Authorization

CSE 565 - Fall 2025

Computer Security

Hongxin Hu (hongxinh@buffalo.edu)

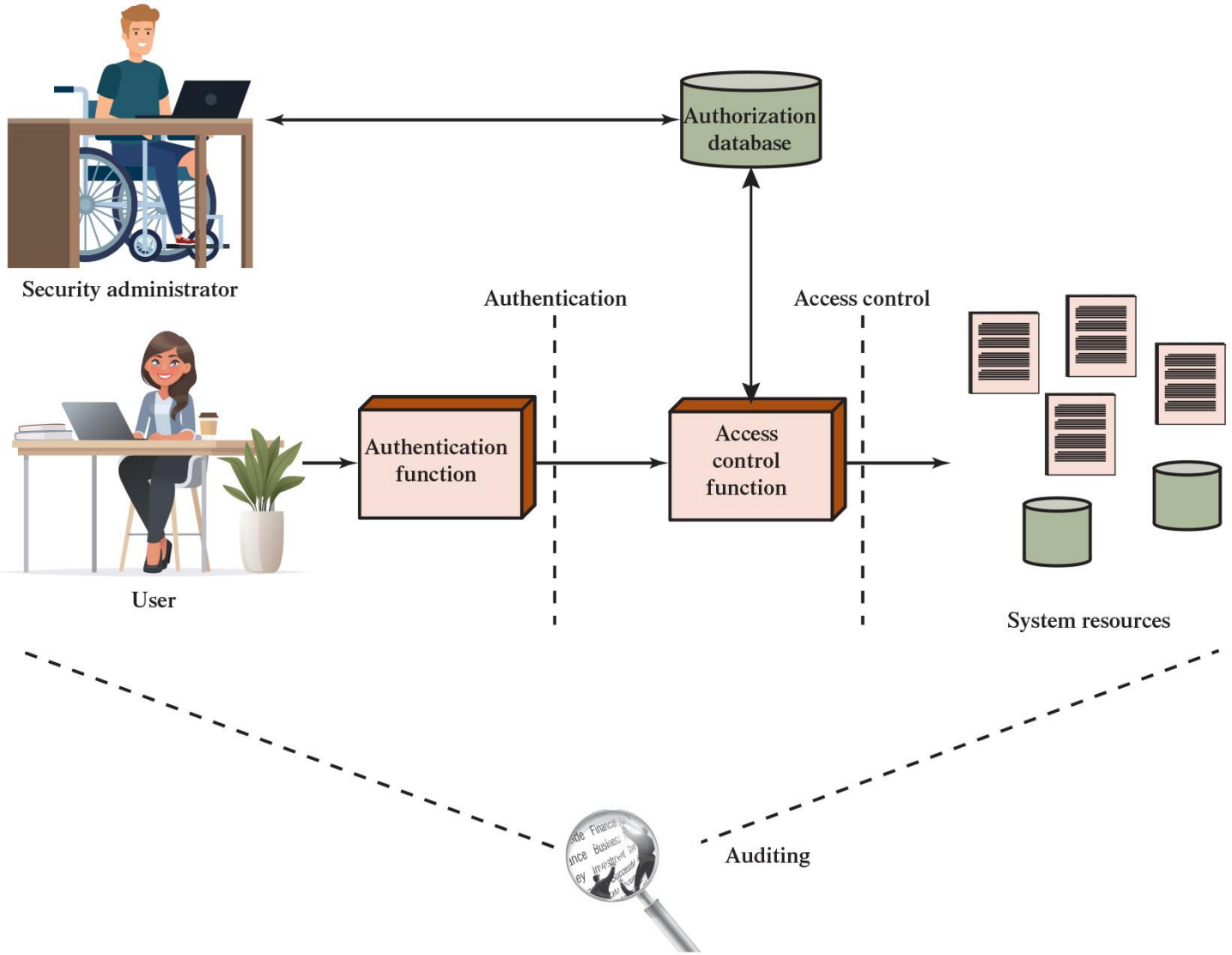
Updates

- **Project 1 Secret-Key Encryption**
  - Deadline: Thursday, September 18

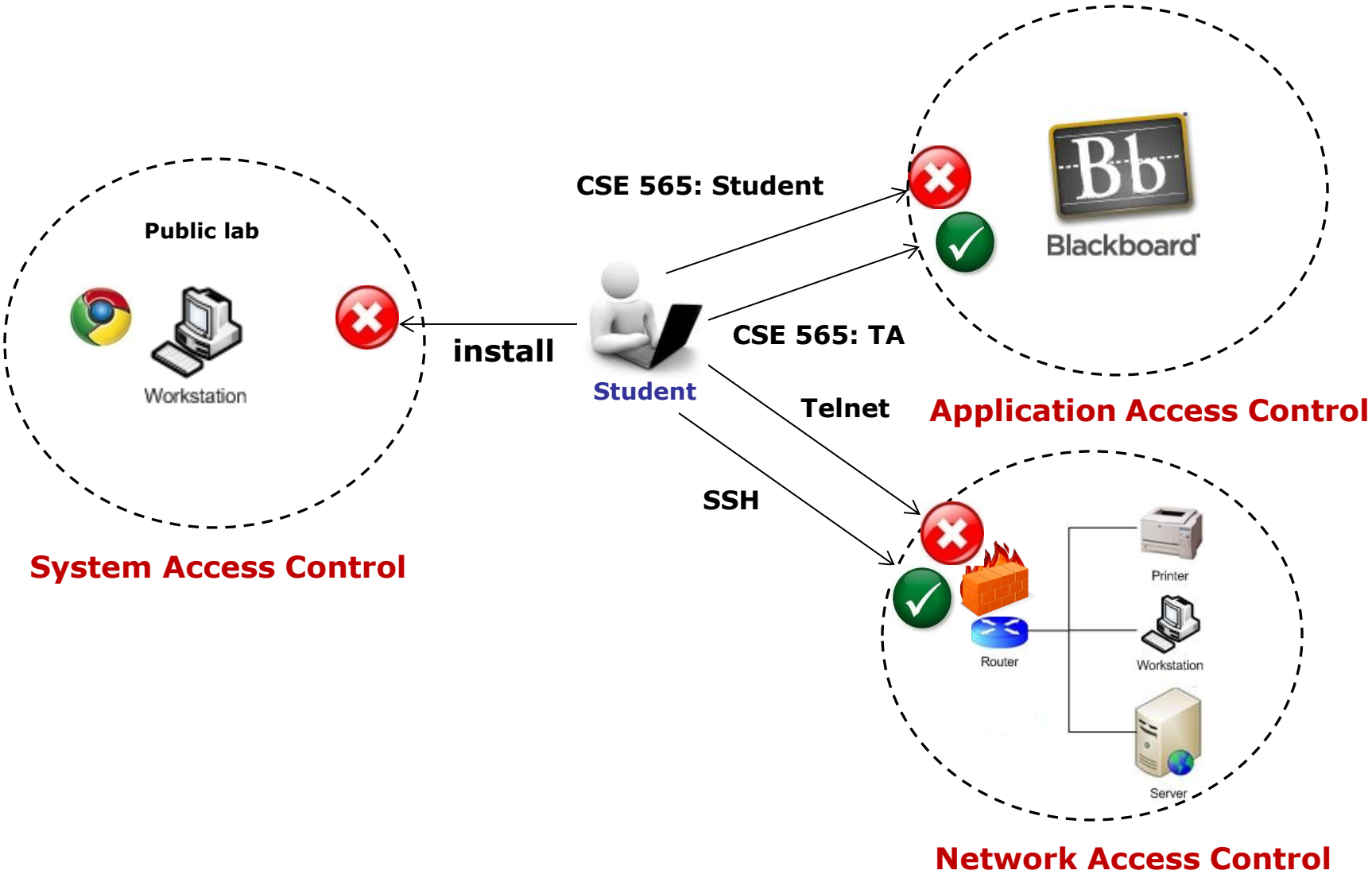
Authentication vs Authorization

- Authentication — Who goes there?
  - Restrictions on who (or what) can access system
- **Authorization** — Are you allowed to do that?
  - Restrictions on actions of authenticated users
  - Authorization is a form of **access control**

