6.915 Computer and Network Security	Fall Term, 1995

Lecture 11 : October 17, 1995

Lecturer: Ron Rivest *Scribe: Yang-hua Chu*

Today's lecture is on how to share information securely among users in a shared computer system.

Depending on who owns the right to change security policy, there are two popular mechanisms to achieve secure information sharing:

- discretionary access control: owner sets the policy
- mandatory access control: system reinforces the policy

Access control list and capability are two popular ways to implement discretionary access control. They are described in detail in this lecture.

Due to time constraint, mandatory access control is discussed in the next lecture.

1 Goals on Secure Information Sharing

In a single-user personal computer, there is no need to have a protection mechanism on information sharing. Owner (or whoever has the control over the computer) has the right to read, write, execute, and administer the data in the computer. There may be hardware or software locking mechanisms for the whole storage device, (using password or smartcard), but individual files and data are not meant to be shared. So we will confine our discussion on multi-user computer system.

To start our discussion, we need a language to talk about security policy, and then we can define our goals on information sharing with clarity.

1.1 Defining the Language

Subjects: are entity in a computer system where authorizations are granted. Mostly they are referred to as users or processes, identified by their login names or unique identifiers with some form of authentication.

Objects: are resources that subjects can share, such as files, segments, I/O devices, data structures, or memory.

Access: are rights a subject can exercise on an object. Common access rights include read, write, execute, modify, delete, and administer. Other data-structure specific rights are, as an example, push and pop for stacks.

Other: such as groups, compartments, labels, time, console vs. remote access.

1.2 Design Principles

Saltzer and Schroeder on the paper "The Protection of Information in Computer Science" (the assigned reading for today's lecture) talked about several design principles that applied particularly to protection mechanism in a shared system. They are described below:

1. **Economy of mechanism:** The design of protection system should be kept small and simple. Simplicity generally means faster running time and less implementation errors.

2. **Fail-safe defaults:** Default access should be set to no access. Saltzer argued that implementation bugs can be detected more easily because users would complain if their access are denied.

3. **Complete mediation:** Every access to every object must be checked for authority. The system should not flip coins to decide whether a mediation should occur.

4. **Open design:** The design should not be secretive, so that it permits the mechanism to be examined by many reviewers. However, intruders may take advantage of the published but not yet corrected security holes to gain unauthorized access. A closed design gives more hardness for intruders to learn the underlying protection mechanism, but it may result in serious logical or implementation bugs that can go unnoticed with a small number of implementators and reviewers.

5. **Separation of privilege:** Like a bank safe-deposit box, the protection mechanism should require two keys to gain access of an object. This principle is generally used for critical data such as missile launch system and registrar office (which stores my freshman grades).

6. **Least privilege:** Each subject should operate using the least set of privileges necessary to complete the job. It prevents possible Trojan Horse attacks.

7. **Least common mechanism:** We want to minimize the common code shared by all users, therefore infecting one shared code does not propagate the effects to all others. Example of common code is library calls.

8. **Psychological acceptability:** Like any other software packages, it is essential that human interface is intuitive and easy to use. It is conceivable that users may make crucial mistakes that make their love letter to be world readable. Users may eventually ignore the security features on information sharing which could be another potential problem.

9. **Work factor:** When designing the system, we should compare the cost of circumventing the mechanism with the resources of a typical attacker. A military site needs a better protection mechanism than Athena environment because the intruders are more powerful and the penalty of information leak is far greater.

10. **Detect and prevent compromise:** The design should make intrusion detection easy and tamper-proof. If possible, there should also be a backup system, such that when damages caused by intrusions is discovered, the system is capable of undoing most of them.

Of course, there are other design principles suitable for specific situations. It is good to use risk analysis to determine whether to carry out the desired principles.

1.3 Policy Ownership

If the policy ownership of an object is the creator, then it is called **discretionary access control**. If the system reinforces the policy, it is called **mandatory access control**. Of course, we can combine both mechanisms to achieve higher degree of information protection and sharing.

There are two popular ways to achieve discretionary access control, namely access control list and capabilities. We will discuss each in detail.

(Possible research topics) Can you come out with different secure information protection mechanisms?

1.4 Other Issues

When creating an object, we need to decide what type of access should be given. Should we assign the access right the same as its parent object (inheritance)? Should we grant the creator all rights? These issues needs to be thought carefully when designing a secure information protection mechanism.

Rights are constantly granted and revoked. A good design should make changing of subjects' rights easy and efficient.

Spawned program generally needs some access inherited by the user to do things useful. We should be careful to follow the principle of least privilege to grant programs enough privileges to complete its mission. There are other programs that require privileges unavailable by the user, such as changing user's password. The mechanism should be able to cope with those exceptions with great caution, as possible Trojan Horse attack may be used by intruders. Two useful keywords to remember: **encapsulation** is to prevent programs from abusing user's privilege (modifying unintended data) and **confinement** is to prevent programs from disclosing user's data.

