Lecture #6

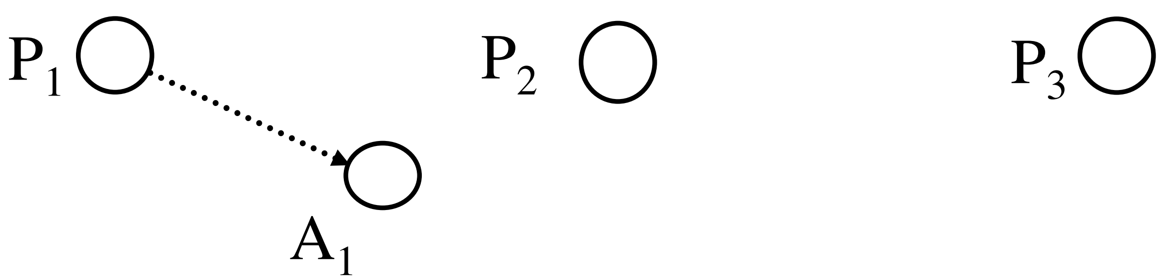
- Multiparent create
- Expressive power
- Typed Access Control Matrix (TAM)
- Overview of Policies
- The nature of policies
  - What they cover
  - Policy languages

Expressiveness

- Graph-based representation to compare models
- Graph
  - Vertex: represents entity, has static type
  - Edge: represents right, has static type
- Graph rewriting rules:
  - Initial state operations create graph in a particular state
  - Node creation operations add nodes, incoming edges
  - Edge adding operations add new edges between existing vertices

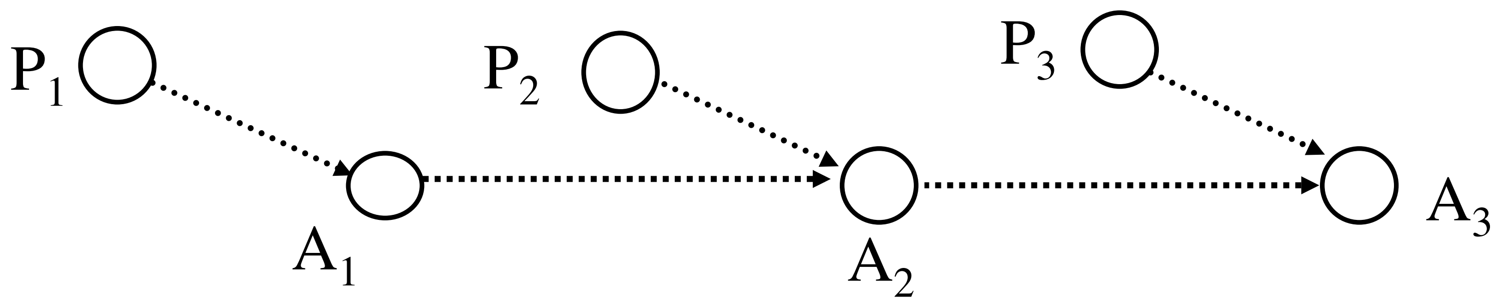
Example: 3-Parent Joint Creation

- Simulate with 2-parent
  - Nodes $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ parents
  - Create node **C** with type *c* with edges of type *e*
  - Add node $\mathbf{A}_1$ of type *a* and edge from $\mathbf{P}_1$ to $\mathbf{A}_1$ of type *e*´



