Lecture 2

Friday, 9 September 2022 15:28

discretionary access control

security problem

access control is ...

A C is used in computer systems in

- application
- middleware
- OS
- memory protection

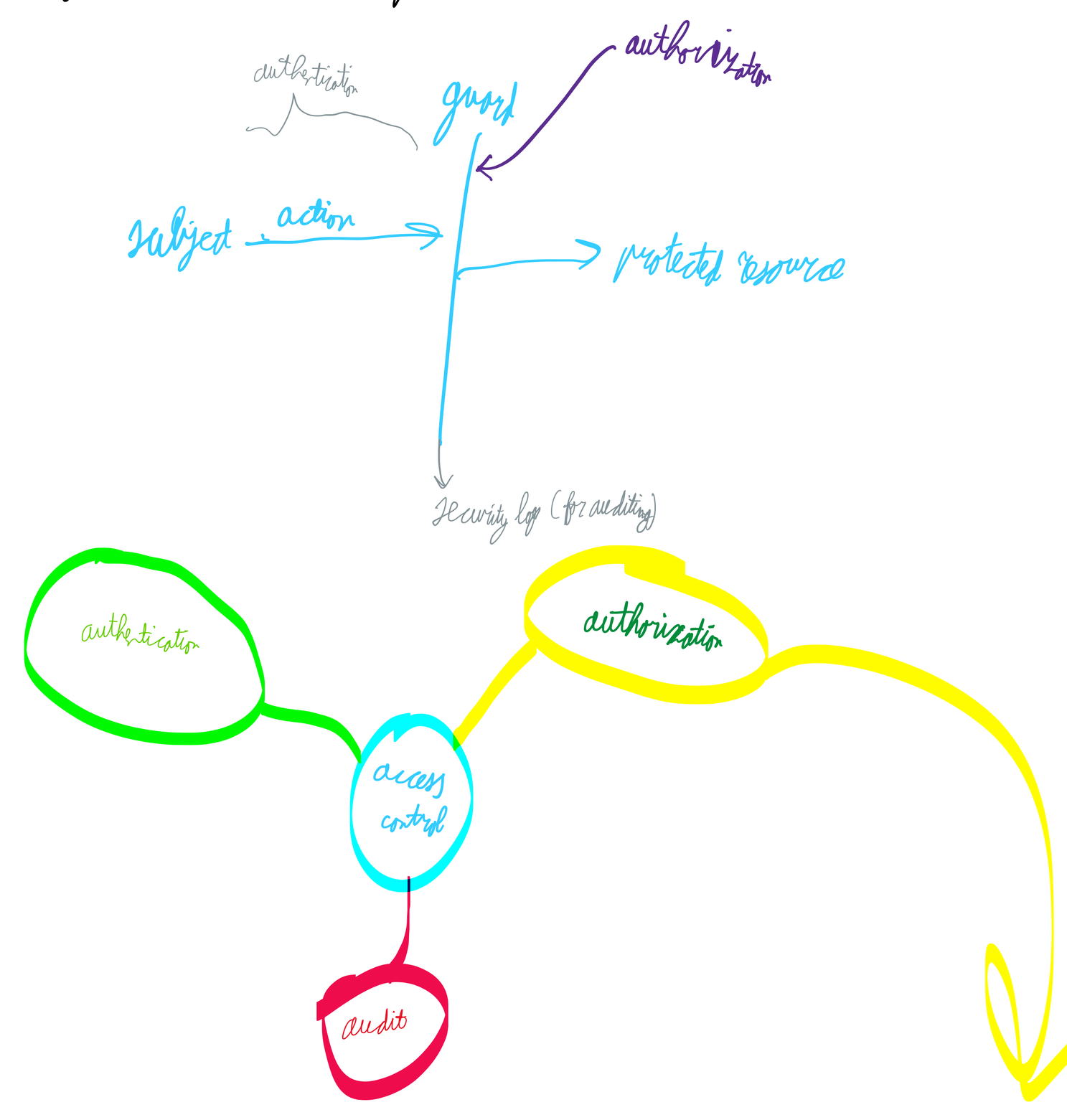
computer security objectives:

confidentiality — includes aspects of privacy

integrity

availability

A C focuses on confidentiality + integrity



exists on multiple levels e.g. hardware, network, middleware, network, OS, application level

access control is domain specific

→ objects and subjects vary

- policy : access requirements
- model : formal representation of policy
- mechanism : low-level implementation

different models allow specification of policies at different levels of granularity

e.g. you don't check per database row

question: who can assign permissions?

