Access Control - I

CSE 565: Fall 2024
Computer Security

Xiangyu Guo (xiangyug@buffalo.edu)

University at Buffalo

Acknowledgement

- We don’t claim any originality of this slides. The content is developed heavily based on
  - Slides from Prof Ziming Zhao’s past offering of CSE565 (https://zzm7000.github.io/teaching/2023springcse410565/index.html)
  - Slides from Prof Marina Blanton’s past offering of CSE565 (https://www.acsu.buffalo.edu/~mblanton/cse565/)
  - Slides from Prof Hongxin Hu’s past offering of CSE565

Announcement

- Assignment 1 & Project 1 is due tomorrow (Wed 09/25) 23:59

Review of Last Week

- Authentication:
  - Password-based authentication
  - Token-based authentication
  - Biometric-based authentication

Today's Topic

- Access control principles
  - Access control matrices
    - Access control lists (ACLs)
    - Capability tickets
- POSIX File Permissions
- Types of access control
  - Discretionary access control
  - Mandatory access control
  - Role-based access control (RBAC)
  - Attribute-based access control (ABAC)

Authorization: overview

Big Picture: Access Control and Other Security Functions

Security Admin

Authorization database

Authentication

Access Control

Users

Authentication function

Access Control function

Auditing

Access Control Principles

- **Least Privilege**
  - Each entity is granted the minimum privileges necessary to perform its work
  - Limits the damage caused by error or intentional unintended behavior
- **Separation of Duty**
  - Practice of dividing privileges associated with one task among several individuals
  - Limits the damage a single individual can do

Modeling Access Control

Access Control Matrix

- A **table** that defines **permissions**.
  - Each *row* of this table is associated with a **subject**, which is a user, group, or system that can perform actions
  - Each *column* of the table is associated with an **object**, which is a file, directory, document, device, resource, or any other entity for which we want to define access rights
- Each cell of the table is then filled with the access rights for the associated combination of subject and object
  - Access rights can include actions such as reading, writing, copying, executing, deleting, and annotating.
  - An empty cell means that no access rights are granted.

Access Control Matrix

Example

- **Subjects** (users) index the rows
- **Objects** (resources) index the columns

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	——	——
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	r	r
Accounting Manager	rx	rx	rw	rw	rw

Access Control Matrix

- Access control matrix has all relevant info
- But how to manage a large access control (AC) matrix?
  - Could be 1000's of users, 1000's of resources
  - Then AC matrix with 1,000,000's of entries
  - Need to check this matrix before access to any resource is allowed
  - Hopelessly inefficient

Implementing Access Control

Access Control Lists (ACLs)

- ACL: store access control matrix by *column*

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	——	——
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	r	r
Accounting Manager	rx	rx	rw	rw	rw

Access Control Lists (ACLs)

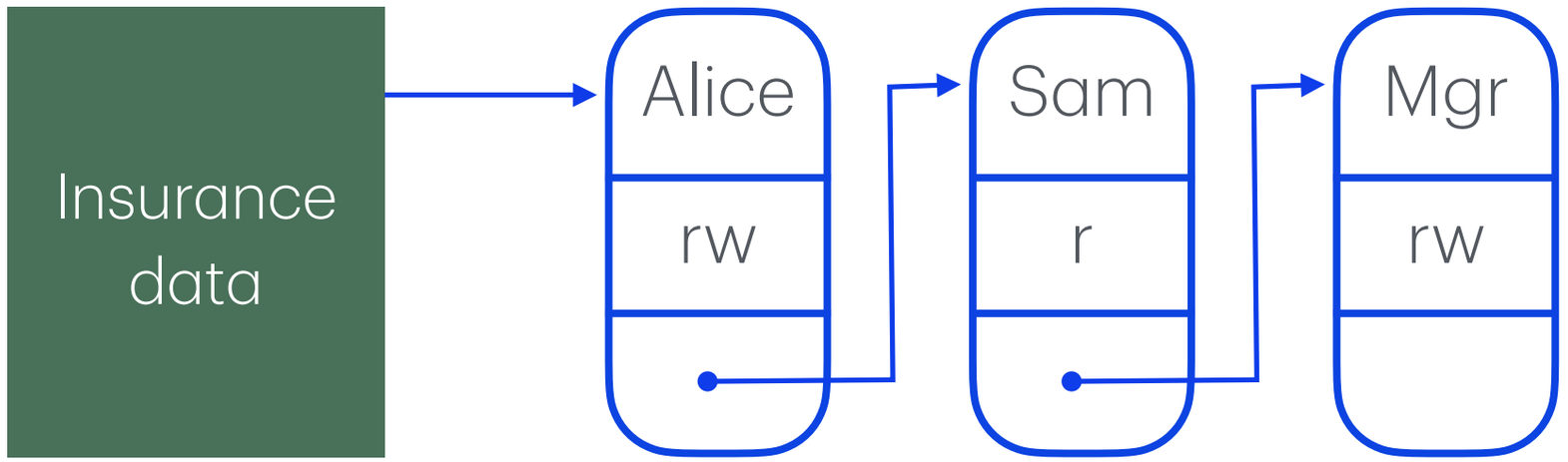
- ACL: store access control matrix by *column*

