CS 4910: Intro to Computer Security

Access Control I

Instructor: Xi Tan

What we already know

- Crypto tools
  - Symmetric crypto
  - Public-key crypto
  - Digital signature
- Authentication
  - Concepts
  - Three types of authentication
    - Password-based
    - Token-based
    - Biometric-based

Access Control

- Access control principles
  - access control matrices
  - access control lists
  - capability tickets

- Types of access control
  - discretionary access control (DAC)
  - mandatory access control (MAC)
  - role-based access control (RBAC)
  - attribute-based access control (ABAC)

Access Control Basics

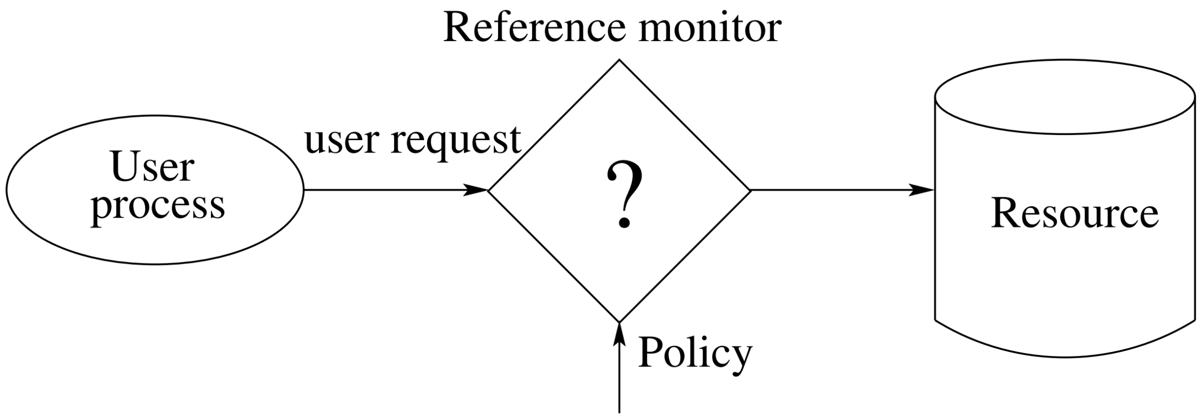
- What is access control?
  - prevention of an unauthorized use of a resource or use in an unauthorized manner
- In some sense, all of security is concerned with access control
- We look at a more specific notion of access control model
- An access control model specifies who is allowed to access what resource and what type of access is permitted
  - it may also specify **when** access is permitted
- What makes it hard?
  - **interaction** between different types of access

Related Security Concepts

- In a broader context, access control is related to the following concepts
  - authentication, identity and credential management
    - creation, maintenance, and verification of user or entity identity and/or credentials
  - authorization and information flow
    - granting rights or privileges based on established trust assumptions and imposing controls on information flow
  - audit and integrity protection
    - system monitoring to ensure proper use of resources and compliance with policies
    - detection of breaches in security and taking corresponding actions and/or making recommendations

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 Principles

objects, *O*,

- a set of resources to be protected
- directories, files, devices, peripherals, even facilities

subjects, *S*

- access to the resources
- each subject can have a number of attributes (name, role, groups)
  - each subject is normally accountable for its actions

Access right or privilege describes the type of access

- e.g., read, write, execute, delete, search

Access control requirements form rules

e.g., subject *s* has *read* access to object *o*

Access Control Matrix

- The rules can be represented as an access control matrix
- Example

	Internal	Local	Long distance	International
Public	CRT			
Students	CRT	CRT	R	R
Staff	CRT	CRT	CRT	R
Administration	CRT	CRT	CRT	CRT

  - C = call, R = receive, T = transfer
- Often access control matrices are sparse and can instead be represented as access control lists (ACLs)

Access Control Lists

- In ACLs each object has a list of subjects authorized to access it and their types of access
    - for each object, a column of the access control matrix is stored
- Example of ACLs for previous system

Internal:	Public/CRT, Students/CRT, Staff/CRT, Administration/CRT
Local:	Students/CRT, Staff/CRT, Administration/CRT
Long distance:	Students/R, Staff/CRT, Administration/CRT
International:	Students/R, Staff/R, Administration/CRT

Capability Lists

- With ACLs, it is hard to determine what privileges a subject has
- We can gather information about subject privileges in so-called capability lists
  - for each subject, store a row of the access control matrix
- Example

public:	Internal/CRT
students:	Internal/CRT, Local/CRT, Long dist/R, International/R
Staff:	Internal/CRT, Local/CRT, Long dist/CRT, International/R
Administration:	Internal/CRT, Local/CRT, Long dist/CRT, Intl/CRT

- Each user has a number of capability tickets and might be allowed to loan or give them to others

Access Control Triples

- To address drawbacks of all previous representations, we can have a table with (s, o, a) triples
    - is not sparse like access control matrices
    - sort by objects to obtain ACLs
    - sort by subjects to obtain capability lists

Subject	Access	Object
Public	C	Internal
Public	R	Internal
Public	T	Internal
Students	C	Internal
…	…	…
Administration	T	Administration

    - This data structure is commonly used in relational DBMSs (database management systems)

ACLs vs. Capability Lists

- The choice of ACLs vs capability lists affects many aspects of the system
  - ACL systems need a namespace for both objects and subjects, while a capability ticket can serve both to designate a resource and to provide authority
  - procedures such as access review and revocation are superior on a per-object basis in ACL systems and on per-subject basis in capability systems
  - ACL systems require authentication of subjects, while capability systems require unforgeability and control of propagation of capabilities
- Most real-world OSs use ACLs
  - Linux

Next

- Access control principles
  - access control matrices
  - access control lists
  - capability tickets
- Types of access control
  - discretionary access control (DAC)
  - mandatory access control (MAC)
  - role-based access control (RBAC)
  - attribute-based access control (ABAC)

Discretionary Access Control

- Discretionary access control (DAC) has provisions for allowing subjects to **grant privileges** to other subjects
- as a result, the access control matrix A can change
- Let triple (s, o, a) represent an access right
- At time i, the state X_i of the system is characterized by (S_i , O_i , A_i)
- Transition t_i takes the system from state X_i to X_{i+1}
  - a single transition $X_i \vdash ti X_{i+1}$
  - series of transitions $X \vdash * Y$

