Module 4: Access Control

Dr. Natarajan Meghanathan
Associate Professor of Computer Science
Jackson State University, Jackson, MS 39232
E-mail: natarajan.meghanathan@jsums.edu

Access Control

- In general, Access Control involves:
  - Preventing unauthorized users from gaining access to resources (deals more with authentication)
  - Preventing legitimate users from accessing resources in an unauthorized manner
  - Enabling legitimate users to access resources in an authorized manner.
- In this module, we will deal more with the "authorization" component of access control:
  - Access control implements a security policy that specifies who or what (e.g. process may have access to each specific system resource and the type of access that is permitted in each instance.
- Access control deals with subjects, objects and access rights.

Access Control

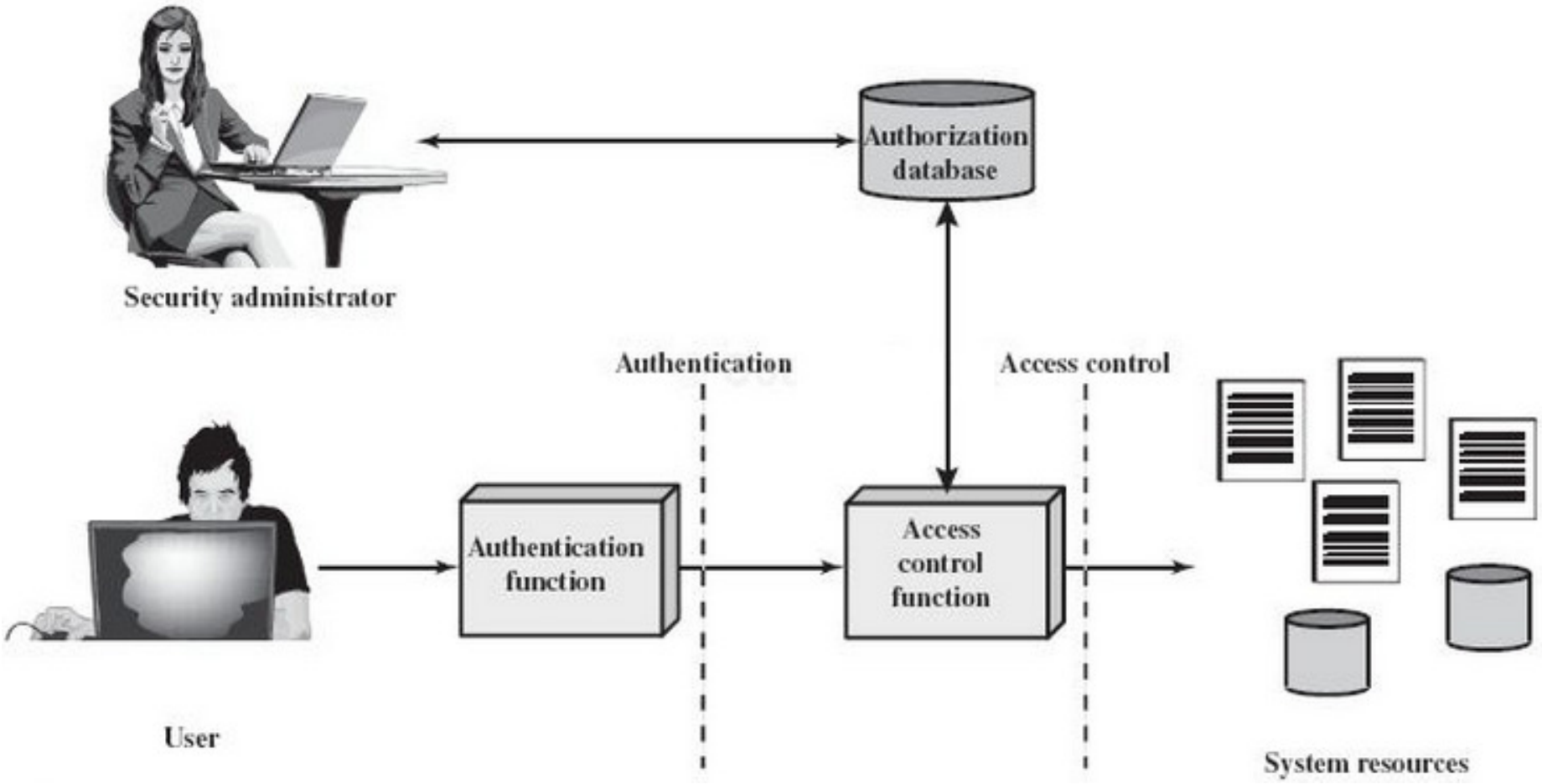


Figure 4.1: W. Stallings: Computer Security: Principles and Practice: 2nd Edition

Access Control Policies

- Discretionary Access Control (DAC)
  - Controls access based on the identity of the requestor (subject) and on access rules stating what the requestor is (or is not) allowed to do.
  - The owner of a resource (object) can delegate access permissions to other users (subjects).
- Mandatory Access Control (MAC)
  - Subjects and Objects are assigned security clearances
  - A subject having an equal or higher security clearance than the object can only access the object.
  - A subject that has clearance to access an object cannot enable another subject to access that object (access control decisions are taken at the admin level; not at the subject-level)
- Role-based Access Control (RBAC)
  - Controls access based on the roles that users (subjects) have within the system
  - There are rules stating what accesses are allowed to each role.

Requirements for Access Control

- Reliable input: Subjects (users) and sources of access requests (like IP address, MAC address) validated through a proper authentication mechanism