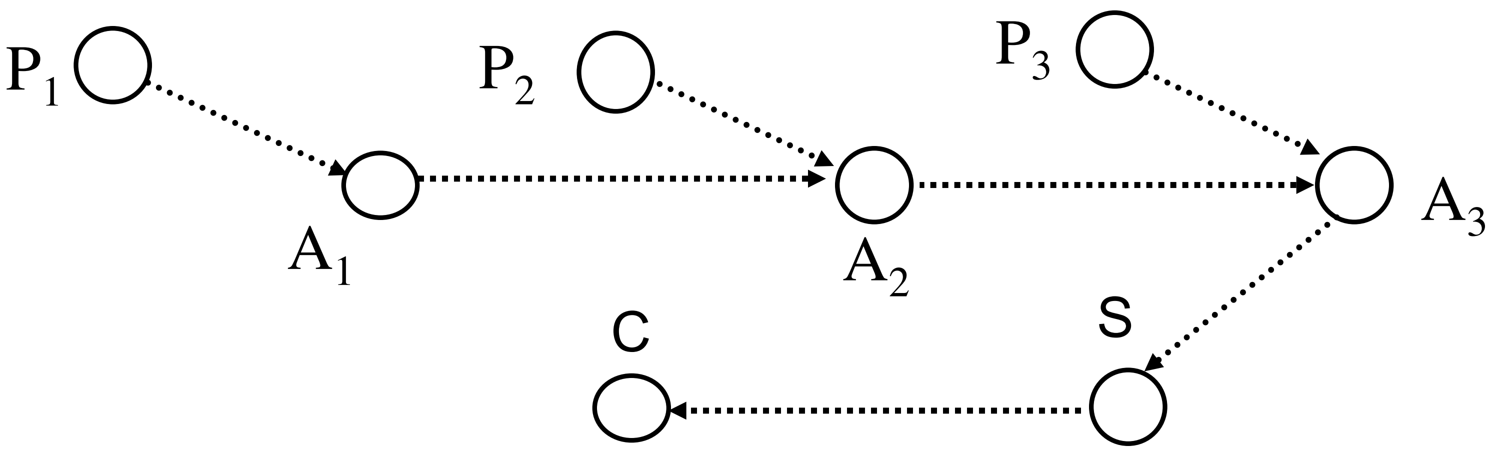
Next Step

- $\mathbf{A}_1$, $\mathbf{P}_2$ create $\mathbf{A}_2$; $\mathbf{A}_2$, $\mathbf{P}_3$ create $\mathbf{A}_3$
- Type of nodes, edges are *a* and *e*´