Access Control and Other Security Functions

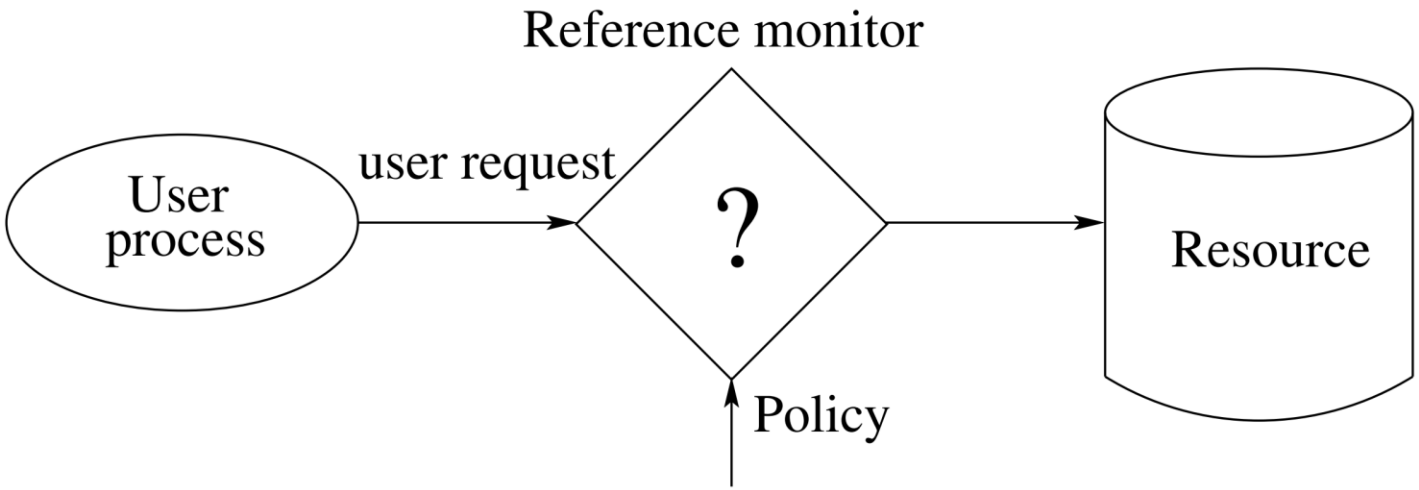


Access Control in the Real World



Access Control Model Basics

- Reference monitor mediates access to resources
  - Complete mediation means controlling all accesses to resources



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

Access Control Matrices

- **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.

Example Access Control Matrix

- **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	rw	rw
Accounting Manager	rx	rx	rw	rw	rw

Are You Allowed to Do That?

- **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

Authorization

- Authorization enforced by
  - Access Control Lists
  - Capabilities

Access Control Lists (ACLs)

- ACL: store access control matrix by **column**
  - Example: ACL for **insurance data** is in **blue**

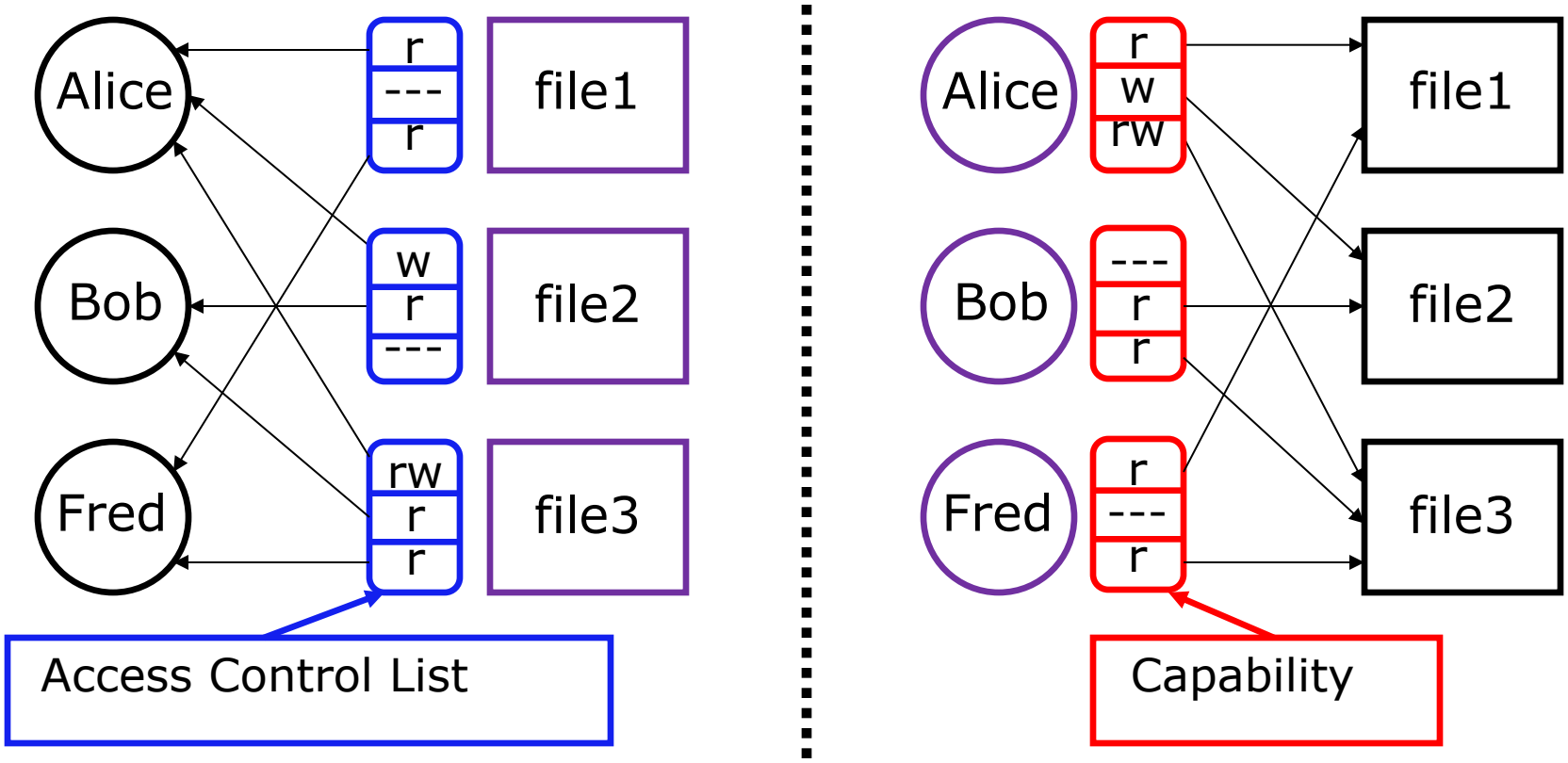
	OS	Accounting program	Accounting data	**Insurance data**	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

- Store access control matrix by **row**
  - Example: Capability for **Alice** is in **red**

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

ACLs vs Capabilities



- Note that arrows point in opposite directions

ACLs vs Capabilities

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

- Question:
  - Facebook – ACLs vs Capabilities?