Example: ACL for Insurance data

	OS	Accounting program	Accounting data	**Insurance data**	Payroll data
Bob	rx	rx	r	— —	— —
Alice	rx	rx	r	**rw**	rw
Sam	rwx	rwx	r	**r**	r
Accounting Manager	rx	rx	rw	**rw**	rw

Access Control Lists (ACLs)

- ACL: store access control matrix by *column*



Note: Bob is not in the list since he has no permission

Capabilities (or C-Lists)

- Capabilities: Store access control matrix by *row*

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	——	——
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	r	r
Accounting Manager	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

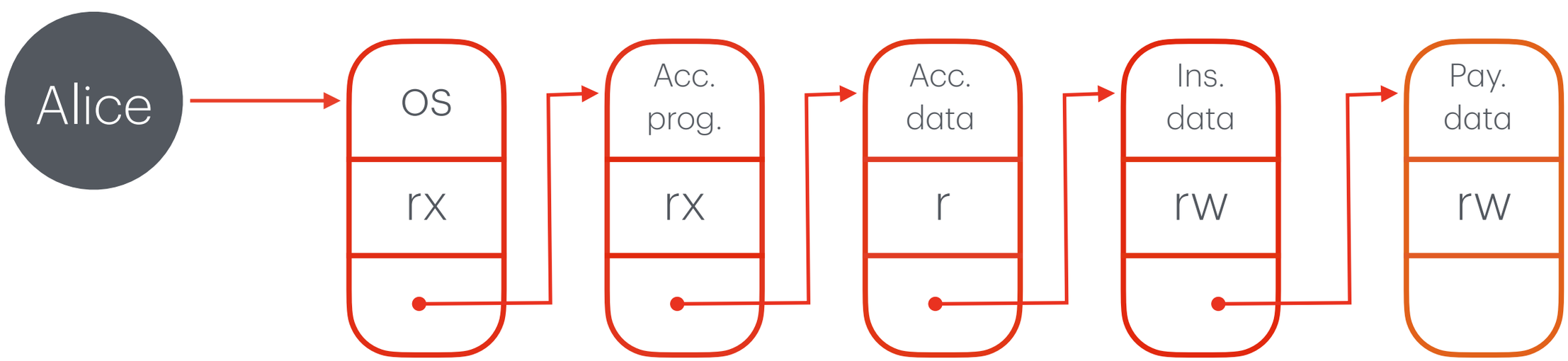
- Capabilities: Store access control matrix by row

