Protection in Operating Systems

Reading: Ch. 5, van Oorschot

Furkan Alaca

University of Toronto Mississauga

CSC347H5, Fall 2018

Access Control Definition

Access Control

"Access control implements a security policy that specifies who or what (e.g., in the case of a process) may have access to each specific system resource and the type of access that is permitted in each instance."

(Stallings & Brown)

Access Control Context

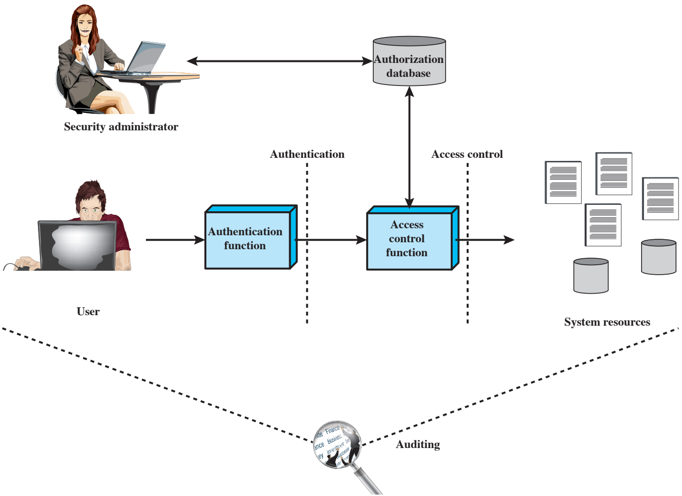


Figure 4.1 Relationship Among Access Control and Other Security Functions

Terminology

Subject

An entity capable of accessing objects. Typically three classes: **owner**, **group**, and **world** (UNIX: **user**, **group**, **others**).

Object

A resource to which access is controlled, e.g., files, directories, memory segments, devices, databases, communication ports, processors.

Access Right

Describes the way in which a subject may access an object, e.g., read, write, execute, delete, create, search.

Access Control Policies

Discretionary Access Control (DAC)

Controls access based on the identity of the requester and the access rules stating what requesters are allowed to do. Termed **discretionary**, since an entity may be permitted to enable another entity to access some resource.

Mandatory Access Control (MAC)

Controls access based on comparing objects' security labels with subjects' security clearances. Termed **mandatory**, because an entity that has clearance to access a resource may not, just by its own volition, enable another entity to access that resource.

(See: SELinux and AppArmor on Linux, Mandatory Integrity Control on Windows)

Access Control Policies (2)

Role-Based Access Control (RBAC)

Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.

Attribute-Based Access Control (ABAC)

Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions.

Discretionary Access Control

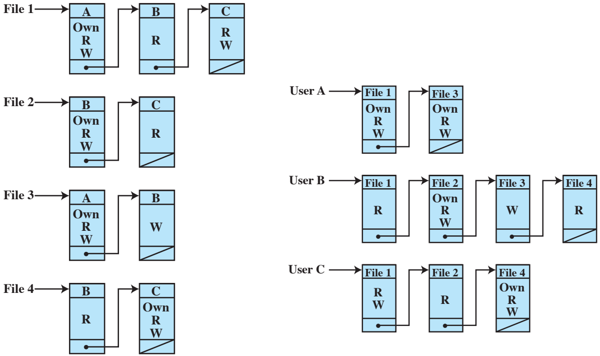
Access Matrices

- In DAC, access rights can be stored using an **access matrix**, where one dimension identifies subjects and the other identifies objects
- Fast and easy lookup, but the matrices are typically sparse and can get very large

		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

- **Access Control Lists** (ACLs) can be obtained by decomposing the access matrix by columns: object-centric approach
- **Capability tickets** can be obtained by decomposing the access matrix by rows: subject-centric approach



(b) Access control lists for files of part (a)

(c) Capability lists for files of part (a)

Discretionary Access Control
Authorization Tables

- Each row of the **authorization table** represents one access right of one subject to one resource
- Can be implemented using a relational database
- Sorting by object makes it similar to an ACL
- Sorting by subject makes it similar to a capability list