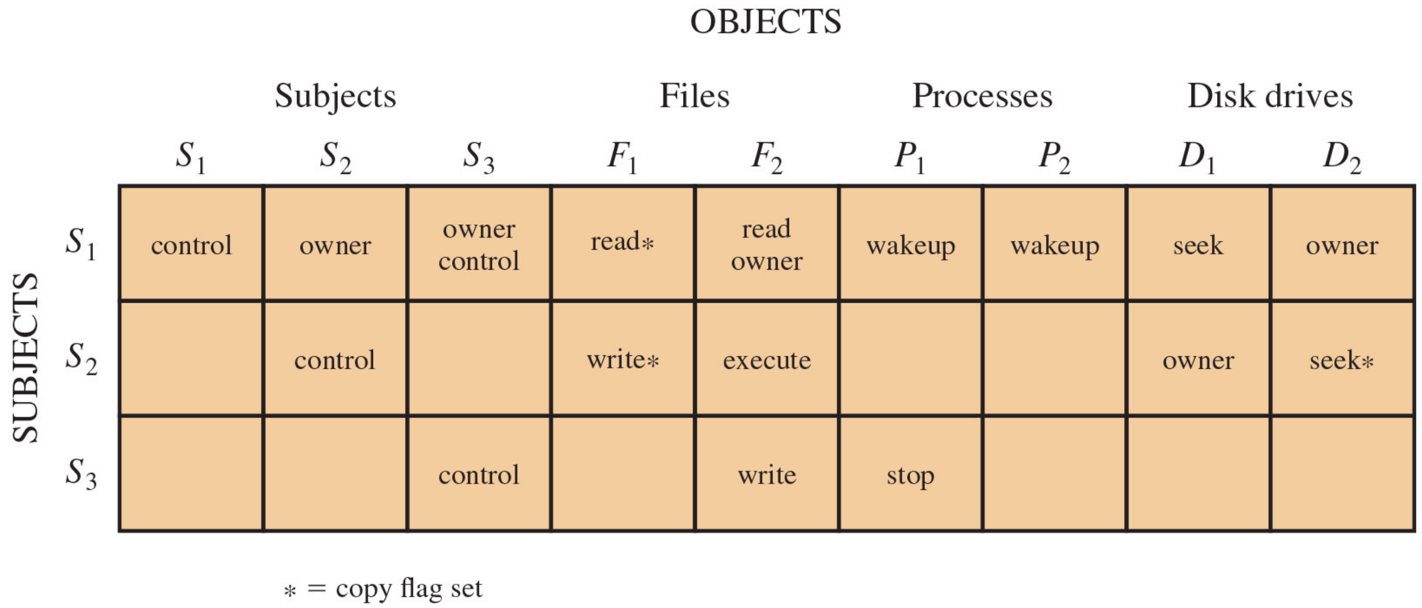
Discretionary Access Control

- 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	Subjects			Files		Processes		Disk drives	
	S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
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
- Grant right gives rise to the principle of attenuation of privilege:
  - a subject may not give rights it does not possess
- Each particular model has a set of rules that define acceptable modifications to the access control matrix

Discretionary Access Control

- Primitive commands
  - create object o (with no access)
    - $S_{i+1} = S_i,\ O_{i+1} = O_i \cup \{o\},\ \forall x \in S_{i+1}, A_{i+1}[x, o] = \emptyset,$
      $\forall x \in S_{i+1}, \forall y \in O_i, A_{i+1}[x, y] = A_i[x, y]$
  - create subject s (with no access)
    - add s to the set of subjects and objects, set relevant access to $\emptyset$
  - add right r to object o for subject s
    - $A_{i+1}[s, o] = A_i[s, o] \cup \{r\}$, everything else stays the same
  - delete right r from $A_i[s, o]$
  - destroy subject s
  - destroy object o

Discretionary Access Control

- Building more useful commands
  - s creates object o
    - create object o with no access
    - add right own to object o for subject s
  - s adds right r to object o for subject $s`$
    - if $(r* \in A_i\ [s, o]$ or $own \in A_i\ [s, o])$, then

$$A_{i+1}[s`, o] = A_i\ [s`, o] \cup \{r\}$$

    - leave the rest unchanged
  - s deletes object o
    - if ($own \in A_i\ [s, o]$), then remove all access rights $\forall x \in S_i$ from $A[x, o]$ and destroy 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

- 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.

DAC in Unix File System

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

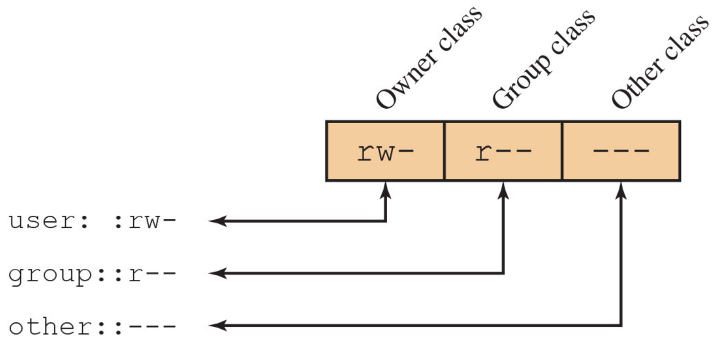
- **Read** – The Read permission refers to a user's capability to read the contents of the file.

- **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.

