Authorization

Slides by Hussain Almohri

Authorization vs. Authentication

- When principals are properly authenticated, systems must separate their roles and privileges.
- Major models of authorization define subjects (principals) and objects, and their relationships.
- Authenticating a principal does not automatically imply authorizing the principal to access an object.
- Programmers tend to simplify authorization, despite its complicated issues.

Levels of Protection

- Not sharing at all (complete isolation)
- Sharing copies of objects, original objects, or subsystems
- Enabling mutually suspicious subsystems to cooperate
- Memoryless subsystems (keeping no secret)
- Certified subsystems (validated trustworthiness)

Graham and Denning

Mutually suspicious subsystems

- Components
  - Objects being accessed (e.g., memory pages)
    - Unique object identification number (given at creation)
  - Subjects: process (program in execution), domain (environment)
    - Model regards subjects as objects
  - Protection system (governs rules for authorization)

Protection State

- Protection State: All the information specifying the types of access subjects have to objects.
- How to represent the protection state?
- How to enforce the protection state?
- How to alter the protection state?

Representation

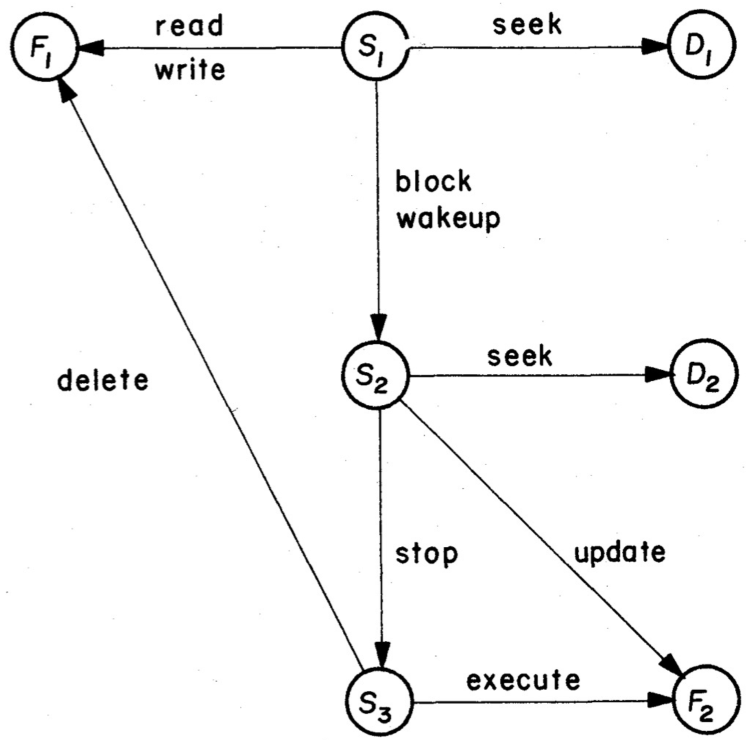
Access attribute A[S,X] = α
S has α access to X.

OBJECTS

		subjects			files		devices	
		S_1	S_2	S_3	F_1	F_2	D_1	D_2
SUBJECTS	S_1		block wakeup		read write		seek	
	S_2			stop		update		seek
	S_3				delete	execute		