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

- 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_m	p_1	...	p_l
s_1									
...									
s_n									

Discretionary Access Control

- The access control matrix can be extended to include different types of objects

		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

- For simplicity assume that we are dealing with one type of objects

Discretionary Access Control

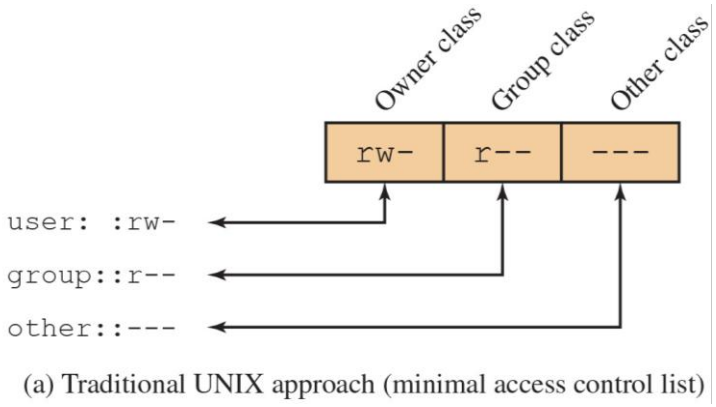
- Suppose we have the following access rights
  - basic read and write
  - own: possessor can change their own privileges
  - copy or grant: possessor can extend its privileges to another subject
    - this is modeled by setting a copy flag on the access right
    - for example, right r cannot be copied,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

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
- Example

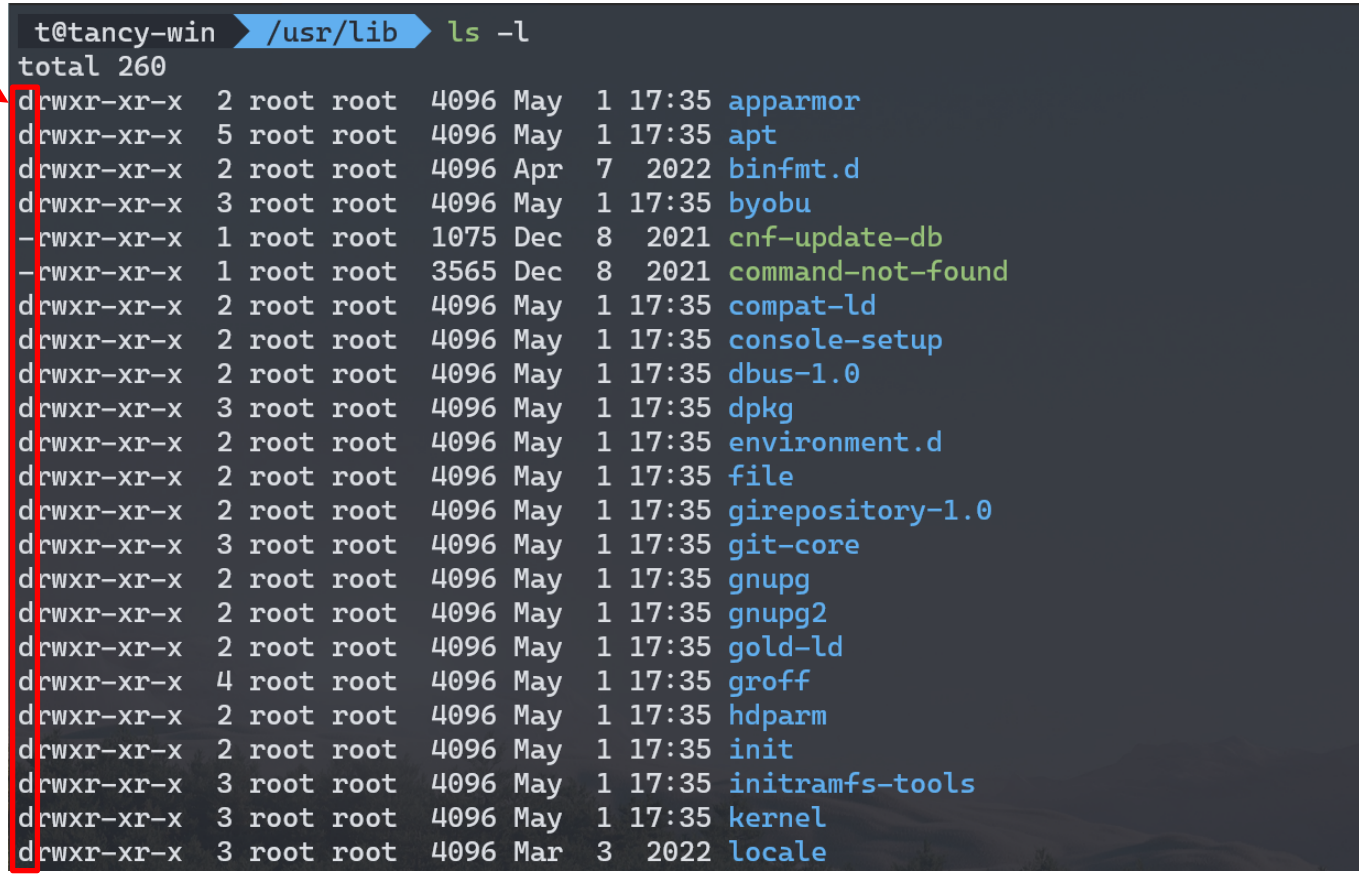


(a) Traditional UNIX approach (minimal access control list)

