Lecture 34: Access Control

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

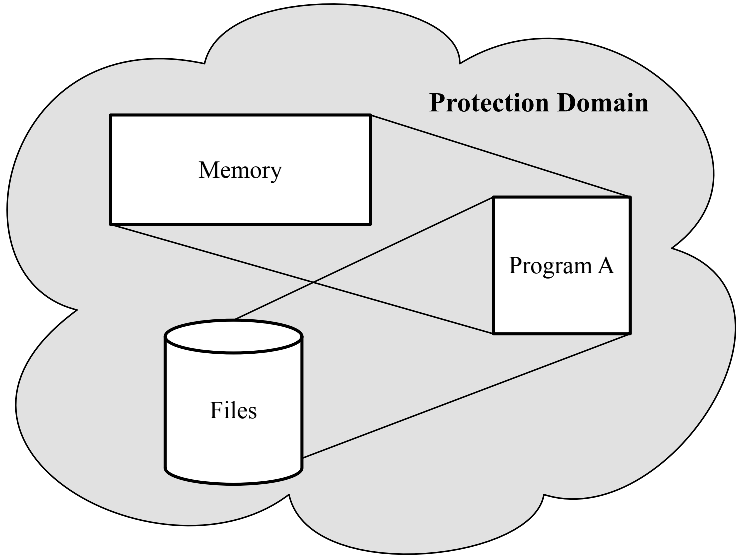
Goals for Today

- Learning Objectives:
  - Discuss Mandatory Access Control
- Announcements, etc:
  - Forensics CP1 due Today!
  - Forensics CP2 due **December 6th**


Reminder: Please put away devices at the start of class

Protection

- A protection system describes the conditions under which a system is secure.
- The state of a system is the collection of all values of memory locations, storage, registers, and other components of the system.
- The subset of this information focused on protection is the protection state. This is also known as the protection domain.



Access Control Matrix

- An access control matrix is one way to represent protection state.
- Columns are objects, rows are subjects, entries represent rights
- To determine if Si has right to access Oi, find appropriate entry
- There is a matrix for each set of rights

	o_1	o_2	o_3
s_1	Y	Y	N
s_2	N	Y	N
s_3	N	Y	Y

Question: Secrecy

- Does the following protection state ensure the secrecy of J's private key file, object O1?

	O_1	O_2	O_3
J	R	RW	RW
S_2	-	R	RW
S_3	-	R	RW

Question: Integrity

- Does the following protection state protect the integrity of J's public key file, object O2?

	O_1	O_2	O_3
J	R	RW	RW
S_2	-	R	RW
S_3	-	R	RW

Trusted Processes

- Does it matter if we do not trust some of J's processes?
- **Trojan Horse**: Attacker-controlled code run by J can violate secrecy.
- **Confused Deputy**: Attacker may trick untrusted code run by J to violate integrity.

	O_1	O_2	O_3
J	R	RW	RW
S_2	-	R	RW
S_3	-	R	RW

Least Privilege

- The principle of least privilege states that a system should only provide those rights needed to perform the processes function and no more.
  - **Implication 1**: You want to reduce the protection domain to the smallest possible set of objects
  - **Implication 2**: You want to assign the minimal set of rights to each subject
  - **Caveat**: You need to provide enough rights and a large enough protection domain to get the job done.

Least Privilege

- **Goal**: Limit permissions to those required and no more. Consider three processes for user J.