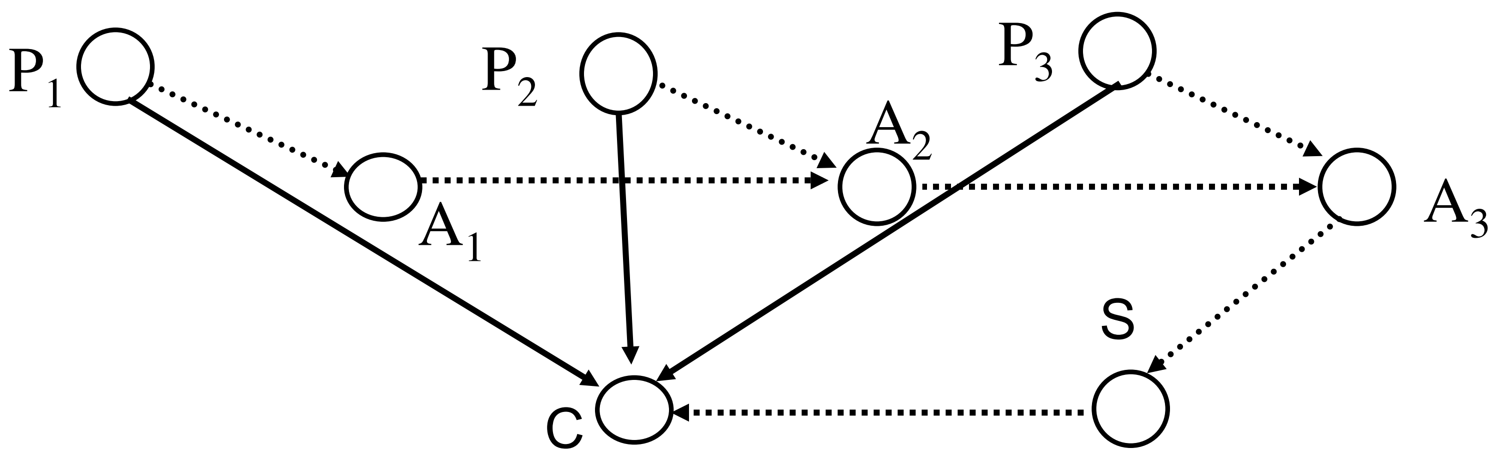
Next Step

- $\mathbf{A}_3$ creates **S**, of type *a*
- **S** creates **C**, of type *c*



Last Step

- Edge adding operations:
  - $\mathbf{P}_1 \rightarrow \mathbf{A}_1 \rightarrow \mathbf{A}_2 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_1$ to **C** edge type *e*
  - $\mathbf{P}_2 \rightarrow \mathbf{A}_2 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_2$ to **C** edge type *e*
  - $\mathbf{P}_3 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_3$ to **C** edge type *e*



Definitions

- *Scheme*: graph representation as above
- *Model*: set of schemes
- Schemes A, B *correspond* if graph for both is identical when all nodes with types not in A and edges with types in A are deleted

Example

- Above 2-parent joint creation simulation in scheme *TWO*
- Equivalent to 3-parent joint creation scheme *THREE* in which $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, **C** are of same type as in *TWO*, and edges from $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ to **C** are of type e, and no types a and e' exist in *TWO*

Simulation

Scheme *A* simulates scheme *B* iff

- every state *B* can reach has a corresponding state in *A* that *A* can reach; and
- every state that *A* can reach either corresponds to a state *B* can reach, or has a successor state that corresponds to a state *B* can reach
  - The last means that *A* can have intermediate states not corresponding to states in *B*, like the intermediate ones in *TWO* in the simulation of *THREE*

Expressive Power

- If there is a scheme in *MA* that no scheme in *MB* can simulate, *MB* less expressive than *MA*
- If every scheme in *MA* can be simulated by a scheme in *MB*, *MB* as expressive as *MA*
- If *MA* as expressive as *MB* and *vice versa*, *MA* and *MB* equivalent

Example

- Scheme *A* in model *M*
  - Nodes $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$
  - 2-parent joint create
  - 1 node type, 1 edge type
  - No edge adding operations
  - Initial state: $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, no edges
- Scheme *B* in model *N*
  - All same as *A* except no 2-parent joint create
  - 1-parent create
- Which is more expressive?

Can *A* Simulate *B*?

- Scheme *A* simulates 1-parent create: have both parents be same node
  - Model *M* as expressive as model *N*

Can *B* Simulate *A*?

- Suppose $\mathbf{X}_1$, $\mathbf{X}_2$ jointly create $\mathbf{Y}$ in *A*
  - Edges from $\mathbf{X}_1$, $\mathbf{X}_2$ to $\mathbf{Y}$, no edge from $\mathbf{X}_3$ to $\mathbf{Y}$
- Can *B* simulate this?
  - Without loss of generality, $\mathbf{X}_1$ creates $\mathbf{Y}$
  - Must have edge adding operation to add edge from $\mathbf{X}_2$ to $\mathbf{Y}$
  - One type of node, one type of edge, so operation can add edge between any 2 nodes