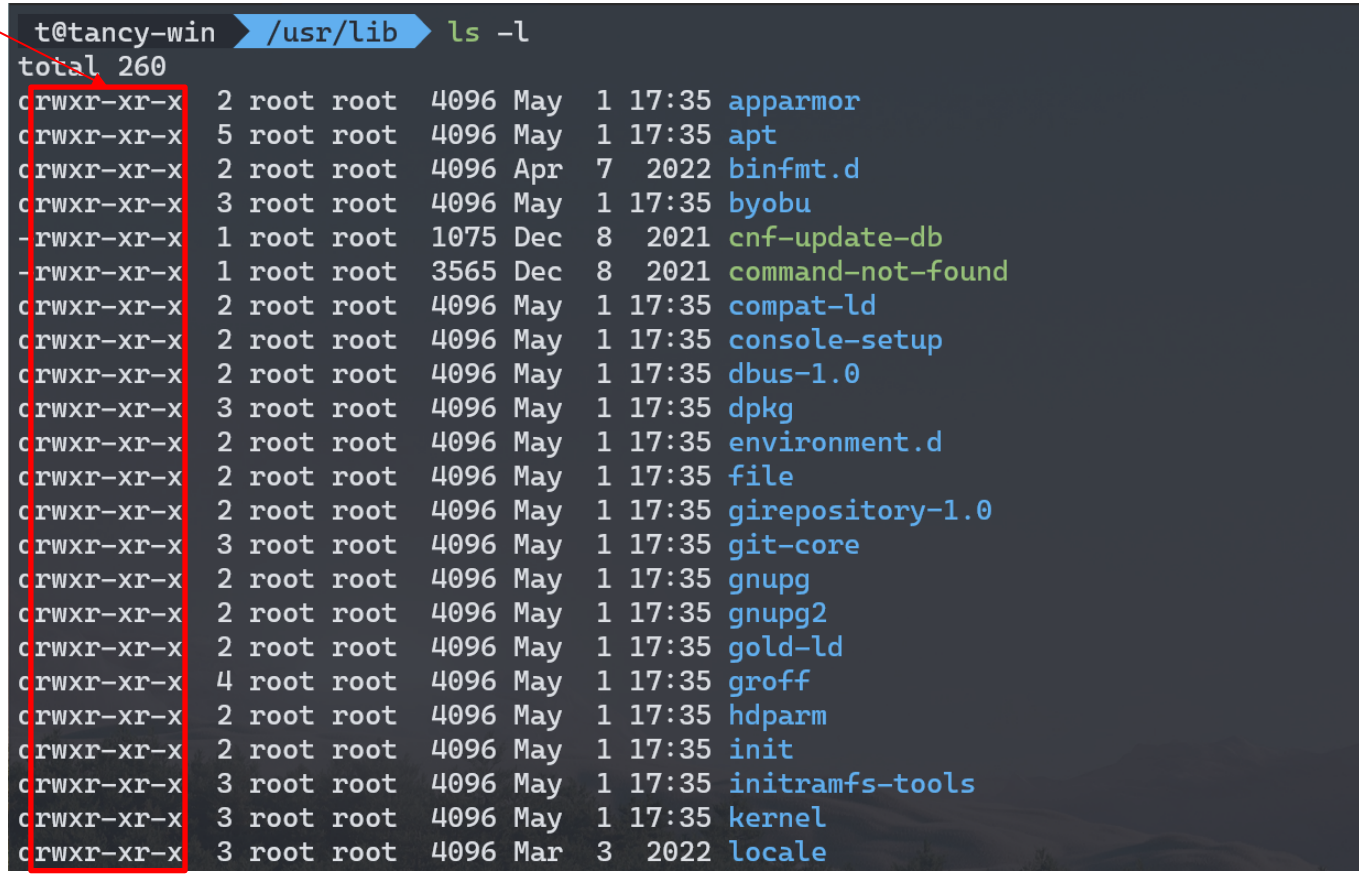
DAC in Unix File System

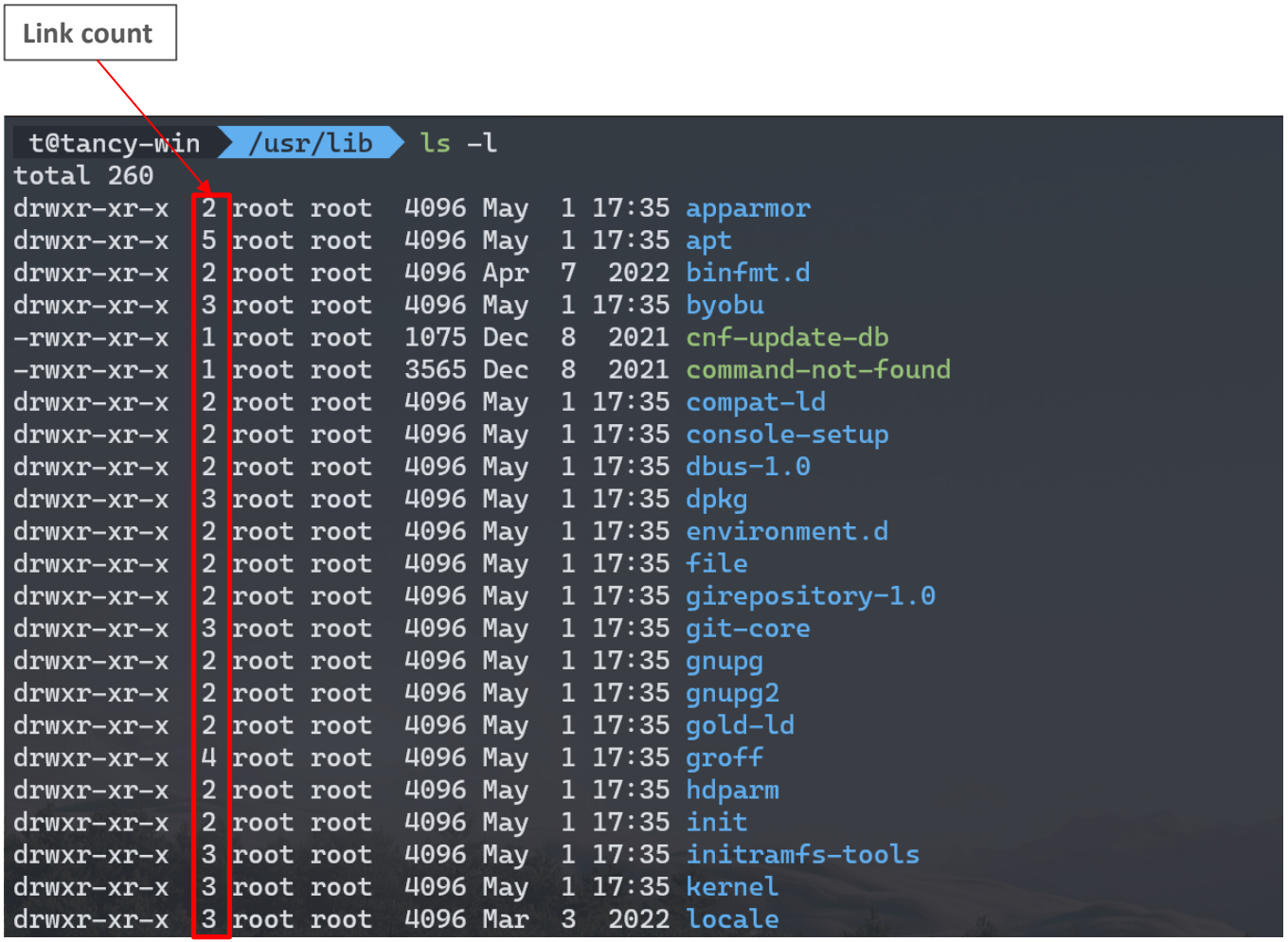
- DAC is implemented by using commands *chmod* and *chown*
- A special user “superuser” or “root” is exempt from regular access control constraints
- Many Unix systems support additional ACLs
  - owner (or administrator) can add to a file users or groups with specific access privileges
  - the permissions 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

File type: First field in the output is file type. If the there is a – it means it is a plain file. If there is d it means it is a directory, c represents a character device, b represents a block device.