- Support for fine and coarse grained specifications: Depending the need, access rules may be specified at the level of individual records and individual fields within records (fine-specs) and sometimes for a collection of records/objects (coarse-grained spec to reduce admin burden).
- Least privilege: Each system entity should be granted the minimum system resources and authorizations that the entity needs to do work.
- Open and Closed Policies: Closed Policy – only accesses that are explicitly stated should be allowed; Open Policy – Authorizations specify which accesses are prohibited; all other accesses are allowed.
- Policy combinations and Conflict resolution: Care should be taken that there are conflicts due to combination of more than one policy, and if there are conflicts, they need to be resolved.
- Administrative Policies: There is a security administration function for specifying the authorization database that acts as an input to the access control function.

Basic Elements of Access Control

- Subject: An entity capable of accessing objects.
  - Subject typically represents a process (the process takes on the attributes, such as access rights, of the user or application)
    - Owner: creator of a resource
    - Group: group of users; membership in the group is sufficient for certain access rights
    - World: Users who are not included in the categories of owner and group may be able to access the resources with limited permissions.
- Object: Resource to which access is controlled.
  - An entity that contains and/or receives information.
  - E.g.: Records, blocks, pages, segments, files, directories, messages, programs, etc.
- Access right: describes the way in which a subject may access an object:
  - *Read* (incl. copy or print); *Write* (incl. read access; add, modify or delete); *Execute*; *Delete*; *Create*; *Search* (list the files in a directory or search the directory)

DAC: Access Matrix

		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

Figure 4.3 (a): W. Stallings: Computer Security: Principles and Practice: 2nd Edition