No

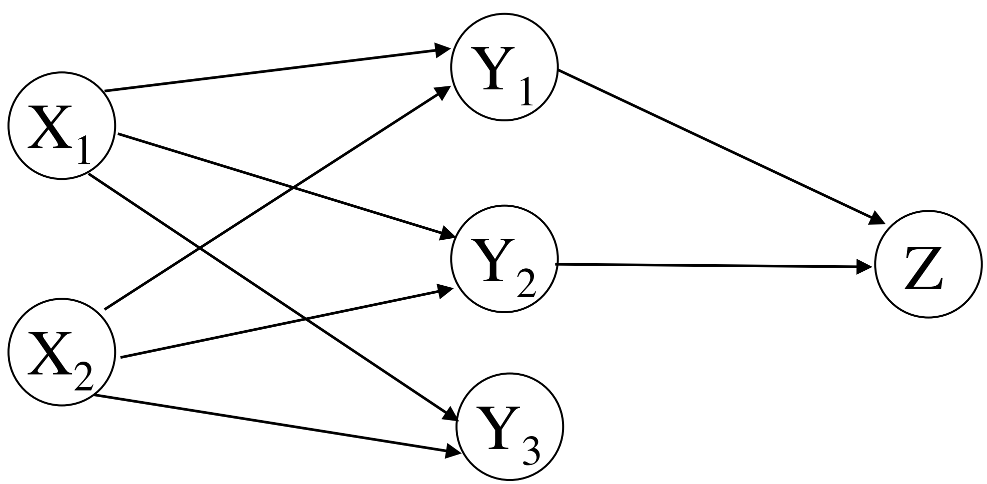
- All nodes in *A* have even number of incoming edges
  - 2-parent create adds 2 incoming edges
- Edge adding operation in *B* that can edge from $\mathbf{X}_2$ to **C** can add one from $\mathbf{X}_3$ to **C**
  - *A* cannot enter this state
    - *A*, cannot have node (**C**) with 3 incoming edges
  - *B* cannot transition to a state in which **Y** has even number of incoming edges
    - No remove rule
- So *B* cannot simulate *A*; *N* less expressive than *M*

Theorem

- Monotonic single-parent models are less expressive than monotonic multiparent models
- Proof by contradiction
  - Scheme *A* is multiparent model
  - Scheme *B* is single parent create
  - Claim: *B* can simulate *A*, without assumption that they start in the same initial state
    - Note: example assumed same initial state

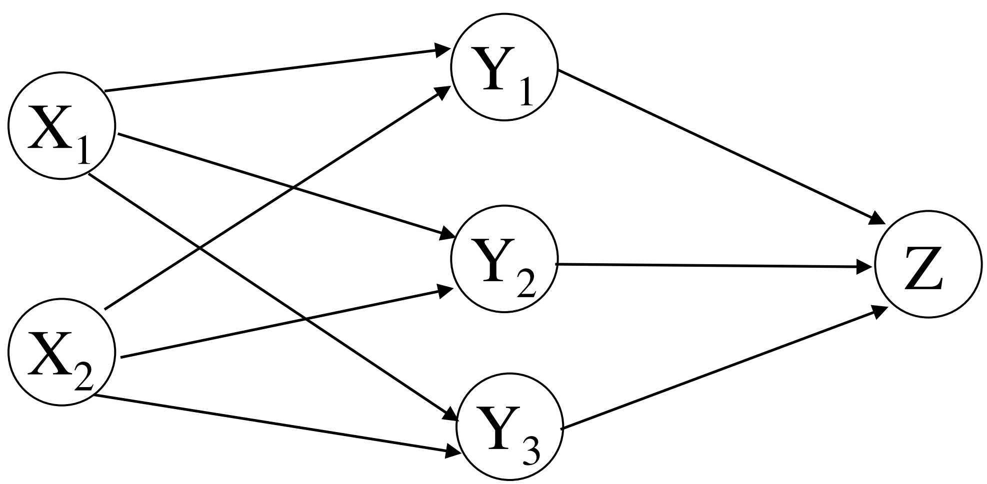
Outline of Proof

- $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *A*
  - They create $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using multiparent create rule
  - $\mathbf{Y}_1$, $\mathbf{Y}_2$ create $\mathbf{Z}$, again using multiparent create rule
  - *Note*: no edge from $\mathbf{Y}_3$ to $\mathbf{Z}$ can be added, as *A* has no edge-adding operation



Outline of Proof

- **W**, $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *B*
  - **W** creates $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using single parent create rule, and adds edges for $\mathbf{X}_1$, $\mathbf{X}_2$ to all using edge adding rule
  - $\mathbf{Y}_1$ creates **Z**, again using single parent create rule; now must add edge from $\mathbf{X}_2$ to **Z** to simulate *A*
  - Use same edge adding rule to add edge from $\mathbf{Y}_3$ to **Z**: cannot duplicate this in scheme *A*!



