulting value is put into the *Check* field of the capability and returned to the client.
Formally:
Check field = *F*(random number XOR rights bits)

When the restricted capability arrives at the server, the server finds the original random number from its tables and EXCLUSIVE-ORs it with the *Rights* filed from the capability, passing this result through *F*. If the result agrees with the *Check* field the capability is considered valid.

After we saw how access control is accomplished using capabilities and access

control lists , we will now talk about a mechanism, that enables flexible access control using IPC redirection.

7. Flexible Access Control using IPC Redirection Mechanism

The inter-process communication (IPC) redirection mechanism is a mechanism that enables efficient and flexible access control for micro-kernel systems. This mechanism is used in L4.

In micro-kernel systems, IPC is the only means of communication between services. Thus , it is important for the system to use IPC to enforce its access control. Trusted processes, called **reference monitor**, are assigned the task of enforcing the system's access control.
The IPC redirection mechanism removes the management of access control from the kernel, enabling so the implementation of a variety of system-specific mechanisms.

The next sections are organized as follows. In section 5.1, we define what a reference monitor is. In section 5.2, we present the mechanism, and in section 5.3, we describe how this mechanism can be used to implement reference monitor.

7.1 Reference Monitor

A reference monitor is a trusted user-process that has the task to decide which process under the system's security policy has access to an object. It controls any communication channel and enforce the system's access control.
The reference Monitor can :

1. block IPCs between 2 processes
2. revoke capabilities when changes occur in access control policy
3. prevents a process to use a capability that has been delegated to an other process

7.2 IPC Redirection Mechanism

As we mentioned above, the kernel doesn't manage the redirections policies but it controls the mechanism. The kernel implements IPCs that are redirected depending on the policy specified by an user-process called **redirection controller**.

The redirection controller can set (also arbitrary) redirection policy for processes in its redirection set, the set of processes for which it may set redirection policy.

7.2.1 Redirection Function

We define a redirection function R as follows:

s,d,i are processes and P is the set of all processes
R(s,d) -> i , for s,d,i in P

The function R maps an IPC source process (s) and destination process (d) to an interim destination (i). For example, setting R(s,d) = d allows direct IPC from source to destination. Setting R(s,d) = i redirects the source IPC to an interim destination i (unless i = s).

7.2.2 Role of the Interim Destination

The interim destination can block, revise, or forward IPCs between source and destination .
The destination process needs to know the identity of the original IPC source in order to authorize operations. The interim process may specify the identity of the original IPC source. And the kernel provides :

1. a valid identity of the last interim destination
2. a restriction of the set of sources that can be claimed by an interim destination (preventing the interims destination from claiming unauthorized sources)

If there are multiple interim destinations between source and destination, each interim destination appends its identity to the chain of destinations. This aids the final destination in authentification.

7.3 The protocol

The protocol works as follows: Given the identity of the source and the destination processes, the kernel attempt to retrieve the redirection data entry from its table (e.g a hash table). If there is no redirection entry for the combination of source and destination , then the kernel has a *redirection fault*, upon which it forwards the IPC to the redirection controller of the source. In this case the redirection controller acts like an interim destination. Additionally , it can set one or more entries of redirection data to redirect future IPC to the appropriate destination.

But if there is a redirection entry, the kernel redirects the IPC to the interim destination that can block, revise, or forward the IPC. Any returned IPC from the destination process is redirected by the kernel to the interim destination of the destination. The interim destination of the destination process could be the same one as the interim destination of the source process.

7.4 Example with Reference Monitor

Consider a redirection controller (RC) (Fig 1.) and its redirection set containing

processes s, and the reference monitor (M). RC redirects all IPCs from s to any destination d to M. The redirection function is R(s,_) = M. The reference monitor performs all access control checks for these processes and may sends IPCs directly to destination (R(M,_) = d).

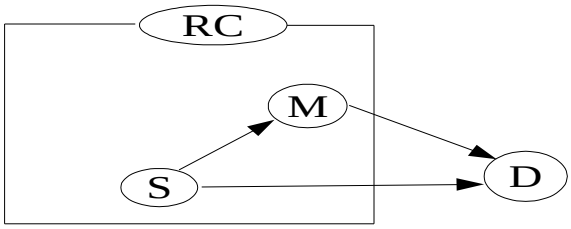

Figure 1. Process M monitors opeations by its processes
as specified by RC

Any returned IPC from d is automatically redirected to its reference monitor that could be M or another process.
The reference monitor acts as an interim destination and enforces the system's security policy . Additionally, different reference monitors may implement efficiently different mechanisms to enforce the security requirements of their processes .