Access Control Lists and Capability Lists

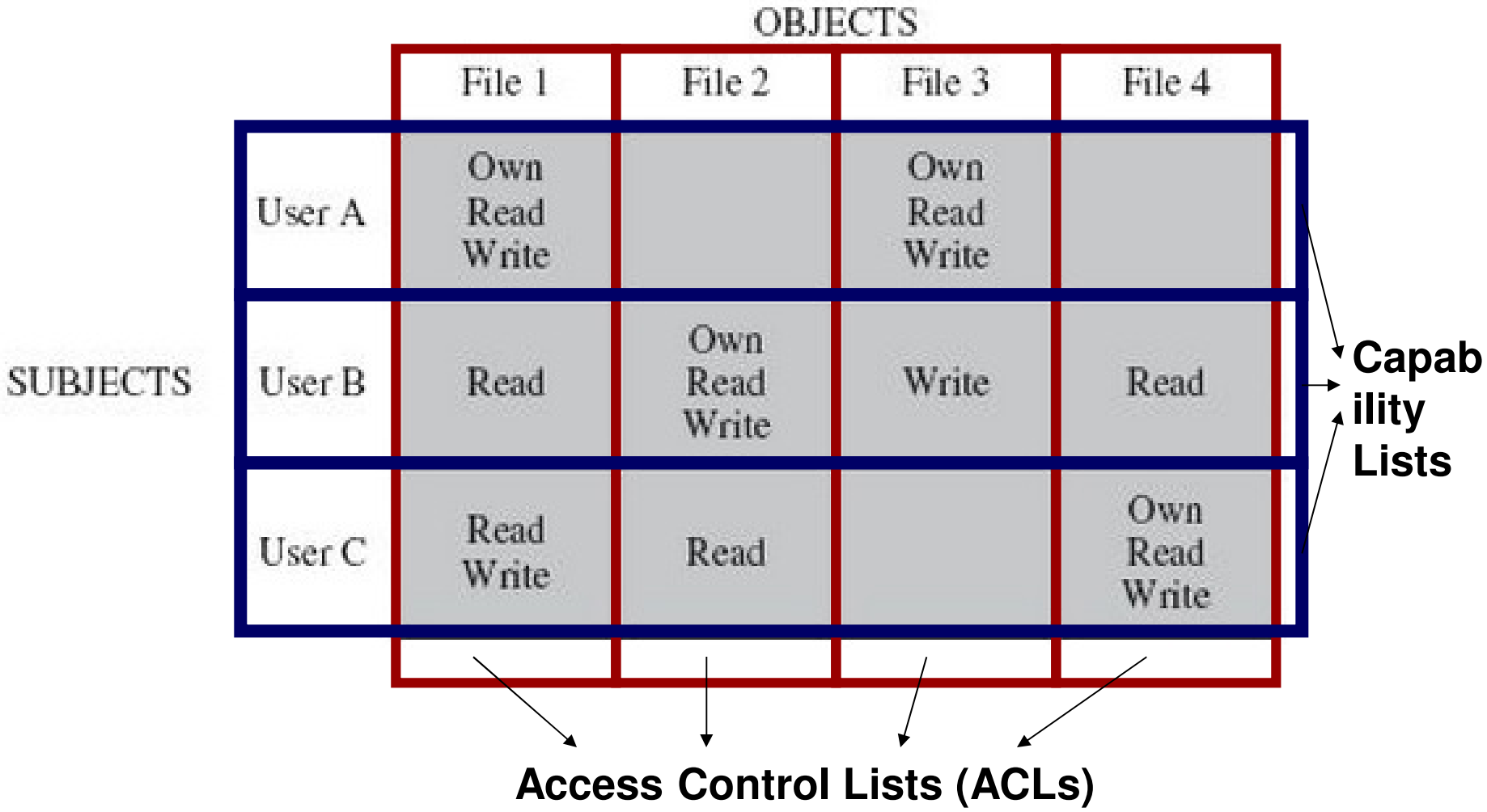