An example of an access matrix *A*

Representation

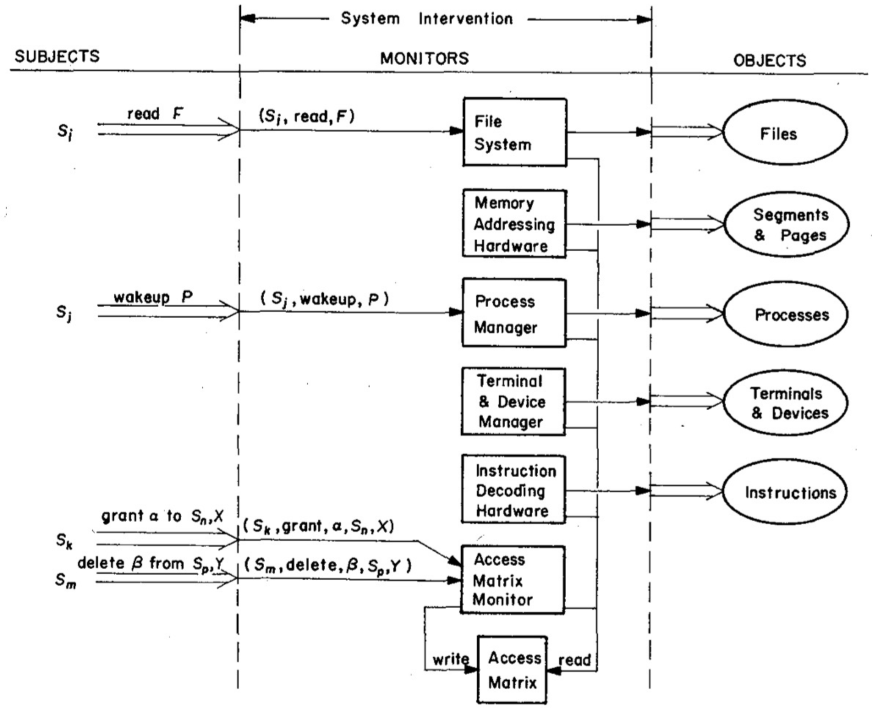


Mechanism

1. S initiates access to X in manner α.

2. System supplies (S, α, X) to monitor of X.

3. Monitor of X interrogates access to determine if α is in A[S,X]. If so, access is permitted.



Challenge: How to protect ID of each subject?

Rules of the Model

R1—3 especially used by access matrix monitor

TABLE I—Protection System Commands

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\left\{\begin{matrix}\alpha^* \\ \\ \alpha\end{matrix}\right\}$ **to** S, X	'α^*' in $A[S_o, X]$	store $\left\{\begin{matrix}\alpha^* \\ \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R2	**grant** $\left\{\begin{matrix}\alpha^* \\ \\ \alpha\end{matrix}\right\}$ **to** S, X	'owner' in $A[S_o, X]$	store $\left\{\begin{matrix}\alpha^* \\ \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	ω:= **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into ω
R5	**create object** X	none	add column for X to A; store 'owner' in $A[S_o, X]$
R6	**destroy object** X	'owner' in $A[S_o, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_o, S]$	delete row for S from A; execute **destroy object** S

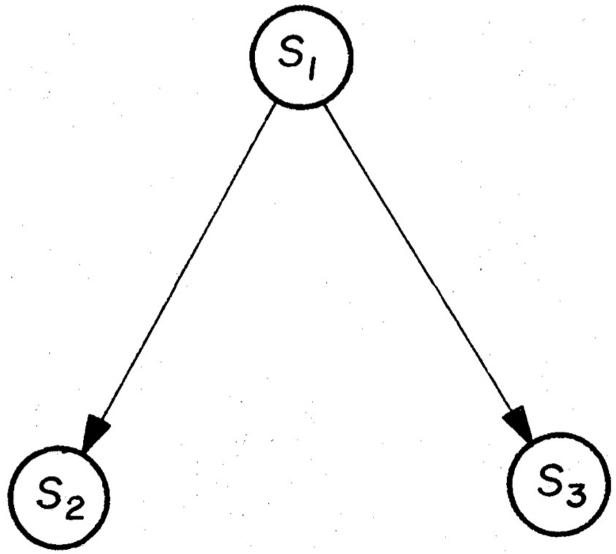
Example

	S_1	S_2	S_3	F_1	F_2	D_1	D_2
S_1	control	owner block wakeup	owner control	read * write *		seek	owner
S_2		control	stop	owner	update	owner	seek *
S_3			control	delete	owner execute		

Creating Objects

- Add new column to matrix for a new object O.
- Owner will grant access to any S on O using R2.
- To delete O, remove its column (only by owner).
- Add new column and row for a new subject object S. S will have control access to itself.
- To delete S, owner will remove its row and column from A.

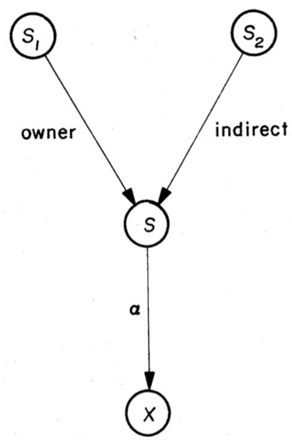
Ownership Hierarchy



A universal subject has no owner.