Table 4.1 Authorization Table for Files in Figure 4.2

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

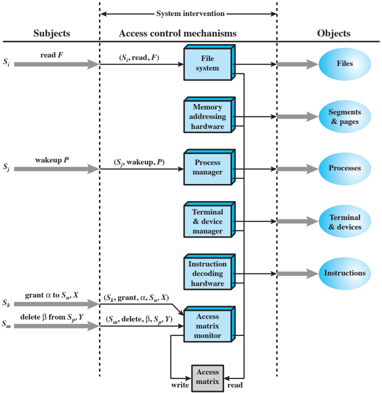


Figure 4.4 An Organization of the Access Control Function

OBJECTS

		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

* - copy flag set

Figure 4.3 Extended Access Control Matrix

Table 4.2 Access Control System Commands

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\left\{\begin{matrix}\alpha^* \\ \alpha\end{matrix}\right\}$ **to** S, X	'α*' in $A[S_o, X]$	store $\left\{\begin{matrix}\alpha^* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R2	**grant** $\left\{\begin{matrix}\alpha^* \\ \alpha\end{matrix}\right\}$ **to** S, X	'owner' in $A[S_o, X]$	store $\left\{\begin{matrix}\alpha^* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$

Discretionary Access Control

Protection Domains

- The model discussed so far associates a set of capabilities with a user
- More general approach would be to associate capabilities with **protection domains**
- Users can spawn processes under a protection domain, which has a subset of the access rights of the user
  - Could be even more granular: Individual procedures within a single process
- Practical example used by many operating systems: execution in **user mode** vs. **kernel mode**

- UNIX files are administered by the OS using **inodes** (index nodes)
- A directory is simply a file that contains a list of file names and pointers to their associated inodes
- An inode is a control structure that contains key information needed by the OS for a particular file, including its permissions and other control information
- On the disk, there is an inode table or inode list that contains the inodes of all the files in the file system

Discretionary Access Control

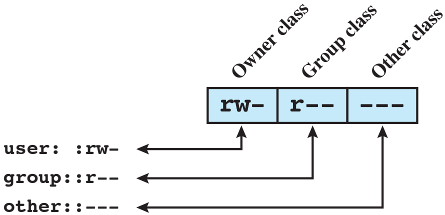
UNIX File Access Control (2)

- Each UNIX user is assigned a unique user ID (UID), and is a member of a primary group and possibly other groups, each identified by a group ID (GID)
- When a new file is created, it is designated as owned by the UID of its creator
- A new file's group is designated as the creator's primary GID

Discretionary Access Control

UNIX File Access Control (3)

- The file's owner ID and group ID, together with 12 protection bits, are stored in the inode
- The first 9 bits specify read, write, and execute permissions for:
  - the owner of the file
  - other members of the group to which the file belongs
  - all other users



(a) Traditional UNIX approach (minimal access control list)

Discretionary Access Control

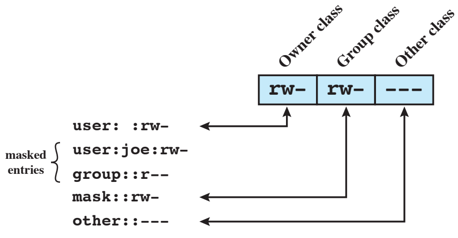
UNIX File Access Control (4)

- The SetUID and SetGID bits allow the system to grant the user executing the file to temporarily assume the rights of the owner UID or GID during execution
- The SetGID permission applied to a directory causes newly created files to inherit the directory's GID
- The sticky bit, when applied to a directory, allows each file in the directory to only be renamed, moved, or deleted by its owner
- The "superuser" is exempt from the usual file access constraints

Discretionary Access Control

UNIX Access Control Lists

- The traditional UNIX approach becomes cumbersome if there are a large number of different groupings of users requiring a range of access rights to different files
- ACLs allow permissions to be granted to other named users or groups
- The group permissions specify the permissions for the owner group for the file, but also function as a mask for permissions granted to the other named users and groups



(b) Extended access control list

Role-Based Access Control

- RBAC systems assign access rights to roles instead of individual users
- Users are assigned to different roles, either statically or dynamically, according to their responsibilities
- A single user may be assigned multiple roles
- Multiple users may be assigned to a single role

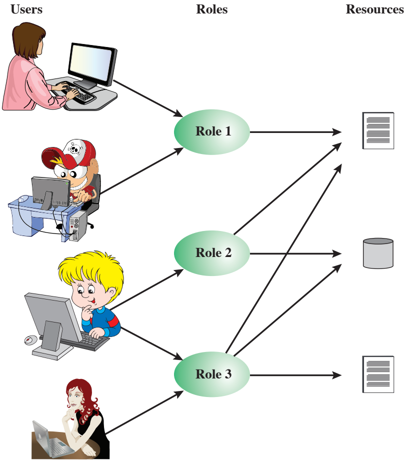


Figure 4.6 Users, Roles, and Resources

Role-Based Access Control

Access Control Matrix Representation

- Can be represented as an access control matrix and a table of user-to-role mappings
- Each role should contain the minimum set of access rights needed for that role
- A user can initiate a **session** with only the roles needed for a particular task