	o_1	o_2	o_3
J_1	R	RW	RW
J_2	-	R	RW
J_3	-	R	RW

Least Privilege

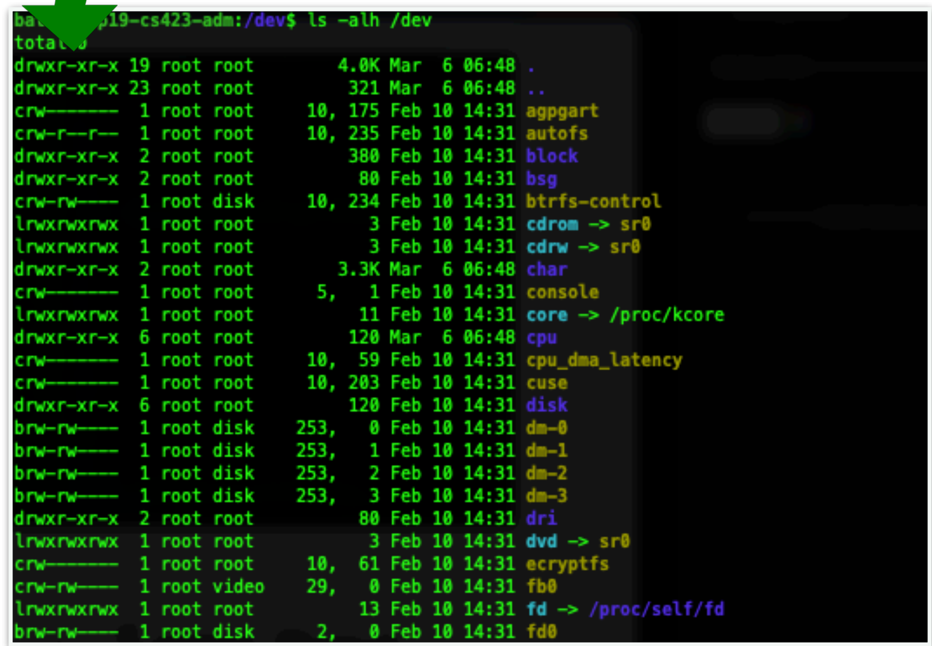
- **Goal**: Limit permissions to those required and no more. Consider three processes for user J.
  - Restrict privilege of the processes J1 and J2 to prevent leaks of O3.

	O_1	O_2	O_3
J_1	R	RW	-
J_2	-	R	-
J_3	-	R	RW

Process State Transition

- When we removed the access permissions from O3, we changed the access control matrix. If this happened during system execution (e.g., chmod), this would represent a state transition.
- We moved the system from protection state X_i to state $X_{(i+1)}$.
- A sequence of state transitions that updates the access control matrix is also known as a transformation procedure.

Discretionary Access Control (DAC)



Access Mask defines permissions for User, Group, and Other

```
chmod u=rwx,g=rx,o=r myfile
chmod 754 myfile
```

<- Same thing

4 stands for "read",
2 stands for "write",
1 stands for "execute", and
0 stands for "no permission."

DAC vs MAC

1. Discretionary access control: "owners" of an object define its policy.
   - Users have discretion over who has access to what objects and when (i.e., users are trusted)
   - Canonical example: the UNIX filesystem (RWX assigned by file owners)
2. Mandatory access control: the environment enforces a static policy
   - Access control policy is defined by the administrators and the system environment, user has no control over rights
   - Canonical example: Process labeling (system assigns labels for processs and objects)

Protection System

- Consider how to secure a system. A protection system is made up of not only the protection state (e.g., access control matrix), but also the **administrative operations** to modify the protection state and a **reference monitor**.
- A reference monitor can enforce the protection state if its *implementation* meets the following three guarantees:
    1. Tamperproof
    2. Complete Mediation
    3. Simple Enough to Verify

Designing a Security Policy

- Given a perfect **protection system**…
- and what we now know about the principle of **least privilege**…
- What should our security policy (i.e. **protection state**) look like?