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

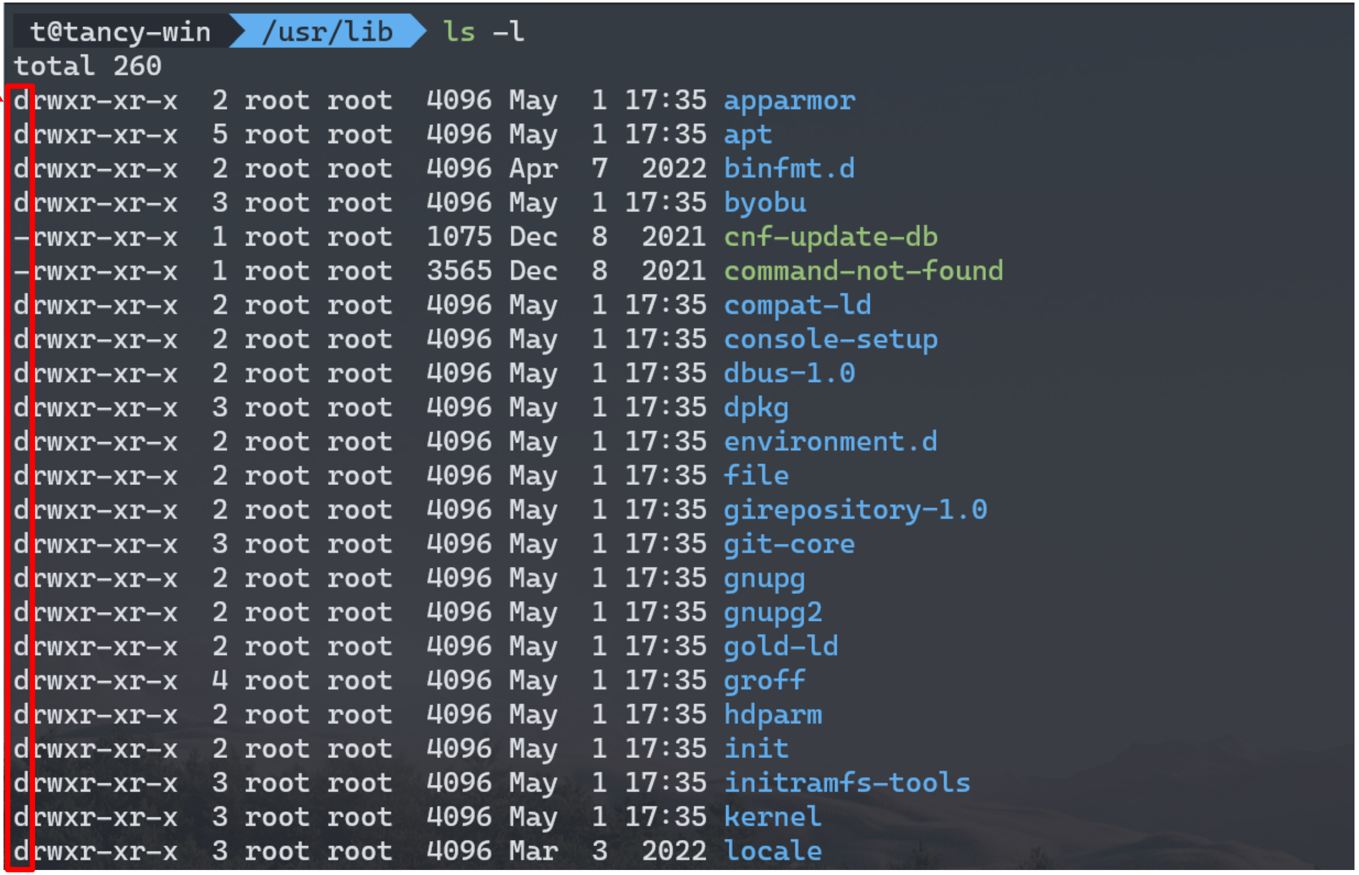


(a) Traditional UNIX approach (minimal access control list)

DAC in Unix File System

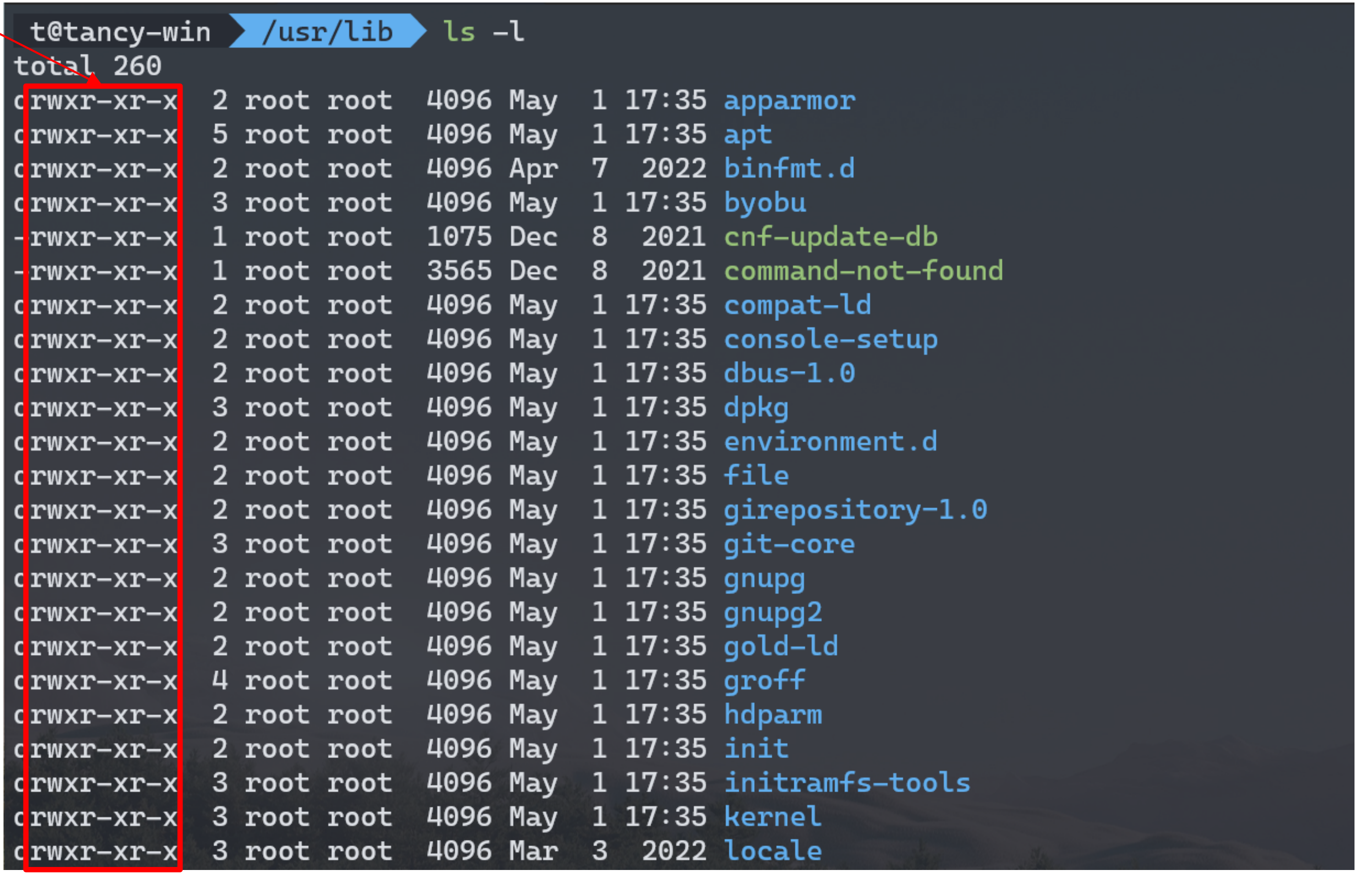
- DAC is implemented by using commands *chmod* and *chown*
- A special user “superuser” or “root” is exempt from regular access control constraints
- Many Unix systems support additional ACLs
  - owner (or administrator) can add to a file users or groups with specific access privileges
  - the permissions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