Meaning

- Scheme *B* cannot simulate scheme *A*, contradicting hypothesis
- ESPM more expressive than SPM
  - ESPM multiparent and monotonic
  - SPM monotonic but single parent

Typed Access Matrix Model

- Like ACM, but with set of types T
  - All subjects, objects have types
  - Set of types for subjects TS
- Protection state is (S, O, τ, A)
  - $\tau: O \rightarrow T$ specifies type of each object
  - If **X** subject, $\tau(\mathbf{X}) \in TS$
  - If **X** object, $\tau(\mathbf{X}) \in T - TS$

Create Rules

- Subject creation
  - **create subject** *s* **of type** *ts*
  - *s* must not exist as subject or object when operation executed
  - $ts \in TS$
- Object creation
  - **create object** *o* **of type** *to*
  - *o* must not exist as subject or object when operation executed
  - $to \in T - TS$

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s **of type** t
- Postconditions:
  - $S' = S \cup \{ s \}, O' = O \cup \{ s \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)], \tau'(s) = t$
  - $(\forall y \in O')[a'[s, y] = \varnothing], (\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o **of type** t
- Postconditions:
  - $S' = S, O' = O \cup \{ o \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)], \tau'(o) = t$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

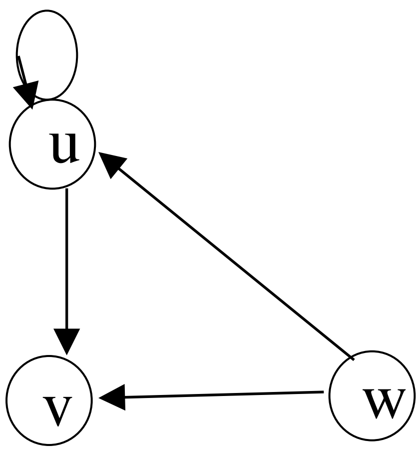
Definitions

- MTAM Model: TAM model without **delete**, **destroy**
  - MTAM is Monotonic TAM
- $\alpha(x_1{:}t_1, ..., x_n{:}t_n)$ create command
  - t_i child type in α if any of **create subject** x_i **of type** t_i or **create object** x_i **of type** t_i occur in α
  - t_i parent type otherwise

Cyclic Creates

command *havoc*$(s : u, p : u, f : v, q : w)$
  create subject p **of type** u;
  create object f **of type** v;
  enter *own* **into** $a[s, p]$;
  enter r **into** $a[q, p]$;
  enter *own* **into** $a[p, f]$;
  enter r **into** $a[p, f]$
end

Creation Graph



- u, v child types
- u, w parent types
- Graph: lines from parent types to child types
- This one has cycles

Acyclic Creates

command *havoc*(s : u, p : u, f : v, q : w)

    create object f **of type** v;

    enter *own* **into** $a[s, p]$;

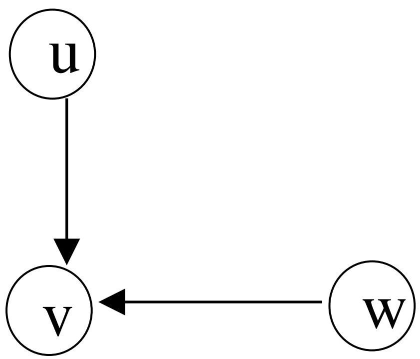
    enter r **into** $a[q, p]$;

    enter *own* **into** $a[p, f]$;

    enter r **into** $a[p, f]$

end

Creation Graph



- *v* child type
- *u*, *w* parent types
- Graph: lines from parent types to child types
- This one has no cycles

Theorems

- Safety decidable for systems with acyclic MTAM schemes
  - In fact, it's *NP-hard*
- Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  - "Ternary" means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM

Policies and All That

- Policy: says what is, and is not, allowed
- Key point is *expression*
  - How do you state it in a precise, understandable way?
  - What do you want it to say?

Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (nonsecure)
    - If the system enters any of these states, it’s a security violation
- Secure system
  - Starts in authorized state
  - Never enters unauthorized state

Confidentiality

- *X* set of entities, *I* information
- *I* satisfies *confidentiality* property with respect to *X* if no $x \in X$ can obtain information from *I*
- *I* can be disclosed to others
- Example:
  - *X* set of students
  - *I* final exam answer key
  - *I* is confidential with respect to *X* if students cannot obtain final exam answer key

Integrity

- X set of entities, I information
- I satisfies *integrity* property with respect to X if all $x \in X$ trust information in I
- Types of integrity:
  - trust I, its conveyance and protection (data integrity)
  - I information about origin of something or an identity (origin integrity, authentication)
  - I resource: means resource functions as it should (assurance)

Availability

- X set of entities, I resource
- I satisfies *availability* property with respect to X if all $x \in X$ can access I
- Types of availability:
  - traditional: x gets access or not
  - quality of service: promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved

Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
  - Security levels in multilevel security models
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model

Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting confidentiality
- Commercial security policy
  - Policy primarily protecting integrity
- Confidentiality policy
  - Policy protecting only confidentiality
- Integrity policy
  - Policy protecting only integrity

Integrity and Transactions

- Begin in consistent state
  - "Consistent" defined by specification
- Perform series of actions (*transaction*)
  - Actions cannot be interrupted
  - If actions complete, system in consistent state
  - If actions do not complete, system reverts to beginning (consistent) state

Trust

Administrator installs patch

1. Trusts patch came from vendor, not tampered with in transit
2. Trusts vendor tested patch thoroughly
3. Trusts vendor's test environment corresponds to local environment
4. Trusts patch is installed correctly

Trust in Formal Verification

- Gives formal mathematical proof that given input i, program P produces output o as specified
- Suppose a security-related program S formally verified to work with operating system O
- What are the assumptions?

Trust in Formal Methods

1. Proof has no errors
   - Bugs in automated theorem provers
2. Preconditions hold in environment in which *S* is to be used
3. *S* transformed into executable *S′* whose actions follow source code
   – Compiler bugs, linker/loader/library problems
4. Hardware executes *S′* as intended
   – Hardware bugs (Pentium `f00f` bug, for example)

Question

- Policy disallows cheating
  - Includes copying homework, with or without permission
- CS class has students do homework on computer
- Anne forgets to read-protect her homework file
- Bill copies it
- Who cheated?
  - Anne, Bill, or both?

Answer Part 1

- Bill cheated
  - Policy forbids copying homework assignment
  - Bill did it
  - System entered unauthorized state (Bill having a copy of Anne's assignment)
- If not explicit in computer security policy, certainly implicit
  - Not credible that a unit of the university allows something that the university as a whole forbids, unless the unit explicitly says so

Answer Part #2

- Anne didn’t protect her homework
  - Not required by security policy
- She didn’t breach security
- If policy said students had to read-protect homework files, then Anne did breach security
  - She didn’t do this

Mechanisms

- Entity or procedure that enforces some part of the security policy
  - Access controls (like bits to prevent someone from reading a homework file)
  - Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

Types of Access Control

- Discretionary Access Control (DAC, IBAC)
  - individual user sets access control mechanism to allow or deny access to an object
- Mandatory Access Control (MAC)
  - system mechanism controls access to object, and individual cannot alter that access
- Originator Controlled Access Control (ORCON)
  - originator (creator) of information controls who can access information

Policy Languages

- Express security policies in a precise way
- High-level languages
  - Policy constraints expressed abstractly
- Low-level languages
  - Policy constraints expressed in terms of program options, input, or specific characteristics of entities on system