Sharing

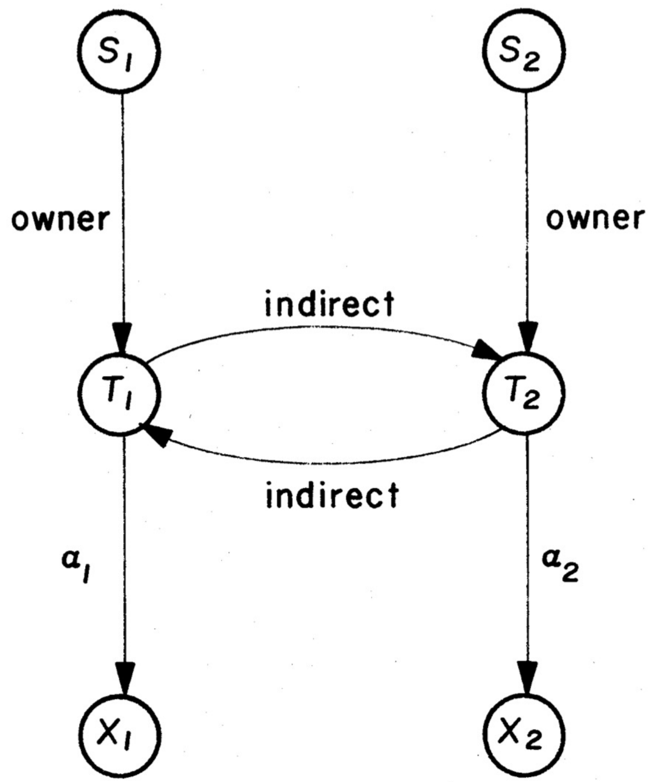
- Suppose S_1 owns S and wishes to share it with S_2. Both S_1 and S_2 distrust each others.
- Indirect attribute: If S_2 is given indirect access to S, S_2 can access and read access attributes of S, but won't be able to acquire accesses of S.
- S_2 initiates access to X through S (S_2,a,S-X).
- Monitor of X checks if indirect is in A[S_2,S] and that a is in A[S,X]



Sharing

T1 can only access objects accessible by T2 but nothing else from S2.

T1 can only use but not acquire access attributes of T2.



Lattice-Based Access Control

Information Flow

- Flow of information from a security class to another.
- Security classes (privileges) are assigned to *objects*.

Information flow policy is a triple $(SC, \to, \oplus)$, where SC is a set of security classes, $\to \subseteq SC \times SC$ is a binary can-flow relation on SC, and $\oplus : SC \times SC \to SC$ is a binary class-combining or join operator on SC.

(Trivial) Flow Example 1

Isolated classes: $SC = \{A_1, \ldots, A_n\}$; for $i = 1 \ldots n$ we have $A_i \rightarrow A_i$ and $A_i \oplus A_i = A_i$; and for $i, j = 1 \ldots n$, $i \neq j$ we have $A_i \not\rightarrow A_j$ and $A_i \oplus A_j$ is undefined.

Information only flows to self.

Denning's Axioms (Lattice)

First: The set of security classes is finite.

Second: $\rightarrow$ is a partial order on SC
Reflexive: $A \rightarrow A$,
Transitive: $A \rightarrow B$, $B \rightarrow C$, $A \rightarrow C$, Antisymmetric: $A \rightarrow B$, $B \rightarrow A$, $A = B$.

Third: SC have a lower bound L, $L \rightarrow A$ for $\forall A \in SC$. Used to model public information (lower bound). Example 1 fails this.

Fourth:
(1) $\oplus$ must be totally defined,
(2) $\oplus$ is a least upper bound:

$A, B, C \in SC$, we have $A \rightarrow A \oplus B$ and $B \rightarrow A \oplus B$, and if $A \rightarrow C$ and $B \rightarrow C$ then $A \oplus B \rightarrow C$

(Nontrivial) Flow Example 2

High-low policy: $SC = \{H, L\}$, and $\rightarrow = \{(H, H), (L, L), (L, H)\}$. The join operator is defined as $H \oplus H = H$, $L \oplus H = H$, $H \oplus L = H$, and $L \oplus L = L$.

Satisfies Denning's axioms (lattice)

Flow Example 3

Isolated classes: $SC = \{A_1, \ldots, A_n\}$; for $i = 1 \ldots n$ we have $A_i \to A_i$ and $A_i \oplus A_i = A_i$; and for $i, j = 1 \ldots n$, $i \neq j$ we have $A_i \not\to A_j$ and $A_i \oplus A_j$ is undefined.

Does not satisfy Denning's axioms (lattice)

Bounded isolated classes: $SC = \{A_1, \ldots, A_n, L, H\}$; $L \to L$, $L \to H$, $H \to H$, and for $i = 1, \ldots, n$, we have $L \to A_i$, $A_i \to A_i$, $A_i \to H$; for $i = 1, \ldots, n$, we have $A_i \oplus A_i = A_i$, $A_i \oplus H = H$, and $A_i \oplus L = A_i$; and for $i, j = 1, \ldots, n$, $i \neq j$, we have $A_i \oplus A_j = H$.