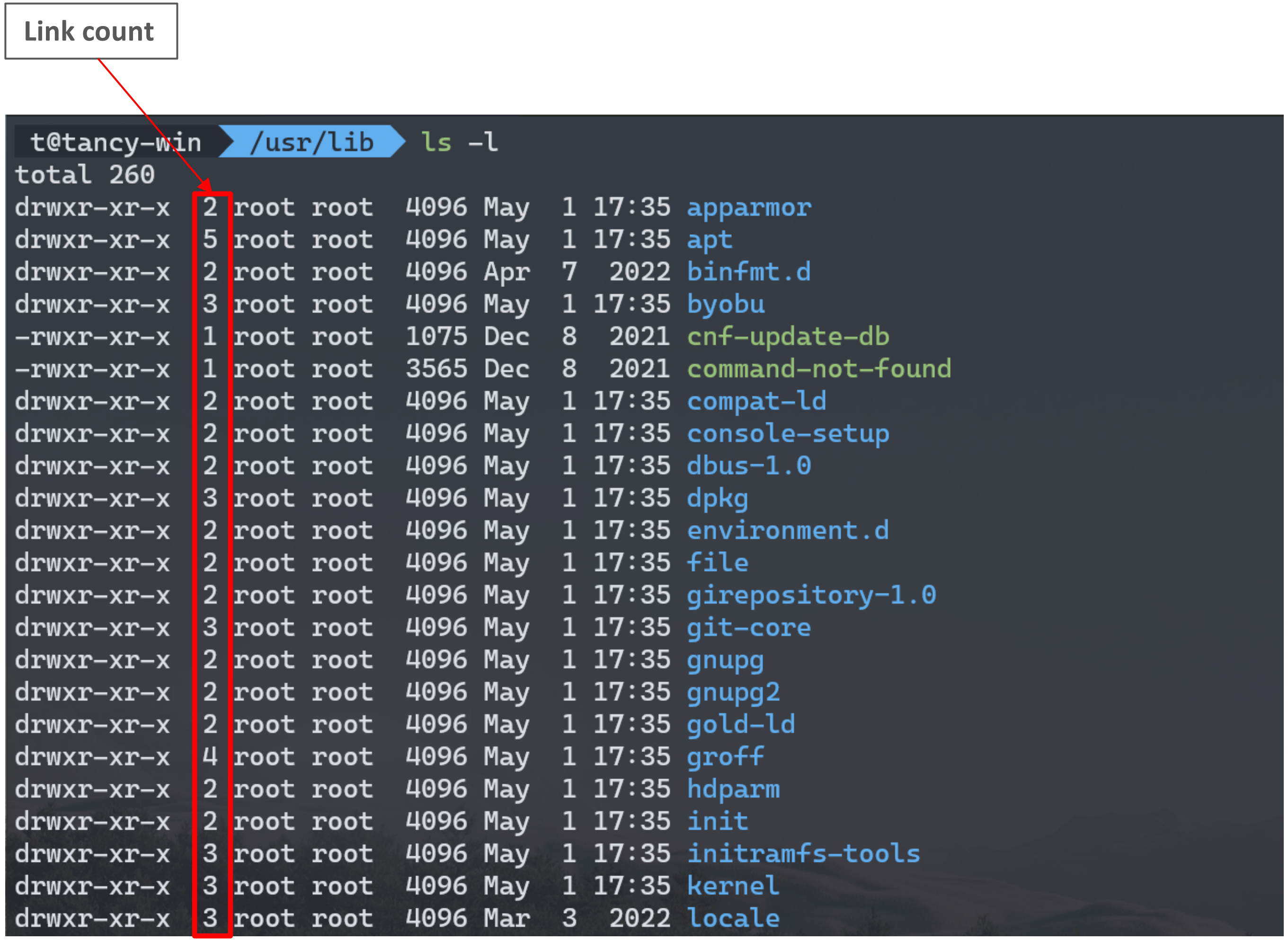


```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root  4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root  4096 May  1 17:35 apt
drwxr-xr-x  2 root root  4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root  4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root  1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```

Permissions for owner, group, and others

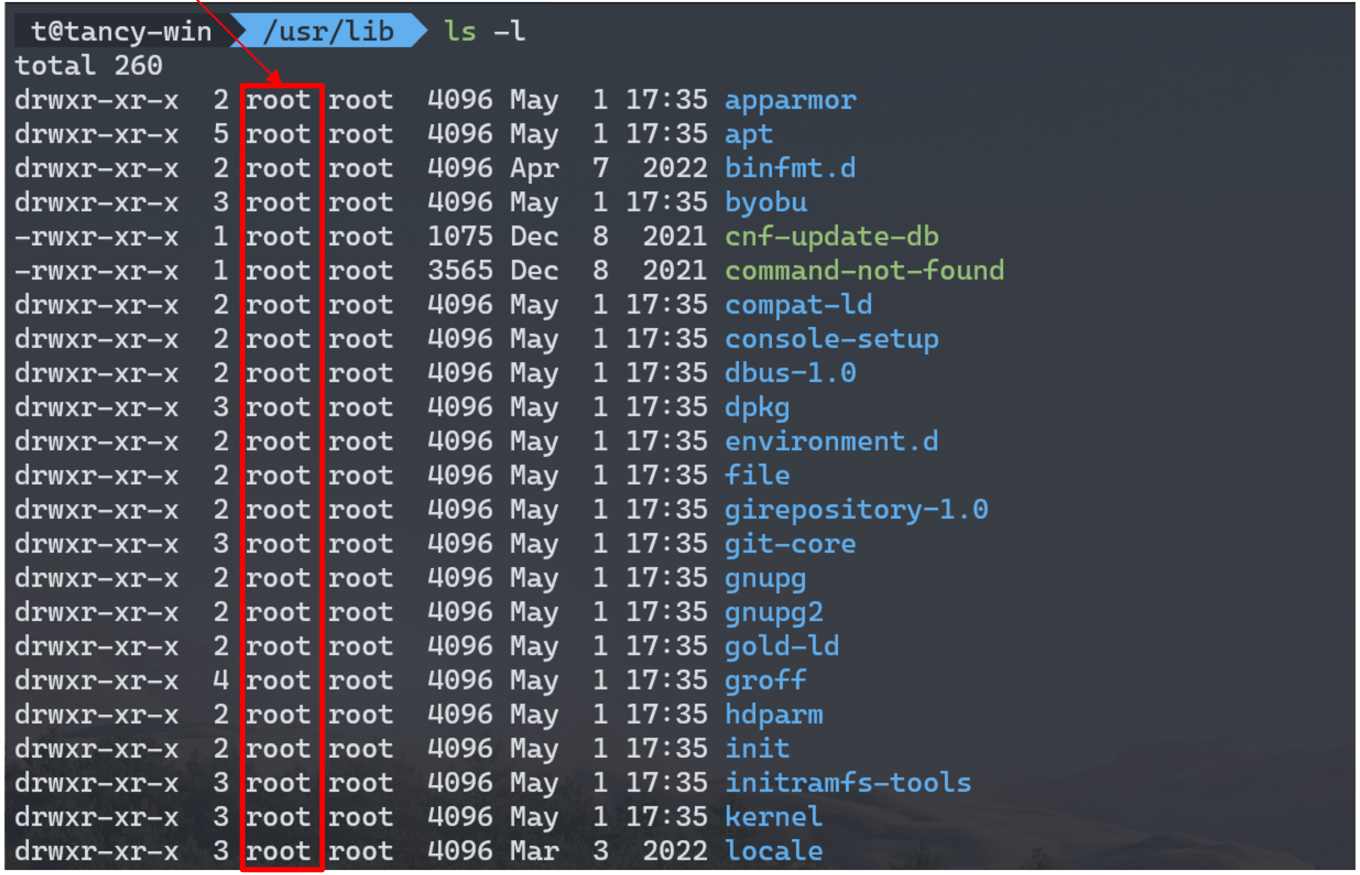