7.5 Conclusion

We can conclude that the IPC redirection mechanism enables significant flexibility in the enforcement of the system's security policy. The authorization mechanism may be customized to the security requirement of the monitored process, and the reference monitor for a process can be changed and also re-established when security requirements demand it.

8. Clans & Chiefs

Clans and Chiefs are introduced as a basic concept of an operating system. Its idea

was influenced by the subject restriction concept of Birlix. In this concept , it is possible to block up suspicious active entities (subjects) by means of subject restriction lists. Each of these lists specifies the set of all partners (mostly servers), which are accessible by the corresponding suspicious subject. The clan concept allows full algorithmic control of process interaction in a user definable but secure way.

Clans & Chiefs can be used for protection, remote communication, debugging, event racing, emulation, connecting heterogeneous systems and even for process migration.

8.1 Definition

A clan is a set of tasks headed by a chief task. All messages from clan members to tasks outside the clan are redirected by the kernel to the chief. The same happens with incoming messages. Thus, the chief inspects all communication of the clan with the outer world. The clans may be nested such that a chief may belong to a clan of another chief.

The mechanism enables not only the protection of the outer world against suspicious subjects, but also the protection of a clan against the outer world.

Clans & Chiefs can be used to define The IPC redirection mechanism mentioned in section 7.

8.2 IPC Redirection Mechanism

In Figure 2, M1, M2, M3 are chiefs. RC is the chief of M1, M2, and M3. Each chief has its clan and controls the flow of IPCs to and from a single process. To simulate the Clans & Chiefs semantics, RC sets the redirection functions as follows:

- P1: R(P1,P2) = M1, R(P1,P3) = M1
- P2: R(P2,P1) = M2, R(P2,P3) = M2
- P3: R(P3,P1) = M3, R(P3,P2) = M3

Similarly, the chiefs IPC are redirected:

- M1: R(M1,P1) = P1, R(M1,P2) = M2, R(M1,P3) = M3
- M2: R(M2,P1) = M1, R(M2,P2) = P2, R(M2,P3) = M3
- M3: R(M3,P1) = M1, R(M3,P2) = M2, R(M3,P3) = P3

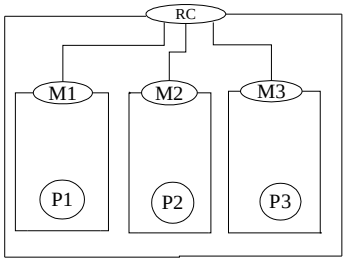

Figure 2. Simulation of Clans & Chiefs with 3 clans

If new process is entered into a clan, its redirection functions are set according to the clan that it is entered .

9. Summary

The access matrix model is a visualization of access rights. The problem with access matrices is that they can become very large. A possibility to solve this problem is to use access control lists or capabilities.
Capability systems can support basic properties that access control list (ACL) based systems do not. In spite of everything ACL based systems have been more used than capability systems. The reasons of that are historical and due to the difficulty to store capabilities as we have mentioned in section 4.2.
IPC Redirection mechanism , used in L4, is a mechanism that enables flexible access control. The Clans & Chief concept can be used to define this mechanism.

References

[1] J. Liedtke: “Clans & Chiefs” In 12. GI/ITG-Fachtagung Architektur von Rechensystemen , page 294-305, Kiel, March 1992 Springer

[2] T. Jaeger, K. Elphinstone, J. Liedtke , V. Panteleenko and Y. Park:”Flexible Access Control Using IPC Rediretion” In 7th Workshop on Hot Topics in Operatings Systems (HotOS-VII), Rio Rico, Arizona, USA , 1999

[3] Essays on Capabilities and Security

[4]J. S. Shapiro: “EROS: A Capability System”, PhD thesis, University of Pennsylvania, April 1999.

[5] A. S. Tanenbaum, S. J. Mullender, R. van Renesse: “Using Sparse capabilities in a Distributed Operating System “ Proceedings of the Sixth International Conf on Distributed Computing Systems , IEEE, pp. 558-563, 1996.

[6] Tanenbaum , Woodhull: “Operatings Systems – Design and Implementation “, 2nd edition, Prentice Hall, 1997