US DoD Policy

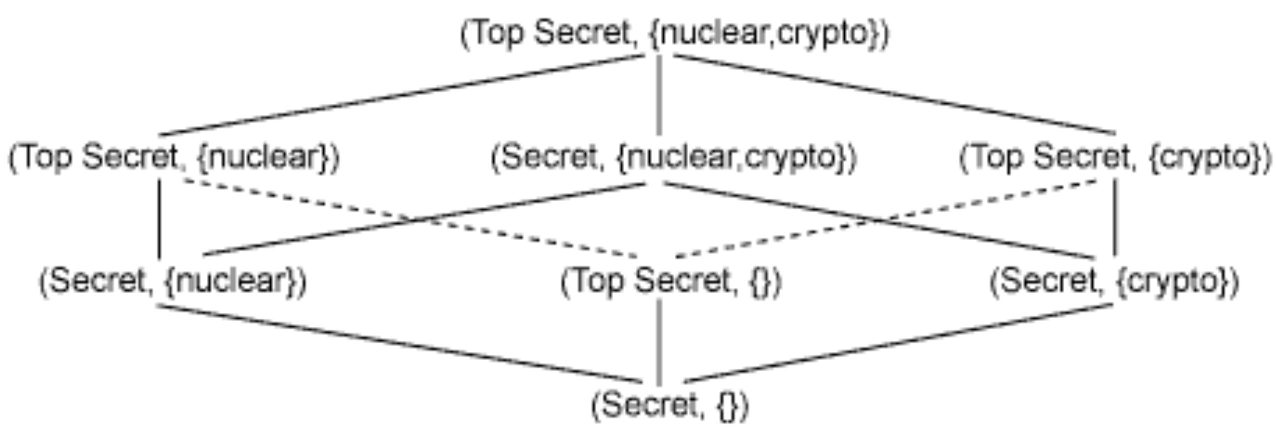
- The **Bell-Lapadula** Information Flow Model
- Used by the US military (and many others), the lattice model defines a Multi-Level Security (MLS) policy:
  - *UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET*
- Categories are represented as an unbounded set:
  - *NUC(lear), INTEL(ligence), CRYPTO(graphy)*
- These levels are used for physical government of documents as well.

US DoD Policy

- All subjects are assigned clearance levels and compartments
  - Alice: (SECRET, {CRYPTO, NUC})
  - Bob: (CONFIDENTIAL, {INTEL})
  - Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL})
- All objects are assigned an access class • DocA: (CONFIDENTIAL, {INTEL})
  - DocB: (SECRET, {CRYPTO})
  - DocC: (UNCLASSIFIED, {NUC})

Bell-LaPadula Model

- A multi-level security model that provides strong confidentiality guarantees.
- Formalizes Classified Information
- State machine (Lattice) specifies permissible actions
- Lattice is comprised of both *levels* and *categories*



Evaluating Policy

- Access is allowed if:
  - subject clearance level ≥ object sensitivity level <u>**and**</u>
  - subject categories ⊆ object categories (read-down property)

Bob
(CONF, {INTEL})

Charlie
(TOP SECRET, {CRYPTO, NUC, INTEL})

Alice
(SEC, {CRYPTO, NUC})

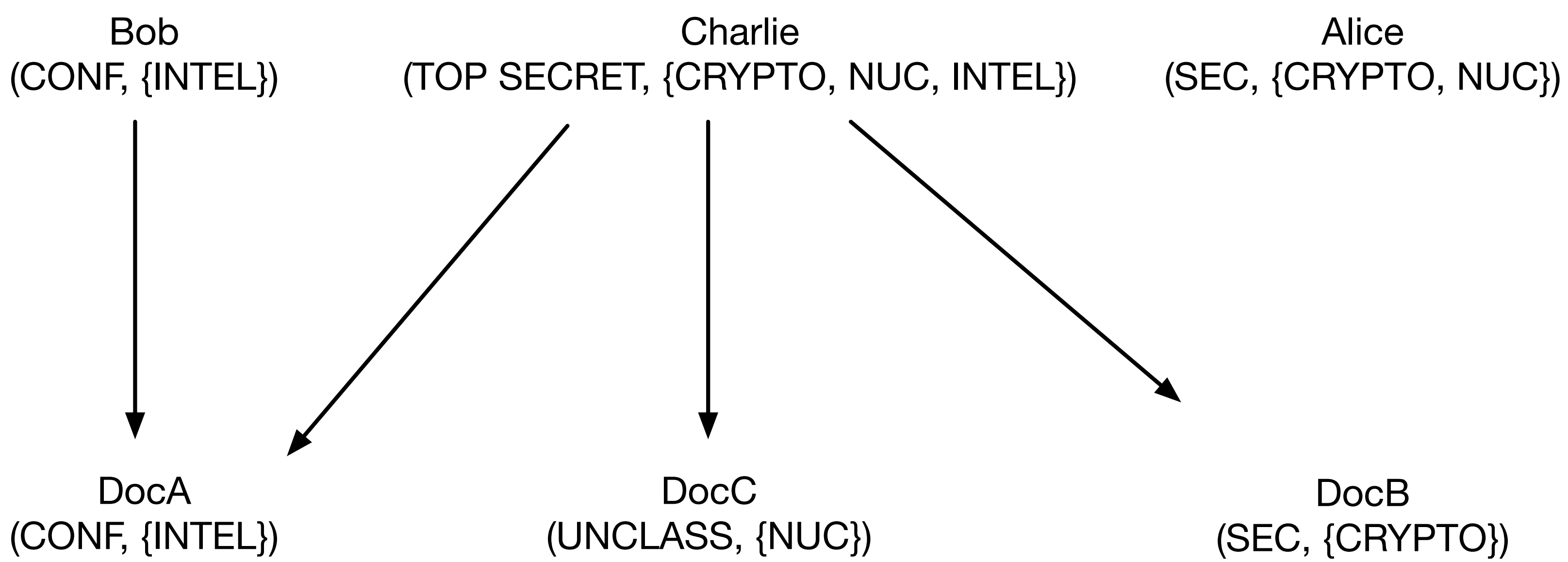
DocA
(CONF, {INTEL})