Example: Capability for Alice

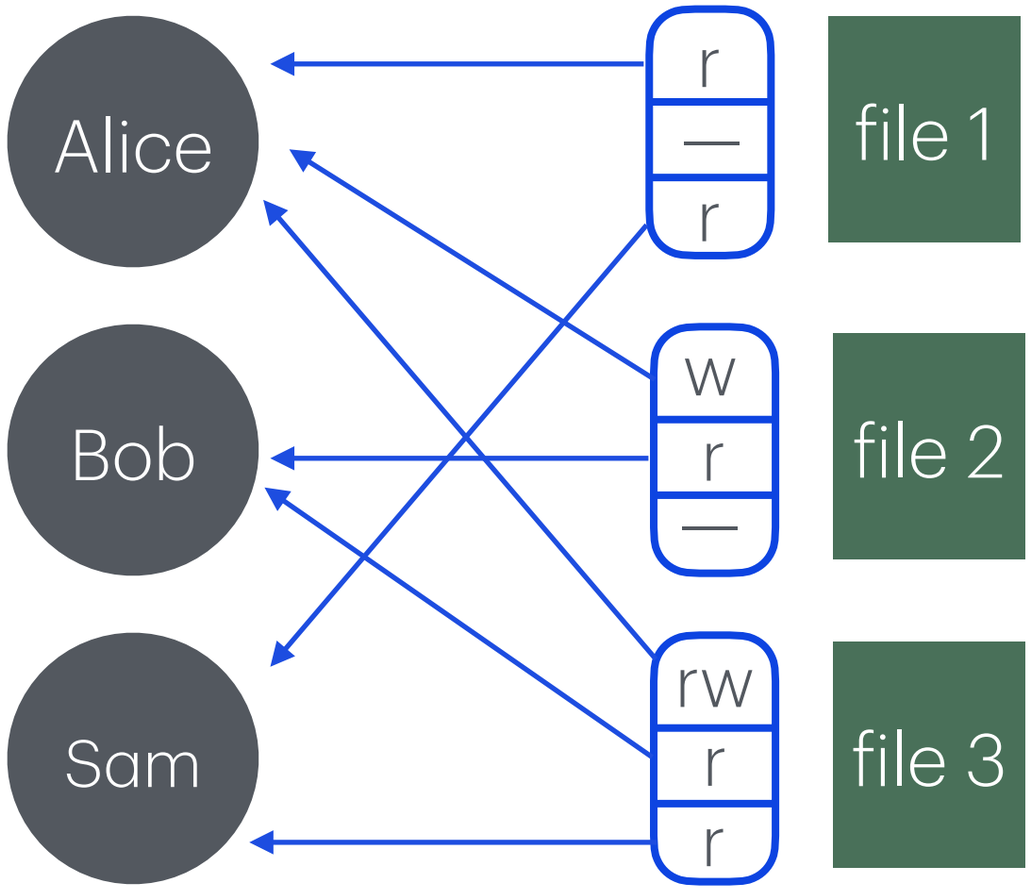
	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	——	——
Alice	**rx**	**rx**	**r**	**rw**	**rw**
Sam	rwx	rwx	r	r	r
Accounting Manager	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

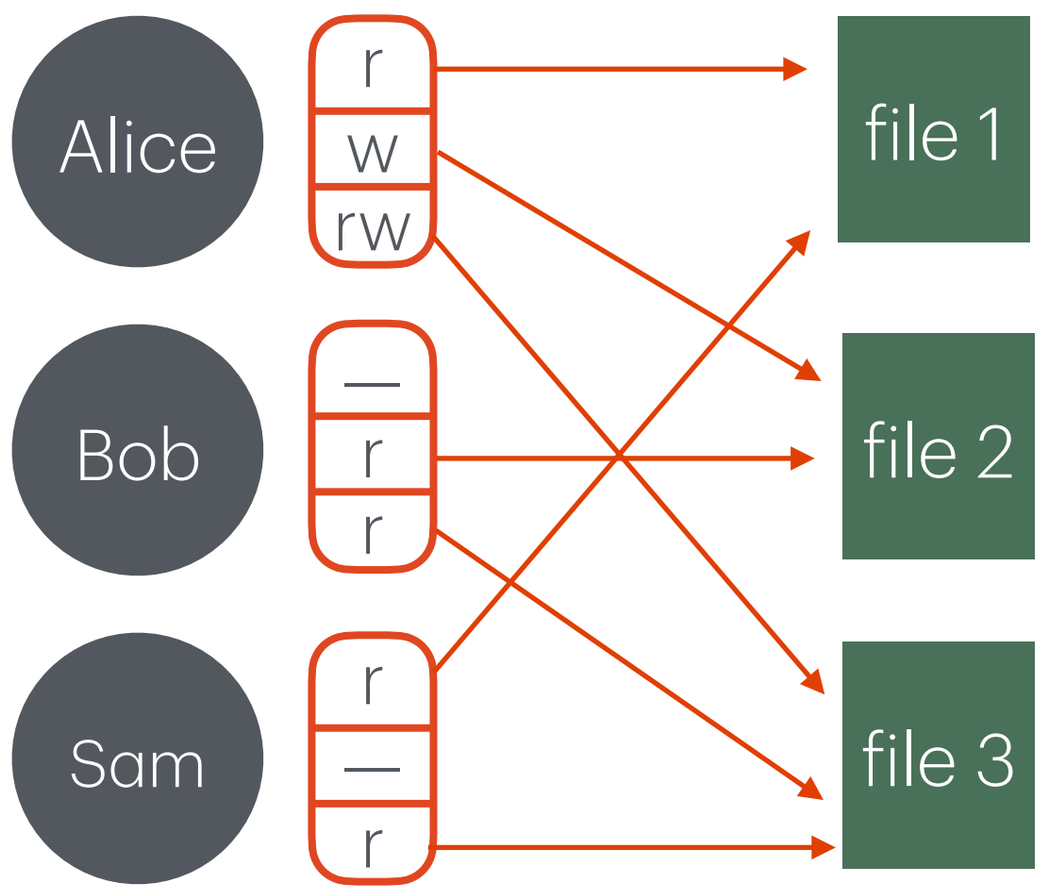
- Capabilities: Store access control matrix by row



