Capabilities

A little reminder about Access Control

- Avoids an access
- Limits an access
- Gives an access
- Take back an access

A program accesses to datas, the user doesn't

- It can be run by different users and won't be able to do the same thing
- It can do stuff that the user can't
- A person can be several users (admin/ normal account)
- User doesn't know what programs he runs (plugins...)

Usual Unix rights

- User: defined by a name + password
- Groups: a user can be in several groups
- Everyone else
- R/W/X defined for those 3 entities for files only
- SUID bit

Problems due to that design

- No possible extension
- Link between user and file
- Access is granted to more than needed
- SUID is dangerous

Example

- I need someone that I trust to keep my house
- In normal life I'd give him the key of my house.
- How would I do that under UNIX?

Solutions

- Main models:
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
  - Rule-Based Access Control (RBAC)

MAC

- Mandatory Access Control defined as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity.

DAC

- Discretionary Access Control defined as “a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by MAC)."

RBAC

- Introduction of rules
- Officially no link between user and files or objects
- MAC + DAC (proved in the 90's)

Implementation

- ACL
  - List of permission link to an object
  - No extension
  - Mainly for file access
  - used since NT, option in Linux,SeLinux (MAC)

- RBAC
  - Rules definition
    - Cannot be used in a kernel because one object = one rule
    - Used in RSBAC implementation (stable version since january 2000)

Capability

- Perfect for distributed system
- Really extensible
- No direct link between user and objects
- Collaboration is possible (you can run my program without seeing my code and I can't retrieve information from you)
- Restriction: a user can access internet but the programs he launches can't
- Goodbye suid

Problems

- Implementation mainly for micro kernel
- Storage of capabilities when computers shutdowns (can be solved by universal persistence or ACL)
- Revocation is hard

History

- 1966
- Dennis and Van Horn in “Programming Semantic for Multiprogrammed Computations”
- No need for it because at first UNIX was more of a cooperative system
- No micro kernels