```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root  4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root  4096 May  1 17:35 apt
drwxr-xr-x  2 root root  4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root  4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root  1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```

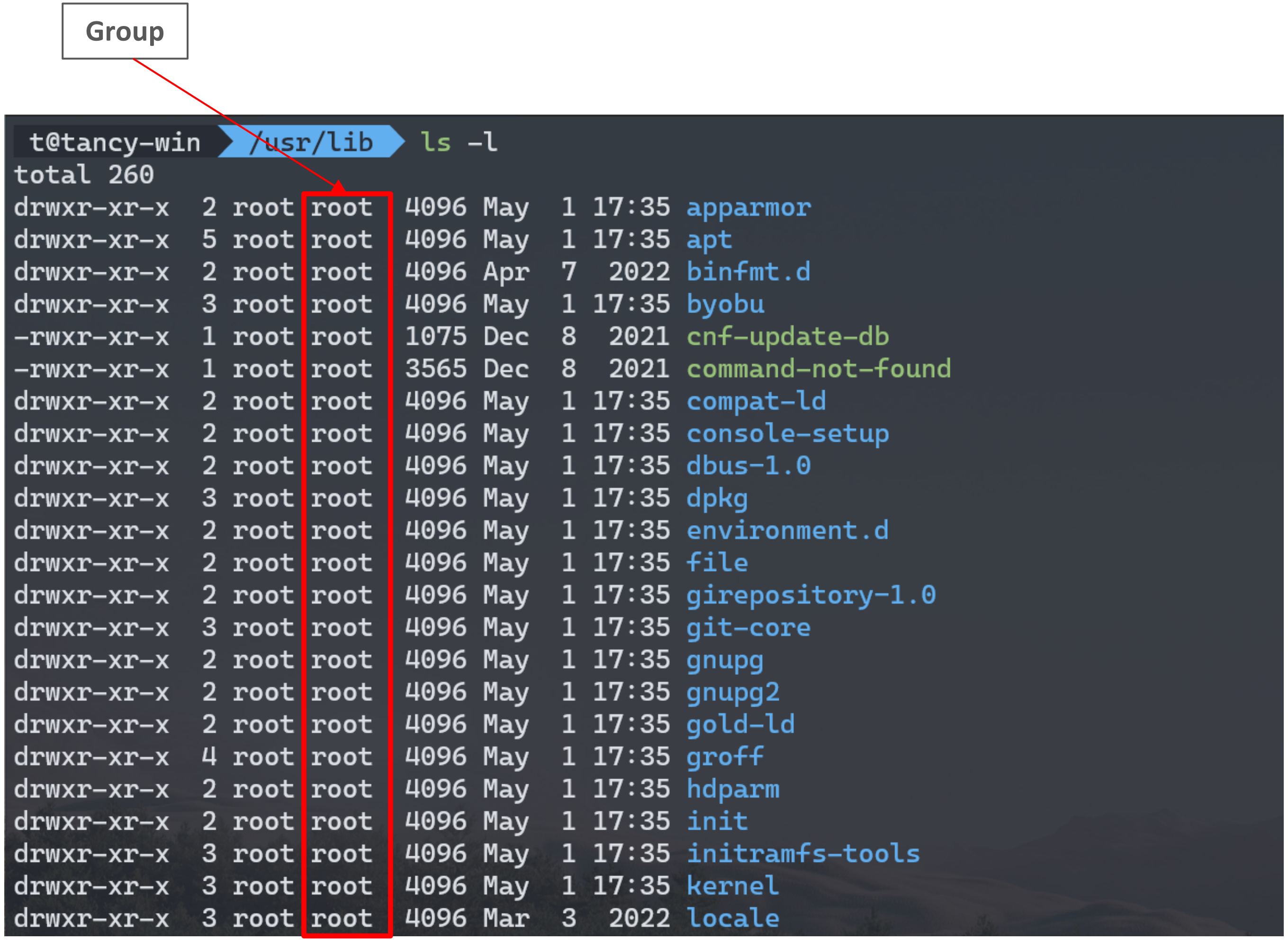


```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root  4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root  4096 May  1 17:35 apt