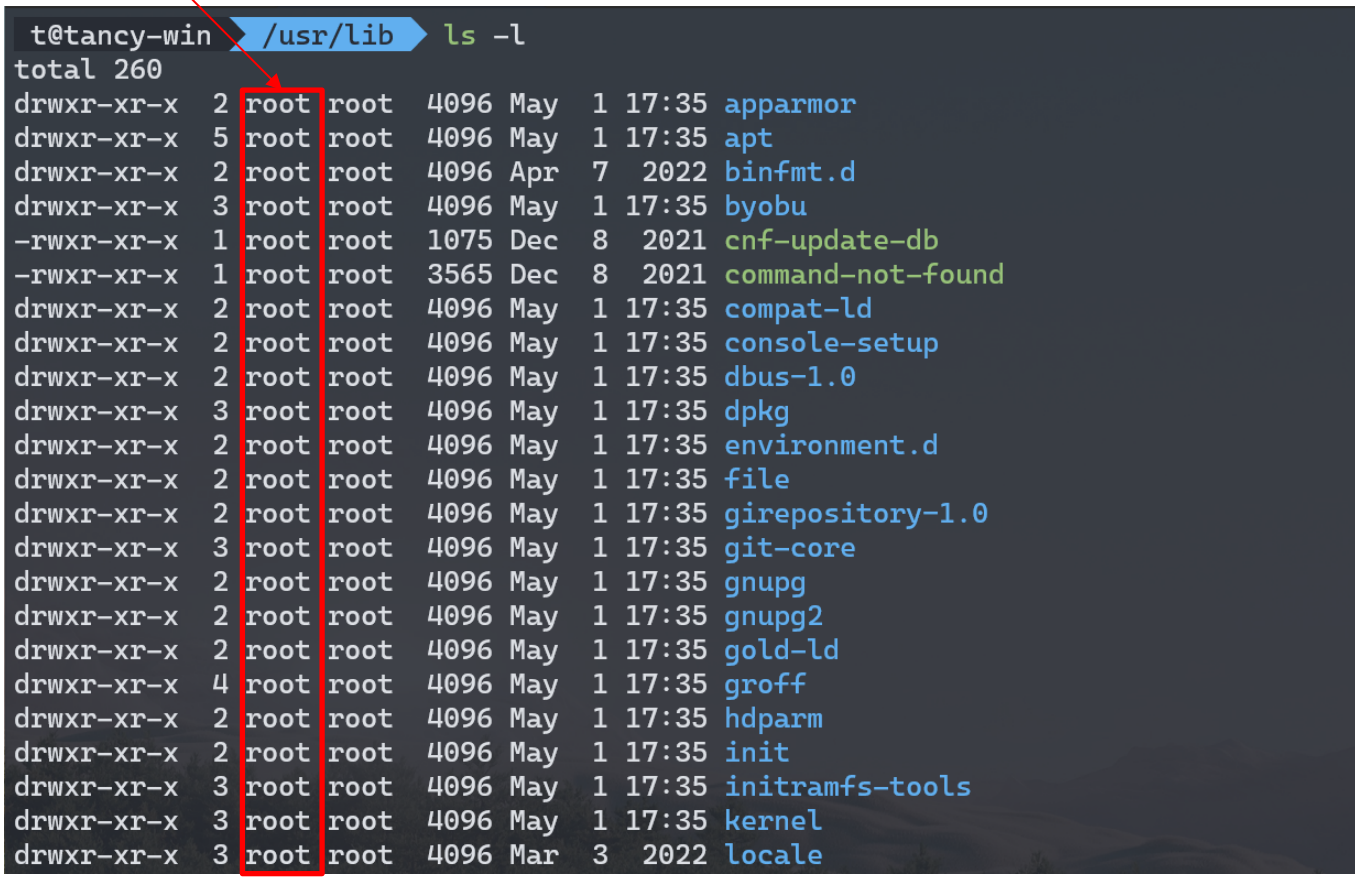


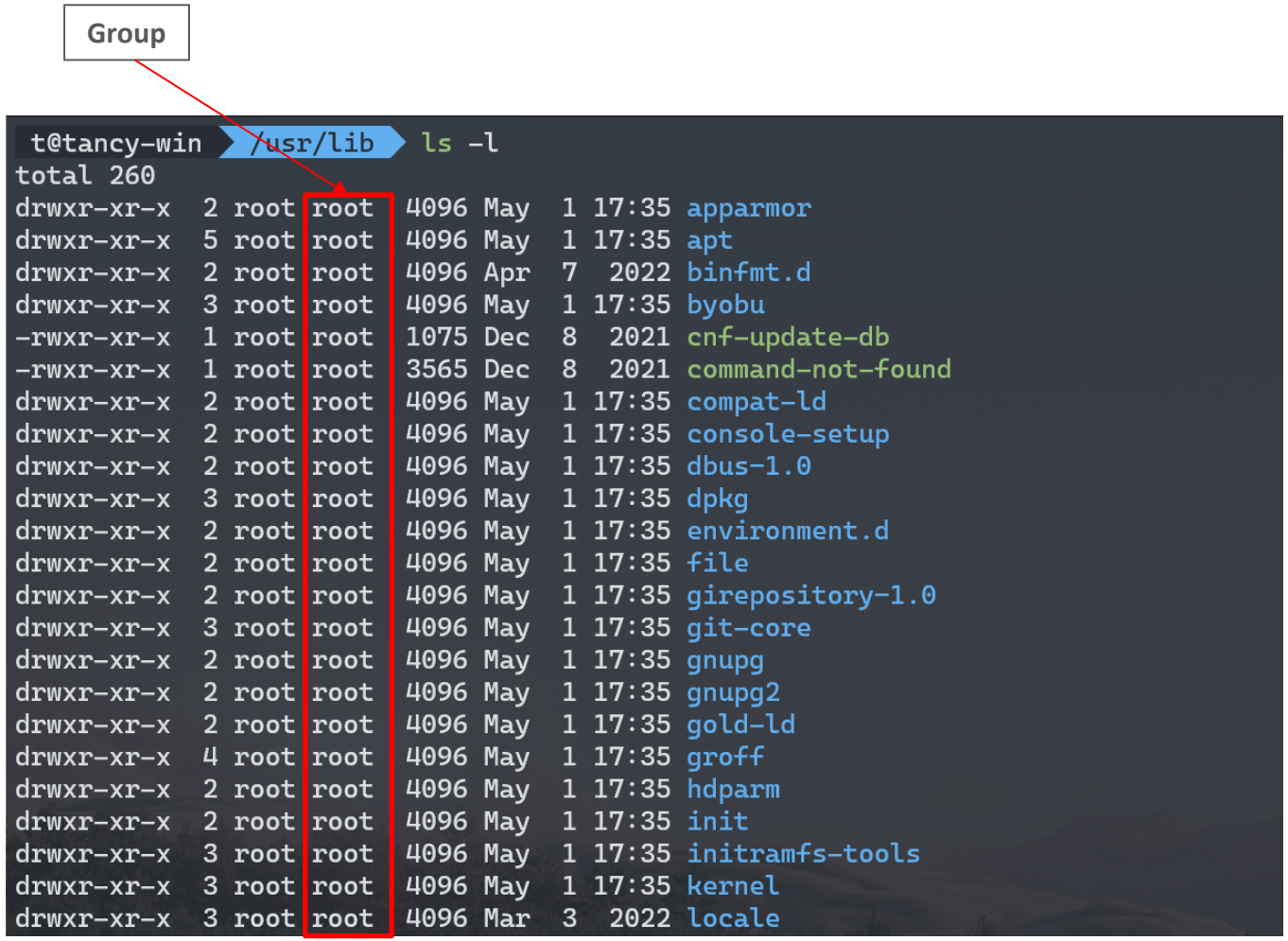
Permissions for owner, group, and others