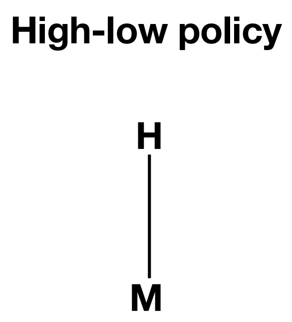
Satisfies Denning's axioms (lattice)

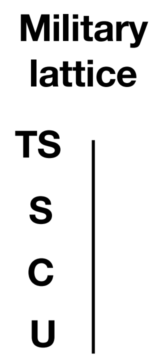
Dominance

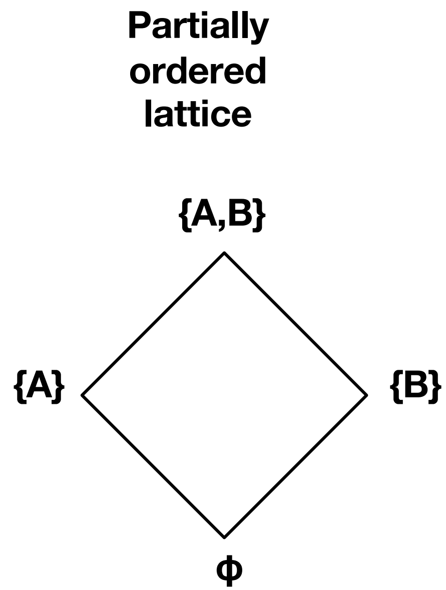
Dominance: $A \geq B$ (A dominates B) if and only if $B \rightarrow A$. Further, $A > B$ (A strictly dominates B) if and only if $A \geq B$ and $A \neq B$. We say that A and B are comparable if $A \geq B$ or $B \geq A$; otherwise A and B are incomparable.

Dominance is the inverse of can-flow.

Hasse Diagram







R. S. Sandhu, "Lattice-based access control models," in Computer, vol. 26, no. 11, pp. 9-19, Nov. 1993, doi: 10.1109/2.241422.

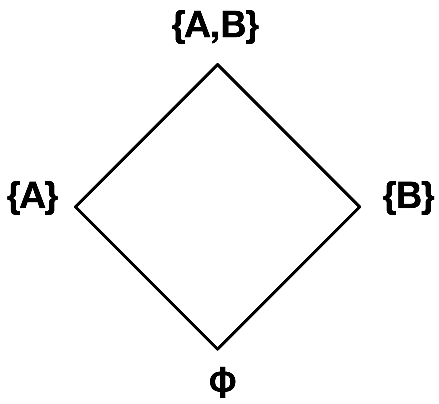
Partially ordered (subset) lattice

E.g., A: salary, B: medical info.
Φ: public info (no salary or medical)

Dominance identical to {A,B}

{A} and {B} are incomparable (partial order)

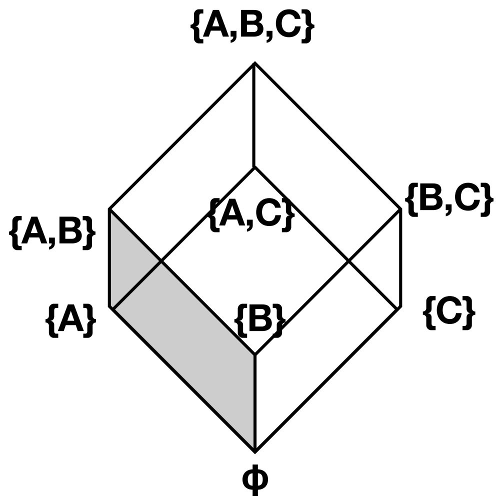
{A} ⊕ {B} = {A, B}



Partially ordered (subset) lattice

E.g., A: salary, B: medical, C: educational
Φ: public info (no salary or medical)

{A} and {B} have two upper bounds {A,B} (least) and {A,B,C}



Example System

Discretionary Access vs **Mandatory Access**

	S_1	S_2	S_3	F_1	F_2	D_1	D_2
S_1	control	owner block wakeup	owner control	read* write*		seek	owner
S_2		control	stop	owner	update	owner	seek*
S_3			control	delete	owner execute		

Simple-security property: Subject s can read object o only if $\lambda(s) \geq \lambda(o)$.
$\star$**-property**: Subject s can write object o only if $\lambda(s) \leq \lambda(o)$.

λ(s) Label on a subject (security clearance)
λ(o) Label on an object (security classification)