2 Access Control List

Access control list (ACL) is a list associated with an object, describing which subjects have what access rights. We can represent ACL in a **access matrix** [1] shown below:

subject\object	File 1	File 2	File 3	...
Jason	orwx	rw	x	...
Joe	-	orwx	x	...
John	-	r	orwx	...
...				

Each column of the access matrix represents the ACL of the object. For example, File 2 on the second column, user Jason is allowed to read and write, Joe is allowed all rights, and John is allowed to read only. As we will see later, each row in the access matrix represents a list of **capabilities** for the given user.

[1] In the matrix, o=own, r=read, w=write, and x=execute

Generally the subject who creates the object becomes the owner. The owner can then change the content of the ACL for his or her own needs. When another subject requests access to the object, the operating system searches the ACL to see if the right is granted for the user to perform the action.

2.1 Groups

Group is a collection of subjects. The idea comes from the fact that subjects that shares common objects are generally identified as a group of people, such as all members in the accounting department. When a file needs to be shared by the accounting department, we have to type out all the names in the accounting department. Furthermore, if a member in the department leaves, the ACL of all share objects need to modified, which is quite troublesome. Hence the notion of groups comes handy to specify in ACL.

An ACL with group is a list of (subject, group, access). The list is ordered according to the subject, from specific to general (default is placed last). First match of the query determines the access right. Doing so allows individual access to be specified differently from his or her group access.

Example: the list (John, accounting, orw), (Joe, accounting, r), (*, accounting, rw), (*, audit, r), (*, *, -) means that John has owner right. Everyone except for Joe has read and write access. Joe the trouble-maker is allowed read access only. Audit department has read access, and all other people has no access to that object.

There is an issue as to whether a subject should be able to belong to more than one group at the same time. If a subject can belong to one group at a time, it would be hard to run a cross-group program such as auditing. However, if a subject can belong to more than one group at a time, it may result in Trojan House due to the violation of least privilege principle. System V UNIX allows an subject to belong to only one group at a time, while Standard UNIX allows multiple groups.

2.2 UNIX

In UNIX files are stored with the name of owner and group. The ACL contains the list of length three: user, group, and others. Owner can specify read, write, and execute access. Moreover, there are two extra bits for the spawned program:

- **SUID:** spawned program inherits the right of the user

- **SGID:** spawned program inherits the right of the group

For an object that is a directory, the read and execute access are different. The read access allows you to "ls" the content. The execute access allows you to find a file only if you already know the name.

2.3 Multics

In Multics the basic building block of a storage media is **segment**. Each segment is associated with an ACL. The ACL is an open-ended list of names of users who are permitted to reference the segment. Each principle identifier in the ACL contains three parts:

- **Subject:** Individual user, the same as defined previously.
- **Project:** Similar to group in UNIX, it is a set of users who cooperate and share common files.
- **Compartments:** It places subjects in named projects. The user designates which compartment he or she wishes to be in at the time of authentication. Compartments are useful against Trojan Horse. As an example, user John may set two compartments, one for testing use and the other for private use. To run an unaudited, borrowed program made by Eve, he can operate it in the testing compartment so it does not affect the data in his private compartment.

3 Capabilities

Capabilities, also called protected descriptors or tickets, is a computationally unforgeable ticket, when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket. Capabilities are generated by the operating system upon subject's request with proper authentication. The subject then maintains a list of capabilities he or she owns. The following information is held in each capability:

- addressing information
- access rights (read, write, execute, own)

- transfer rights

In compliance with least privilege, an user grants the smallest subset of the capabilities to the spawned sub-process in order to complete its work. We need to have a mechanism to let the operating system to authenticate the sub-process as part of the original user, so the capabilities are not abused.

3.1 Protecting Capabilities

Capabilities presents possible security holes if they are not secure against forgery. There are various ways to protect capabilities against these attacks:

- **Cryptographic technique:** We can use a MAC (message authentication code) . The operating system acts both as the ticket granter and a guard. It issues the ticket to the user to access objects, and later checks the ticket it upon receiving to ensure no forgery. As long as the MAC is computationally safe, we are safe against forgery.
- hardware protection: This implementation requires one extra bit per word. When capability is created, the bit is set to 1, and any further modification of the location sets that bit to 0, and hence invalidate the capability. It is commonly called tagged architecture.
- protected supervisor space: Capabilities are stored only in the supervisor space which is neither readable nor writable by users. User knows only the index to the table, and the operating system which can access the supervisor space, is able to retrieve the capabilities based on the index given by the users. This ensures that users cannot change the capabilities to gain unauthorized access.

4 Handouts and Readings

There is one handout (H17): Reading for Lecture 11. It contains two papers. One is "The Protection of Information in Computer Systems" by Saltzer and Schroeder. The other is "Protection and the Control of Information Sharing in Multics" by Saltzer.

The assigned reading for today's lecture is section 6.3 and 6.4 in the textbook.