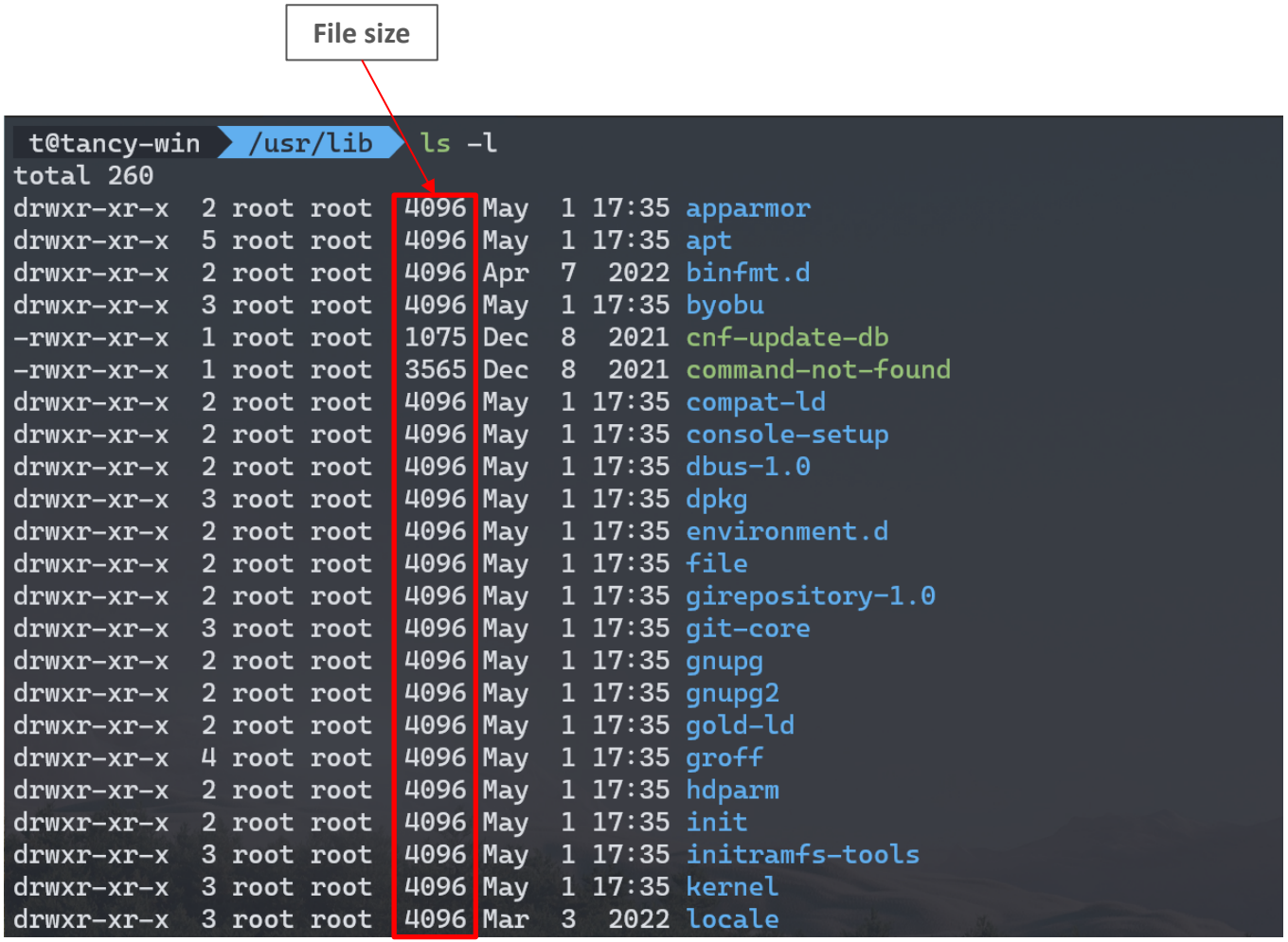


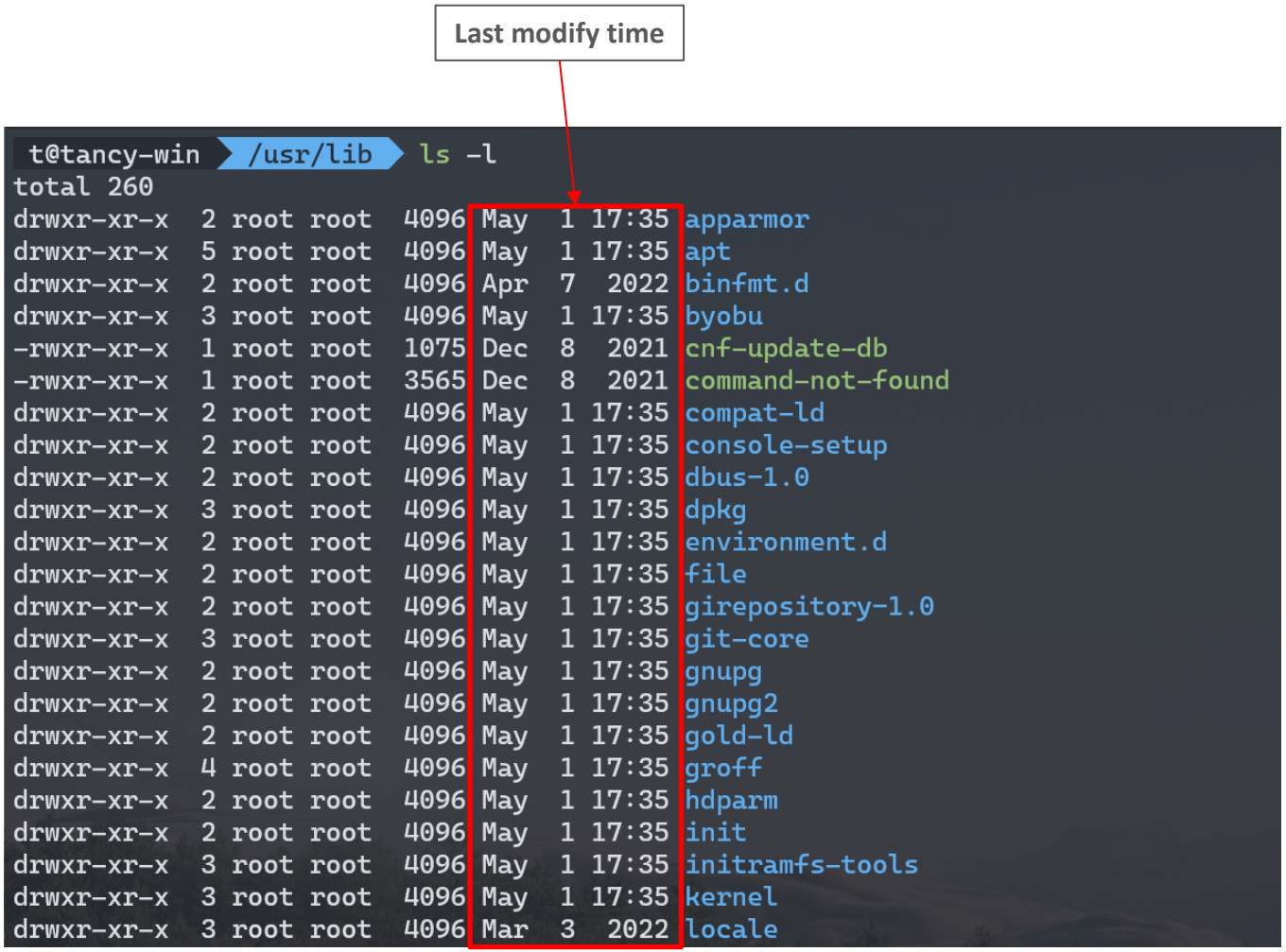

Link count
t@tancy-win /usr/lib ls -l
total 260
drwxr-xr-x 2 root root 4096 May 1 17:35 apparmor
drwxr-xr-x 5 root root 4096 May 1 17:35 apt
drwxr-xr-x 2 root root 4096 Apr 7 2022 binfmt.d
drwxr-xr-x 3 root root 4096 May 1 17:35 byobu
-rwxr-xr-x 1 root root 1075 Dec 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 Dec 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 May 1 17:35 compat-ld
drwxr-xr-x 2 root root 4096 May 1 17:35 console-setup
drwxr-xr-x 2 root root 4096 May 1 17:35 dbus-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 dpkg
drwxr-xr-x 2 root root 4096 May 1 17:35 environment.d