drwxr-xr-x  2 root root  4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root  4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root  1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```

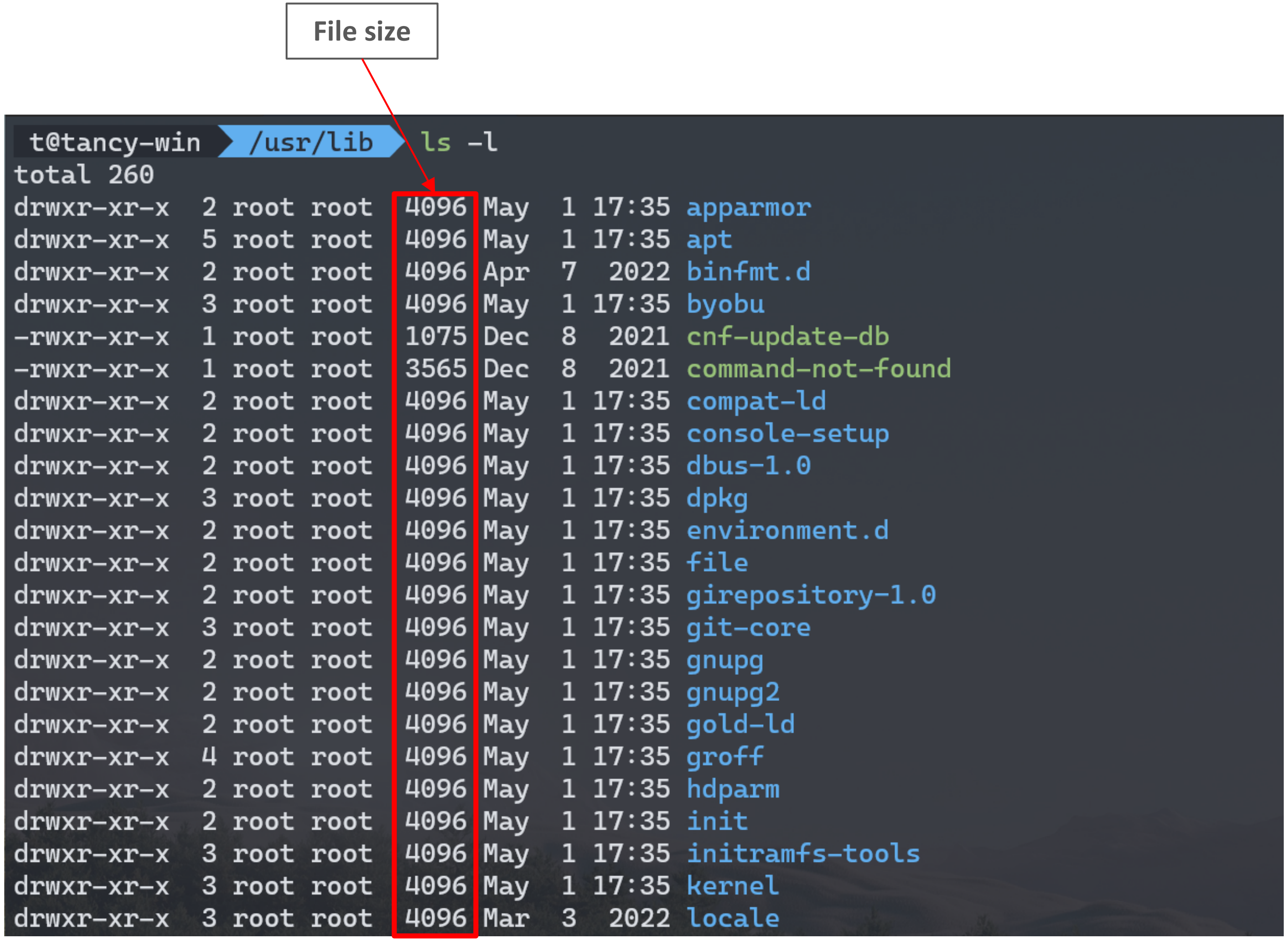
Owner: This field provide info about the creator of the file.



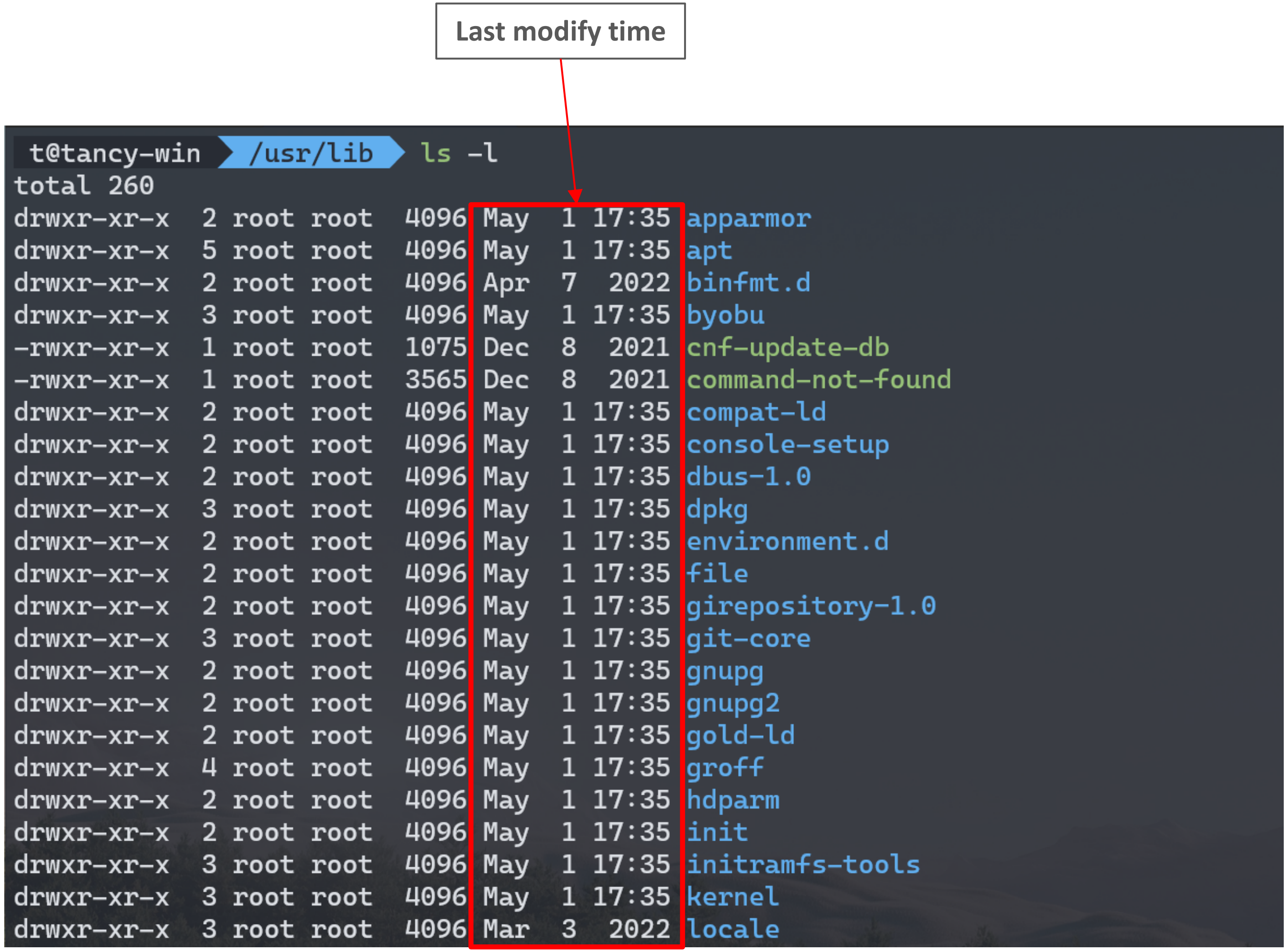
```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root 4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root 4096 May  1 17:35 apt
drwxr-xr-x  2 root root 4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root 4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root 1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```