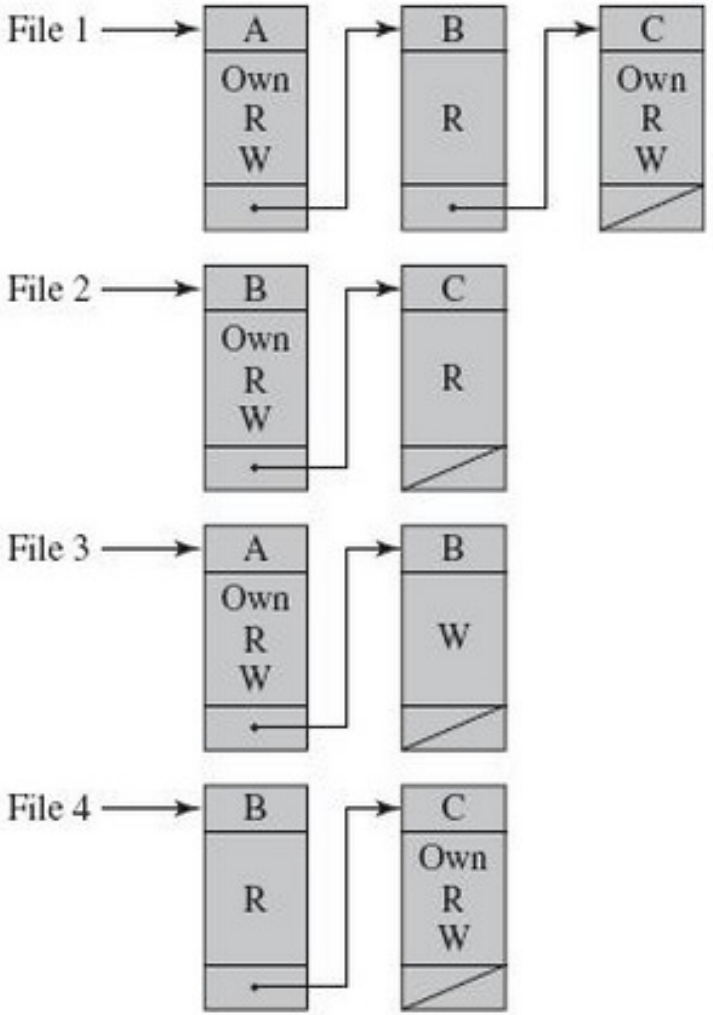
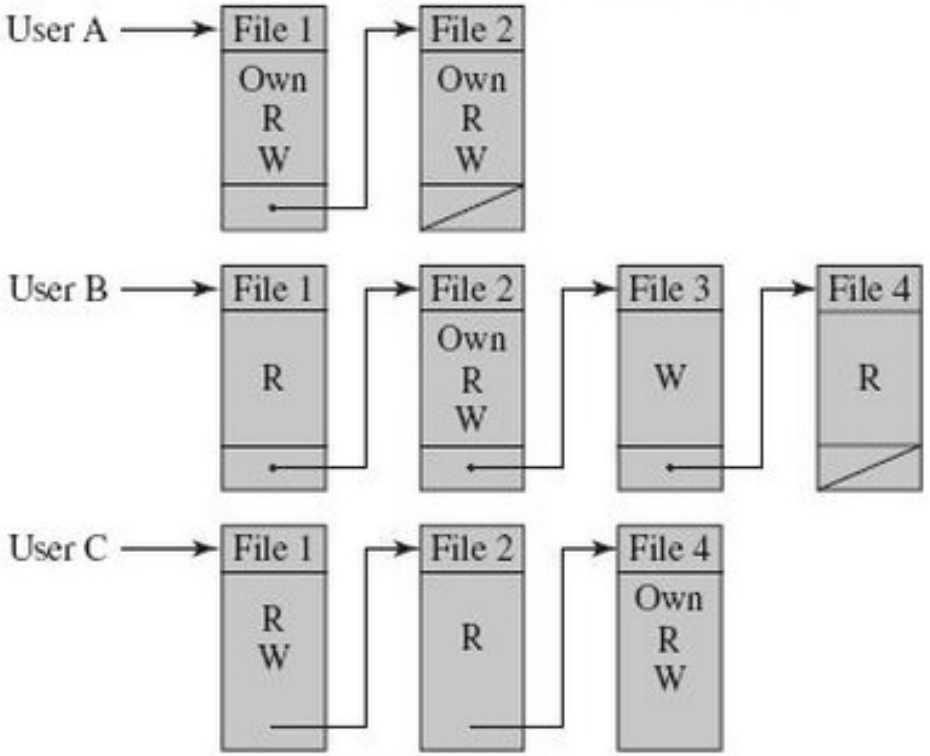