drwxr-xr-x 2 root root 4096 May 1 17:35 file
drwxr-xr-x 2 root root 4096 May 1 17:35 girepository-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 git-core
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg2
drwxr-xr-x 2 root root 4096 May 1 17:35 gold-ld
drwxr-xr-x 4 root root 4096 May 1 17:35 groff
drwxr-xr-x 2 root root 4096 May 1 17:35 hdparm
drwxr-xr-x 2 root root 4096 May 1 17:35 init
drwxr-xr-x 3 root root 4096 May 1 17:35 initramfs-tools
drwxr-xr-x 3 root root 4096 May 1 17:35 kernel
drwxr-xr-x 3 root root 4096 Mar 3 2022 locale

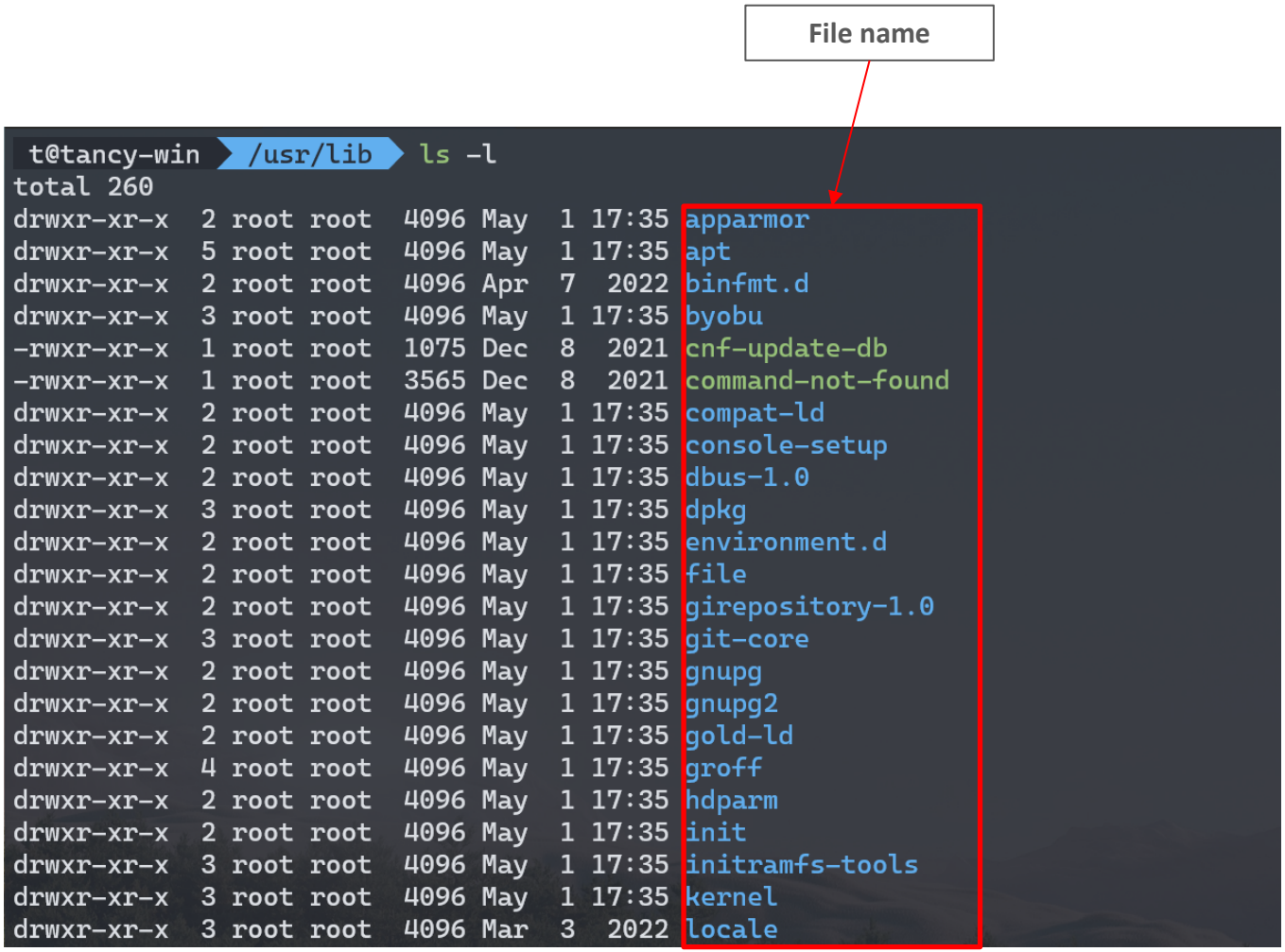
Owner: This field provide info about the creator of the file.