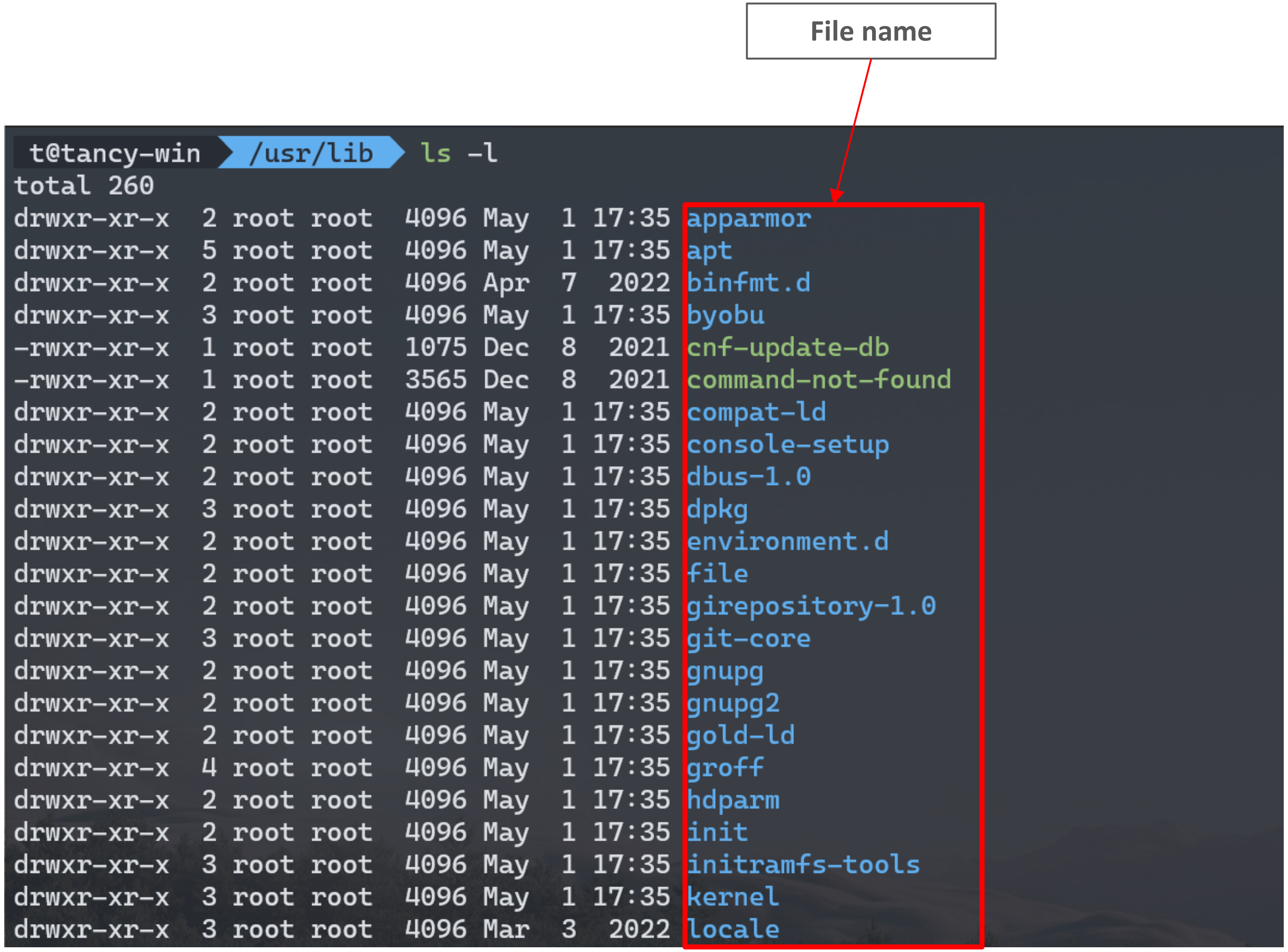
```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root 4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root 4096 May  1 17:35 apt
drwxr-xr-x  2 root root 4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root 4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root 1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```