Figure 4.3 (a): W. Stallings: Computer Security: Principles and Practice: 2nd Edition

ACLs and Capability Lists



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



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

Figure 4.3 (b) and (c): W. Stallings: Computer Security: Principles and Practice: 2nd Edition

Tradeoff between ACLs and Capability Lists

- ACLs are stored only at the authorization center and every request from a subject to access an object has to be validated by searching through the ACL of the object.

- In the context of Capability Lists, the permissions that a user has on an object are captured in the form of a capability ticket that is stored at the user side and submitted to the authorization center at the time of access requests.
  - Though there is less overhead in validating access requests, the capability tickets have to be made secure by accompanying it with a message authentication code that can be verified every time an access request is made.
  - A subject can even loan its capability ticket to other subjects. The security of the objects may be at stake.

UNIX File Access Control

- All types of UNIX files (including directory) are administered by the OS by means of inodes (index node): one inode per file.
- An inode stores the attributes of the file, its permissions and other control information.
- On the disk, there is an inode table that contains the inodes of all files in the file system.
  - When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.
- A directory in UNIX is simply a file that contains a list of file names and pointers to associated inodes.
- Each UNIX user is assigned a unique user identifcation number (User ID).
  - A user is also a member of a primary group (each group identified by a Group ID)
- When a file is created, it is designated as owned by a particular user and belong to a specific group (primary group of the owner or the group of its parent directory, in case the latter's SetGID bit is set).

UNIX File Access Control

- Associated with each file is a set of 12 protection bits.
- Nine of the protection bits specify read, write and execute permission for the owner of the file, other members of the group to which the file belongs, and all other users.
  - These form a hierarchy of owner, group and all others, with the highest relevant set of permissions being used.
- When applied to a directory,
  - the read bit lets one to list the contents of the directory
  - the write bit lets one to create/delete/rename files in the directory
  - the execute bit grants right to descend into the directory or search it for a filename.
- SetUID (Set User ID) bit and SetGID (Set Group ID) bits:
  - If these are set on an executable file: When a user (with execute privileges for this file) executes the file, the system temporarily allocates the rights of the user's ID of the file's creator and the file's group, to those of the user executing the file (referred to as the effective user id and effective group id at the time of execution, in addition to the real user id and real group id).

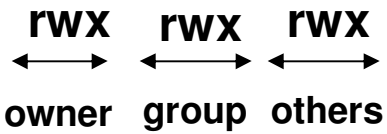
UNIX File Access Control

- SetUID (Set User ID) bit and SetGID (Set Group ID) bits:
  - If these are set on an executable file:
    - This change is effective only while the program is being executed.
    - This feature enables the creation and use of privileged programs that may use files normally inaccessible to other users (e.g., passwd program)
  - If these are set on a directory, the SetGID permission indicates that newly created files will inherit the group of this directory. The SetUID permission is ignored.

- Sticky bit
  - When set for a file, the file contents are not totally moved out of main memory after execution, and are stored in swap space: set for frequently used programs to speedup execution.
  - When set for a directory, only the owner of a file in the directory can move, rename or delete the file: useful for managing files in shared temporary directories.