Group
t@tancy-win /usr/lib ls -l
total 260
drwxr-xr-x 2 root root 4096 May 1 17:35 apparmor
drwxr-xr-x 5 root root 4096 May 1 17:35 apt
drwxr-xr-x 2 root root 4096 Apr 7 2022 binfmt.d
drwxr-xr-x 3 root root 4096 May 1 17:35 byobu
-rwxr-xr-x 1 root root 1075 Dec 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 Dec 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 May 1 17:35 compat-ld
drwxr-xr-x 2 root root 4096 May 1 17:35 console-setup
drwxr-xr-x 2 root root 4096 May 1 17:35 dbus-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 dpkg
drwxr-xr-x 2 root root 4096 May 1 17:35 environment.d
drwxr-xr-x 2 root root 4096 May 1 17:35 file
drwxr-xr-x 2 root root 4096 May 1 17:35 girepository-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 git-core
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg2
drwxr-xr-x 2 root root 4096 May 1 17:35 gold-ld
drwxr-xr-x 4 root root 4096 May 1 17:35 groff
drwxr-xr-x 2 root root 4096 May 1 17:35 hdparm
drwxr-xr-x 2 root root 4096 May 1 17:35 init
drwxr-xr-x 3 root root 4096 May 1 17:35 initramfs-tools
drwxr-xr-x 3 root root 4096 May 1 17:35 kernel
drwxr-xr-x 3 root root 4096 Mar 3 2022 locale


File size
t@tancy-win /usr/lib ls -l
total 260
drwxr-xr-x 2 root root 4096 May 1 17:35 apparmor
drwxr-xr-x 5 root root 4096 May 1 17:35 apt
drwxr-xr-x 2 root root 4096 Apr 7 2022 binfmt.d
drwxr-xr-x 3 root root 4096 May 1 17:35 byobu
-rwxr-xr-x 1 root root 1075 Dec 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 Dec 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 May 1 17:35 compat-ld
drwxr-xr-x 2 root root 4096 May 1 17:35 console-setup
drwxr-xr-x 2 root root 4096 May 1 17:35 dbus-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 dpkg
drwxr-xr-x 2 root root 4096 May 1 17:35 environment.d
drwxr-xr-x 2 root root 4096 May 1 17:35 file
drwxr-xr-x 2 root root 4096 May 1 17:35 girepository-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 git-core
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg2
drwxr-xr-x 2 root root 4096 May 1 17:35 gold-ld
drwxr-xr-x 4 root root 4096 May 1 17:35 groff
drwxr-xr-x 2 root root 4096 May 1 17:35 hdparm
drwxr-xr-x 2 root root 4096 May 1 17:35 init
drwxr-xr-x 3 root root 4096 May 1 17:35 initramfs-tools
drwxr-xr-x 3 root root 4096 May 1 17:35 kernel
drwxr-xr-x 3 root root 4096 Mar 3 2022 locale


Last modify time
t@tancy-win /usr/lib ls -l
total 260
drwxr-xr-x 2 root root 4096 May 1 17:35 apparmor
drwxr-xr-x 5 root root 4096 May 1 17:35 apt
drwxr-xr-x 2 root root 4096 Apr 7 2022 binfmt.d
drwxr-xr-x 3 root root 4096 May 1 17:35 byobu
-rwxr-xr-x 1 root root 1075 Dec 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 Dec 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 May 1 17:35 compat-ld
drwxr-xr-x 2 root root 4096 May 1 17:35 console-setup
drwxr-xr-x 2 root root 4096 May 1 17:35 dbus-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 dpkg
drwxr-xr-x 2 root root 4096 May 1 17:35 environment.d
drwxr-xr-x 2 root root 4096 May 1 17:35 file
drwxr-xr-x 2 root root 4096 May 1 17:35 girepository-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 git-core
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg2
drwxr-xr-x 2 root root 4096 May 1 17:35 gold-ld
drwxr-xr-x 4 root root 4096 May 1 17:35 groff
drwxr-xr-x 2 root root 4096 May 1 17:35 hdparm
drwxr-xr-x 2 root root 4096 May 1 17:35 init
drwxr-xr-x 3 root root 4096 May 1 17:35 initramfs-tools
drwxr-xr-x 3 root root 4096 May 1 17:35 kernel
drwxr-xr-x 3 root root 4096 Mar 3 2022 locale


File name
t@tancy-win /usr/lib ls -l
total 260
drwxr-xr-x 2 root root 4096 May 1 17:35 apparmor
drwxr-xr-x 5 root root 4096 May 1 17:35 apt
drwxr-xr-x 2 root root 4096 Apr 7 2022 binfmt.d
drwxr-xr-x 3 root root 4096 May 1 17:35 byobu
-rwxr-xr-x 1 root root 1075 Dec 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 Dec 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 May 1 17:35 compat-ld
drwxr-xr-x 2 root root 4096 May 1 17:35 console-setup
drwxr-xr-x 2 root root 4096 May 1 17:35 dbus-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 dpkg
drwxr-xr-x 2 root root 4096 May 1 17:35 environment.d
drwxr-xr-x 2 root root 4096 May 1 17:35 file
drwxr-xr-x 2 root root 4096 May 1 17:35 girepository-1.0
drwxr-xr-x 3 root root 4096 May 1 17:35 git-core
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg
drwxr-xr-x 2 root root 4096 May 1 17:35 gnupg2
drwxr-xr-x 2 root root 4096 May 1 17:35 gold-ld
drwxr-xr-x 4 root root 4096 May 1 17:35 groff
drwxr-xr-x 2 root root 4096 May 1 17:35 hdparm
drwxr-xr-x 2 root root 4096 May 1 17:35 init
drwxr-xr-x 3 root root 4096 May 1 17:35 initramfs-tools
drwxr-xr-x 3 root root 4096 May 1 17:35 kernel
drwxr-xr-x 3 root root 4096 Mar 3 2022 locale

Mandatory Access Control

- In mandatory access control (MAC) users are granted privileges, which they cannot control or change
  - useful for military applications
  - useful for regular operating systems
- DAC does not protect against
  - Malware
  - Software bugs
  - Malicious local users
- The SELinux enhancement to the Linux kernel implements the Mandator Access Control (MAC) policy, which allows you to define a security policy that provides granular permissions for all users, programs, processes, files, and devices

MAC in Operating Systems

- The need for MAC
  - host compromise by network-based attacks is the root cause of many serious security problems
    - worm, botnet, DDoS, phishing, spamming
  - hosts can be easily compromised
    - programs contain exploitable bugs
    - DAC mechanisms in OSs were not designed to take buggy software in mind
  - adding MAC to OSs is essential to deal with host compromise
    - last line of defense when everything else fails
- In MAC a system-wide security policy restricts access rights of subjects

Combining MAC and DAC

- It is common to combine mandatory and discretionary access control in complex systems
  - modern operating systems is one significant example
- MAC and DAC are also combined in older models that implement multilevel security (for military-style security classes)
  - Bell-Lapadula confidentiality model (1973)
  - Biba integrity model (1977)

Questions?

Security of Discretionary Access Control

- 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

Safety of DAC Models

- Assume we have an access control matrix

	f_a	f_b	f_c
s_a	*own, r, w*	*r*	*r*
s_b	*r*	*own, r, w*	*r*
s_c	*r*	*r*	*own, r, w*

  - is it safe with respect to r?
  - is it safe with respect to w?
  - what if we disallow granting rights? object deletion?
- Safety of many useful models is undecidable
  - safety of certain models is tractable, but they tend not to apply to real world

Decidability of DAC Models

- Decidable
  - we are 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
  - we are now 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
  - how applicable is this?
- And safety is still undecidable for practical models