ACL vs Capabilities



ACLs



Capabilities

Note that arrows point in opposite directions

ACL vs Capabilities

- **ACLs**
  - Protection is *data-oriented*
  - Good when users manage their own files
  - Easy to change rights to a resource
  - Most real-world OSs use ACLs.
- **Capabilities**
  - Protection is *user-oriented*
  - Easy to delegate
  - Easy to add/delete users
  - More difficult to implement

- Question: Facebook – ACLs vs Capabilities?

UNIX File Access Control

UNIX File Permissions: Permission Groups

Each file and directory has three user-based permission groups:

- **Owner** – A user is the owner of the file. By default, the person who created a file becomes its owner. The Owner permissions apply only the owner of the file or directory
- **Group** – A group can contain multiple users.
  - All users belonging to a group will have the same access permissions to the file.
  - The Group permissions apply only to the group that has been assigned to the file or directory
- **Others** – The others permissions apply to all other users on the system.

UNIX File Permissions: Permission Types

Each file or directory has three basic permission types defined for all the 3 user types:

- **Read** – The Read permission refers to a user's capability to read the contents of the file.

- **Write** – The Write permissions refer to a user's capability to write or modify a file or directory.

- **Execute** – The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Example

```
hacker@access-control~level6:~$ ls -la /tmp
total 36
drwxrwxrwt 1 root   root   4096 Sep 24 04:04 .
drwxr-xr-x 1 root   root   4096 Sep 24 04:04 ..
-rw-rw-r-- 1 root   root      4 Sep  6 16:44 .cc.txt
-rw-r--r-- 1 root   root     55 Sep  6 16:44 .crates.toml
-rw-r--r-- 1 root   root    453 Sep  6 16:44 .crates2.json
drwxr-xr-x 2 hacker hacker 4096 Sep 24 04:04 .dojo
drwxr-xr-x 2 root   root   4096 Sep  6 16:44 bin
drwxr-xr-x 1 root   root   4096 Sep  6 16:20 hsperfdata_root
drwx------ 2     104    105 4096 Sep 15 09:07 tmp.XvrUsDZh8M
```

permission type

owner

group

size

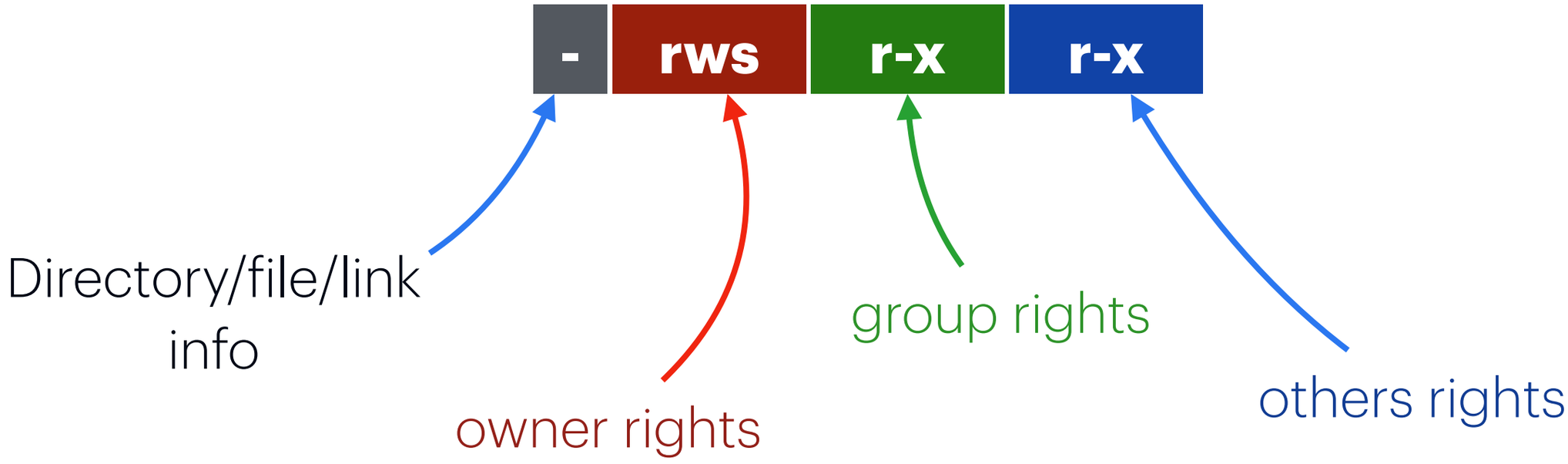
Last modification time

Example: permission type

r: read

w: write

x: execute
s: set-UID/GID execute
t: sticky bit



Access Control Models

Access Control Models

- Discretionary access control (**DAC**)
  - Controls access based on the *identity* of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do
- Mandatory access control (**MAC**)
  - Controls access based on comparing security labels with security clearances
- 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