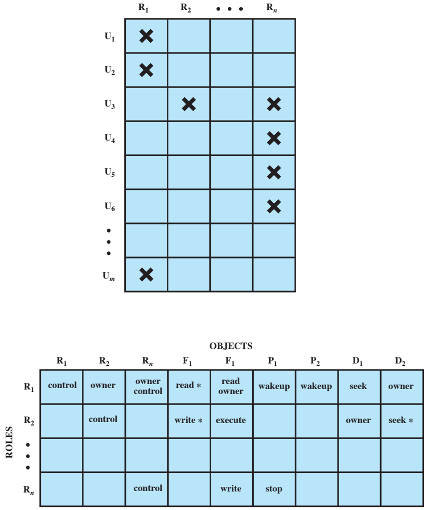


Figure 4.7 Access Control Matrix Representation of RBAC

Role-Based Access Control

Role Hierarchies

- Superior job functions **inherit** access rights from subordinate roles
- One role can inherit access rights from multiple subordinate roles

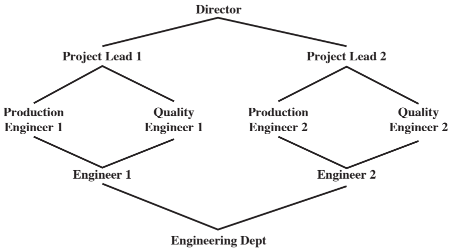


Figure 4.9 Example of Role Hierarchy

- **Mutually exclusive roles** and mutually exclusive permission assignments can be used to achieve the principle of **separation of privilege**
- **Cardinality** constraints can enforce a maximum number of roles which a user is assigned to (either statically or dynamically) or the maximum number of users who can be assigned to a single role
- **Prerequisite roles** can be enforced – this can be useful for enforcing the principle of **least privilege**, since users can invoke a session with their more privileged role only when required

Attribute-Based Access Control

- Can define authorizations that express conditions on properties of the resource, the subject, and the environment
- A subject's attributes may include an identifier, name, organization, job title, role, etc.
- A resource's attributes may include metadata that indicate ownership, time of creation, QoS attributes, etc.
- Environment attributes may include time and date or the network's security level (e.g., Internet vs intranet)

Attribute-Based Access Control Example

Movie Rating	Users Allowed Access
R	Age 17 and older
PG-13	Age 13 and older
G	Everyone

```
R1:can_access (u, m, e)
 (Age(u) ≥ 17 ∧ Rating(m) ∈ {R, PG−13, G}) ∧
 (Age(u) ≥ 13 ∧ Age(u) ≤ 17 ∧ Rating(m) ∈ {PG−13, G}) ∧
 (Age(u) ≤ 13 ∧ Rating(m) ∈ {G})
```

- A policy-based approach gives flexibility and expressive power, e.g., can add environmental attributes such as promotional periods without creating new roles

Memory Protection

- Early computers were large and expensive: programs were prepared ahead of time and submitted for later processing
  - Submitted jobs were “batched” together and run sequentially by an operator
- Time-sharing systems in 1960s offered an alternative: users are presented with a view of running a program on their own machine in real time
- Security issue arises: How to prevent one process from writing into memory used by another? Or even disrupting OS data or code?
- Solution: Provide memory isolation between processes
  - Each process has its own view of memory space (*virtual memory*)
  - All memory accesses go through a hardware memory management unit (MMU), which maps virtual addresses to physical addresses
  - More details: See textbook and/or your OS course

Other Isolation Mechanisms

- Android: Every installed app has its own UID and is sandboxed.
  - How does inter-app communication happen?
    - UID sharing
    - Intents
    - Any other options?
- Seccomp: System call filtering facility in Linux kernel
- VMs, containers
  - How do these differ?

Figures Credit

Figures and Tables on slides 3, 7, 8, 9, 10, 14, 16, 17, 18, 19, and 22 are taken from Computer Security: Principles and Practice 3e by Stallings & Brown.