```
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
```



```
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
```



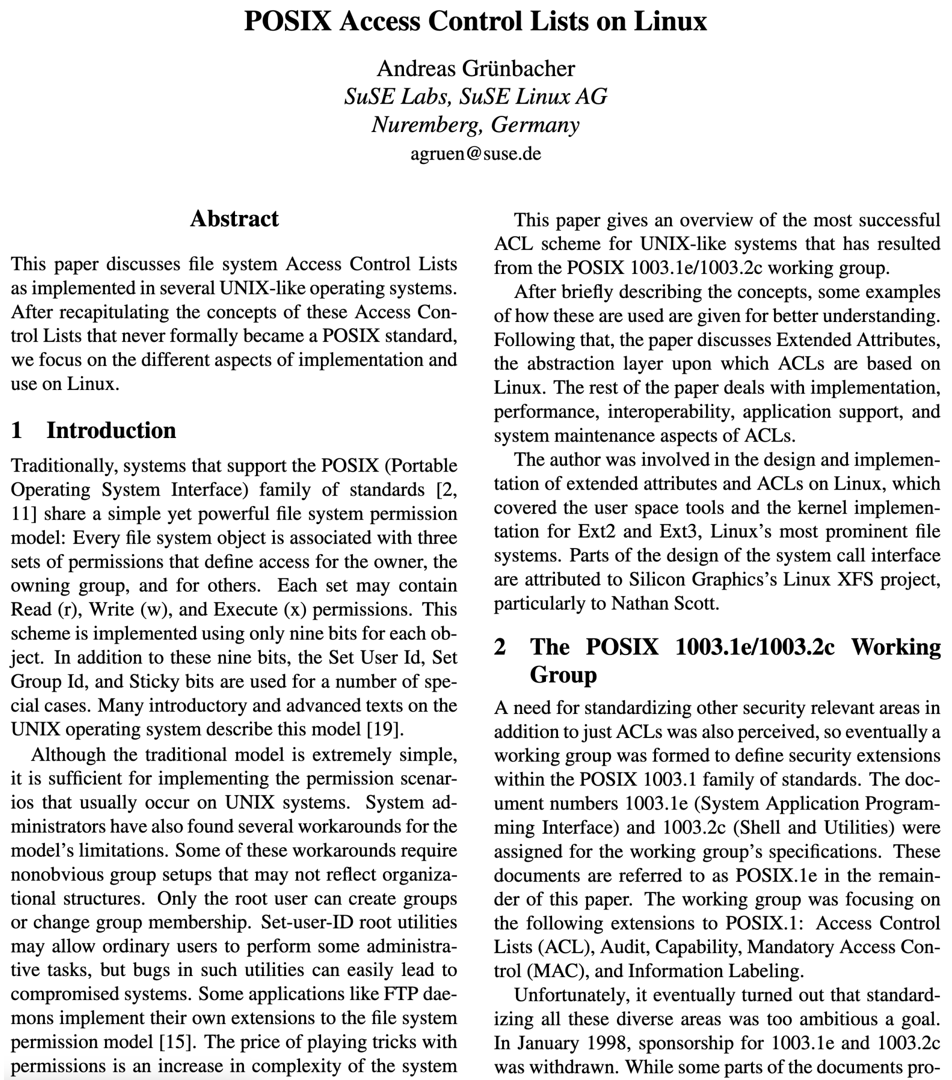
```
t@tancy-win  /usr/lib  ls -l
total 260
drwxr-xr-x  2 root root  4096 May  1 17:35 apparmor
drwxr-xr-x  5 root root  4096 May  1 17:35 apt
drwxr-xr-x  2 root root  4096 Apr  7  2022 binfmt.d
drwxr-xr-x  3 root root  4096 May  1 17:35 byobu
-rwxr-xr-x  1 root root  1075 Dec  8  2021 cnf-update-db
-rwxr-xr-x  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
```

Is Linux File Permission an ACL?

- Linux file permissions are not considered an Access Control List (ACL) because they only define three types of permissions for three categories of users (owner, group, and others).

- ACLs, on the other hand, allow for more fine-grained access control by specifying permissions for individual users or groups.

- However, Linux does support ACLs through the POSIX ACL system, which allows for more complex permission structures beyond the basic file permissions.



POSIX Access Control Lists on Linux

Andreas Grünbacher
SuSE Labs, SuSE Linux AG
Nuremberg, Germany
agruen@suse.de

Abstract

This paper discusses file system Access Control Lists as implemented in several UNIX-like operating systems. After recapitulating the concepts of these Access Control Lists that never formally became a POSIX standard, we focus on the different aspects of implementation and use on Linux.

1 Introduction

Traditionally, systems that support the POSIX (Portable Operating System Interface) family of standards [2, 11] share a simple yet powerful file system permission model: Every file system object is associated with three sets of permissions that define access for the owner, the owning group, and for others. Each set may contain Read (r), Write (w), and Execute (x) permissions. This scheme is implemented using only nine bits for each object. In addition to these nine bits, the Set User Id, Set Group Id, and Sticky bits are used for a number of special cases. Many introductory and advanced texts on the UNIX operating system describe this model [19].

Although the traditional model is extremely simple, it is sufficient for implementing the permission scenarios that usually occur on UNIX systems. System administrators have also found several workarounds for the model's limitations. Some of these workarounds require nonobvious group setups that may not reflect organizational structures. Only the root user can create groups or change group membership. Set-user-ID root utilities may allow ordinary users to perform some administrative tasks, but bugs in such utilities can easily lead to compromised systems. Some applications like FTP daemons implement their own extensions to the file system permission model [15]. The price of playing tricks with permissions is an increase in complexity of the system

This paper gives an overview of the most successful ACL scheme for UNIX-like systems that has resulted from the POSIX 1003.1e/1003.2c working group.

After briefly describing the concepts, some examples of how these are used are given for better understanding. Following that, the paper discusses Extended Attributes, the abstraction layer upon which ACLs are based on Linux. The rest of the paper deals with implementation, performance, interoperability, application support, and system maintenance aspects of ACLs.

The author was involved in the design and implementation of extended attributes and ACLs on Linux, which covered the user space tools and the kernel implementation for Ext2 and Ext3, Linux's most prominent file systems. Parts of the design of the system call interface are attributed to Silicon Graphics's Linux XFS project, particularly to Nathan Scott.

2 The POSIX 1003.1e/1003.2c Working Group

A need for standardizing other security relevant areas in addition to just ACLs was also perceived, so eventually a working group was formed to define security extensions within the POSIX 1003.1 family of standards. The document numbers 1003.1e (System Application Programming Interface) and 1003.2c (Shell and Utilities) were assigned for the working group's specifications. These documents are referred to as POSIX.1e in the remainder of this paper. The working group was focusing on the following extensions to POSIX.1: Access Control Lists (ACL), Audit, Capability, Mandatory Access Control (MAC), and Information Labeling.

Unfortunately, it eventually turned out that standardizing all these diverse areas was too ambitious a goal. In January 1998, sponsorship for 1003.1e and 1003.2c was withdrawn. While some parts of the documents pro-

USENIX ATC 2003

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

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)

Summary

- Access control is central in providing an adequate level of security
- Access control rights can be specified in the form of
  - access control matrix
  - access control lists
  - capability tickets
  - access control tables
- Types of access control
  - already covered DAC and MAC
  - will look at role-based access control (RBAC) and attribute-based access control (ABAC)