1. mandatory access control → central authority/policy
2. discretionary access control → owner's discretion

Principles of access control:

1. Economy of mechanism
   a. Simple design
2. Fail-safe default
   a. Default is no access
3. Complete mediation
   a. Every access to every object must be checked
4. Open design
   a. Security does not depend on secrecy of mechanism
5. Separation of privilege
   a. System that requires 2 keys is more robust than system which requires 1 key
6. Least privilege
   a. No unnecessary privileges should be assigned
7. ???
   a. ???

Challenges:

- Can we express all requirements?
- Does the system not significantly impact performance?
- Is every action checked by the guard and covered by the policy?
- Does the mechanism correctly enforce the policy?

Discretionary access control

Users have full control over objects they create and can pass/delegate their privileges to other users.
Access rules establish which actions can be performed by what subject on what object
Granting/revocation of privileges is regulated by an administrative policy
Access to resources is based on the identity of users.

Lampson model

Set of subjects S, set of objects O, access matrix A (S rows × O columns)
Cells of the matrix contain rights, which can include access, delegation and ??? rights.

Note that the 'own' right does not imply all rights; rights have to be assigned explicitly.

Access matrices tend to be large and sparse. Alternative implementations include authorization tables (storing only non-null entries), access control lists, where the matrix is stored by column, or capability lists, which store the matrix by row.

In access control lists, rights are stored close to resources. The main downside is that it is difficult to audit the rights a user has.
In capability lists, rights are stored close to users. While this makes it easy to audit a user's rights, it also makes it difficult to revoke rights assigned to users (because the file does not have information on which users have permissions for it).

Subjects exercise rights (and are *subject* to policy), while objects have rights exercised on them.

$* perm$ is a right that not only allows the right $perm$, but also allows the user to delegate the right $perm$ to others.

State transitions are described by **commands**. Command structure:

```
Command c(x1, …, xk)
    if x_r_1 in A[x_s_1, x_o_1]
    And x_r_2 in A[x_s_2, x_o_2
    Then
    Confer …
    Confer …
End
```

Copy flag (*): subject can transfer privilege to others
Transfer-only flag (+): subject can transfer privilege to others (including the flag on it), but they lose the privilege when they do so.

Exercise: write a command which allows a process p to create a new process q where parent and child processes can signal (read/write) eaach other.

```
Command create_process(p,q)
    Create subject q
    Enter own into A[p,q]
    Enter r into A[p,q]
    Enter w into A[p,q]
    Enter r into A[q,p]
    Enter w into A[q,p]
End
```

Exercise: compute the access matrix that results from the following initial state by executing the sequence of commands α defined as follows: (see slides for commands & initial state)

	File 1	File 2	File 3
Alice			own
Bob	own		
Charlie		own	
David			+read

Since file 3 already exists when Charlie attempts to create it, that creation attempt will fail. After that, since he is not the owner, he cannot confer a right on it.
The transfer-only command moves the +read right from Bob to David.

Even though Charlie is the owner of file 2, he cannot transfer the read right to Alice (since he does not have the *read right); if he wanted Alice to have this right, he should have conferred it.

safety problem: how can we determine whether a system is secure?

is there an algorithm to verify security?

a state leaks a right r if it is possible to enter the right r into a matrix cell which it wasn't before

a system is secure if ...

safety problem: given initial state, is there a sequence of commands leaking the right → general case: undecidable

↳ reducible to $HALT_{TM}$

without delete/destroy → still undecidable

without create → PSPACE-complete

mono-operational system: decidable

↓ each command consists of a single operation

in the context of e.g. file sharing, leaks aren't necessarily bad...

	S_1	S_2	S_3	S_4	S_5	O
S_1						read
S_2						
S_3						own
S_4						*read
S_5						read

→ this has leaked, so this sequence of commands does leak read privilege

the more expressive the security model is, the more difficult it is to verify security

main limitation of DAC: constraints are imposed on direct access

↳ e.g. copying data into another file makes rights 'different' (on the new file)

Trojan horses ...