- In Mandatory Access Control (MAC) users are granted privileges, which they cannot control or change
- Discretionary access control (DAC) has provisions for allowing subjects to grant privileges to other subjects

Discretionary Access Control

- **Owner-based Control**
  - Each object (e.g., files, directories) has an owner, typically the creator of the object.
- **Flexible Delegation**
  - The owner can delegate access to other users. The granted permission can further propagate.
- **Identity-based Access**
  - Access control is typically based on user identity or group membership.

Discretionary Access Control

- The access control matrix can be extended to include different types of objects
  - **The subjects themselves can also be objects**
- Different types of objects can have different access operations defined for them
  - e.g., stop and wake-up rights for processes, read and write access to memory, seek access to disk drives

	s_1	...	s_n	o_1	o_2	...	o_m
s_1							
s_2							
...							
s_n							

Discretionary Access Control

- Suppose we have the following access rights
  - basic **read** and **write**
  - **own**: possessor can change their own privileges
  - **copy** or **grant**: possessor can extend its privileges to another subject
    - This is modeled by setting a copy flag '*' on the access right
    - for example, right r cannot be copied, but $r*$ can

Discretionary Access Control

Primitive commands

- create-object(o)
- delete-object(o)
- create-subject(s)
- delete-subject(s)
- write-object(o)
- grant-right(r, s, o)
- revoke-right(r, s, o)
- transfer-right(r, s_1, s_2, o)
- chown(s, o)

DAC in Unix File System

- Access control is enforced by the operating system
- **Files**
  - How is a file identified?
  - Where are permissions stored?
  - Is directory a file?
- **Users**
  - Each user has a unique ID
  - Each user is a member of a primary group (and possibly other groups)

DAC in Unix File System

- Subjects are processes acting on behalf of users
  - Each process is associated with a uid/gid pair
- Objects are **files** and **processes**
  - Each file has information about: owner, group, and 12 permission bits
  - read/write/execute for owner, group, and others
  - suid, sgid, and sticky bit

DAC in Unix File System

- DAC is implemented by using commands like **chmod/chown** (explicit) or **cp/mv/rm** (implicit)
  - A special user “superuser” or “root” is exempt from regular access control constraints
- Many Unix systems support additional ACLs
  - Owner (or administrator) can add to a file users or groups with specific access privileges
  - The permissions are specified per user or group as regular three permission bits
  - **setfacl** and **getfacl** commands change and list ACLs
- This is called *extended ACL*, while the traditional permission bits are called *minimal ACL*

Security of DAC

- What is *secure* in the context of DAC?
  - A secure system doesn't allow violations of policy
  - How can we use this definition?
- Alternative definition based on rights
  - Start with access control matrix A that already includes all rights we want to have
  - A **leak** occurs if commands can add right r to an element of A not containing r
  - A system is **safe** with respect to r if r cannot be leaked

Decidability of DAC Models

- **Decidable**
  - Given a system, where each command consists of a single *primitive* command
  - There exists an algorithm that will determine if the system with initial state X_0 is safe with respect to right r
- **Undecidable**
  - Given a system that has *non-primitive* commands
  - Given a system state, it is undecidable if the system is safe for a given generic right
  - The safety problem can be reduced to the halting problem by simulating a Turing machine
- Some other special DAC models can be decidable

Does Safety Mean Security?

- Does “safe” really mean secure?
- Example: Unix file system
  - root has access to all files
  - owner has access to their own files
  - Is it safe with respect to file access right?
- Have to disallow **chmod** and **chown** commands
  - only “root” can get root privileges
  - only user can authenticate as themselves
- Safety doesn’t distinguish a leak from authorized transfer of rights

Security in DAC

- Solution is ***trust***
  - Subjects authorized to receive transfer of rights are considered “trusted”
    - Trusted subjects are eliminated from the access control matrix
- Also, safety only works if maximum rights are known in advance
  - Policy must specify all rights someone could get, not just what they have
  - But how applicable is this?
- And safety is still undecidable for practical models

Questions?