UNIX File Permissions

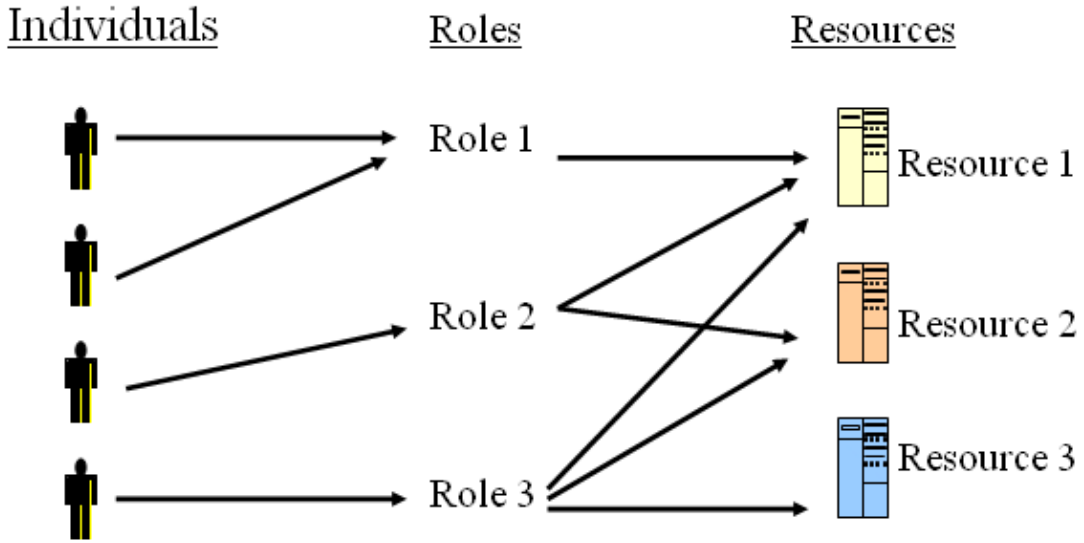
- The owner uses the `chmod` command to set the access rights of a file and can use the `chown` command to change the owner or group of a file.
- The Access rights are: Read (r – 4), Write (w – 2) and Execute (x – 1), nothing (0).
- Each file has associated permissions of the form



- If a file has to be given more than one access right to a class, we have to add their corresponding values.
- **Examples:**
  - Consider a file A.txt. To set read, write and execute permissions to its owner, read and execute only permission to the group and read-only permission to others, use the chmod command as **chmod 754 A.txt**
  - To set the SetUID bit (4) and SetGID bit (2) to a file and set read, write and execute permissions to the owner, read/write permission to group and read permissions to others, use the chmod command as: **chmod 6764 A.txt**
  - To set the sticky bit to a file, use: **chmod +t A.txt**

Role-based Access Control (RBAC)

- RBAC models define a role as a job function within an organization.
- RBAC: Users are assigned to different roles (static or dynamic) according to their responsibilities; each role is assigned the specific access rights to one or more resources (mostly static)
- Set of users may change frequently, as is the assignments of a user to one or more roles.
- Each role should contain the minimum set of access rights needed for that role.
- A user is assigned to a role that enables him or her to perform only what is required for that role.



RBAC Model Sub Categories

- RBAC Session: It is a particular instance of a connection of a user to the system and defines the subset of activated roles. When a user logs in the system, he/ she establishes a session, and during this session, can request to activate a subset of the roles he/she is authorized to play.
- Core RBAC (Flat RBAC): Mainly used if a user is assigned only one role per login session.
- Hierarchical RBAC: If the user can take on multiple roles during a login session, the number of times the access permissions have to be checked can be minimized if the user is authenticated for the role that is higher up in the hierarchy among the roles the user intends to take on during the session.
- Constrained RBAC: The number of roles and/ or the combination of roles that a user can simultaneously take during a login session could be restricted according to a static or dynamic policy of Separation of Duties (SoD).
- The Static SoD constraints can be enforced upfront (i.e., before login) based on the users-role mapping and the maximum number of roles per user per session..
- The Dynamic SoD constraints can be enforced by keeping track of the user access to roles within a session. In addition to limiting the number of roles per session, constraints could be even subjective (for e.g., a user cannot be assigned role r2 if he/she already has a role r1 in the same session).

Hierarchical RBAC

- The hierarchy in a RBAC could be represented either in the form of a tree or a lattice.
- We say a role r_i dominates r_j ($r_i \geq r_j$), if r_i appears higher up than r_j in the hierarchy of roles.
- User Inheritance: All users authorized for a role r are also authorized for any role r' where $r \geq r'$.
- Permission Inheritance: A role r is authorized for all permissions for which any role r', such that $r \geq r'$, is authorized.
- Activation Inheritance: Activating a role r automatically activates all roles r', such that $r \geq r'$.