DocC
(UNCLASS, {NUC})

DocB
(SEC, {CRYPTO})

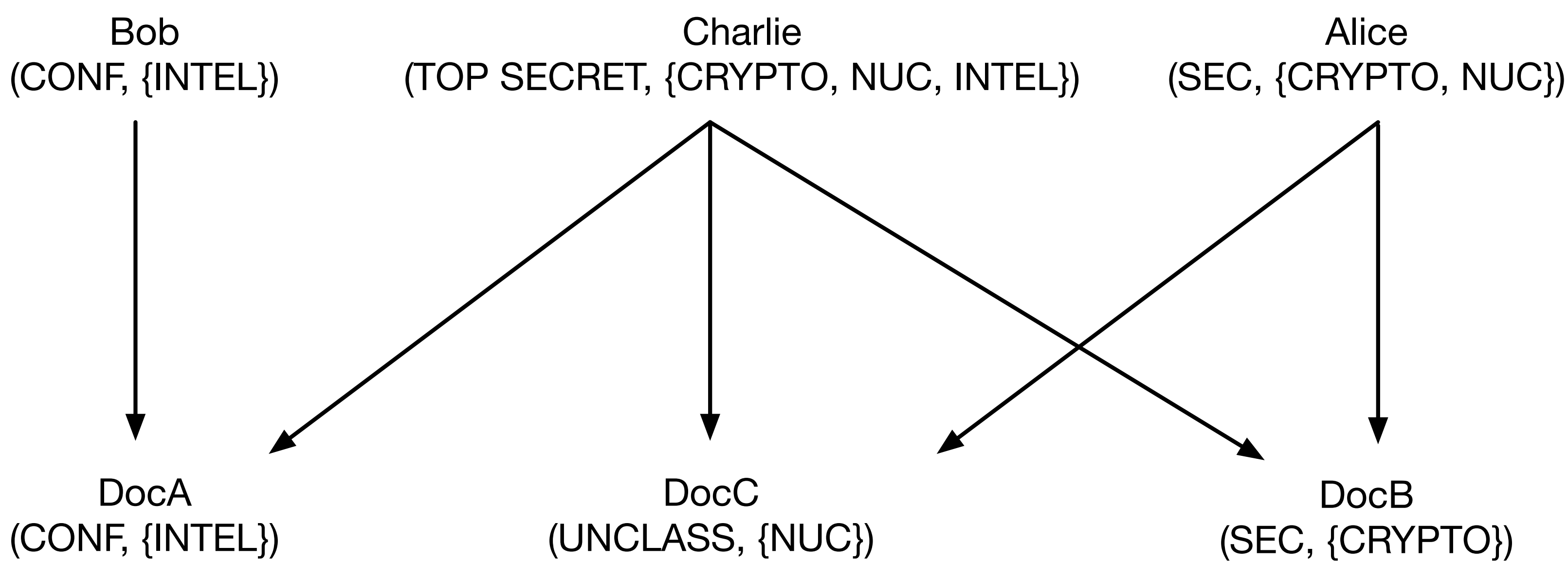
Evaluating Policy

- Access is allowed if:
    - subject clearance level ≥ object sensitivity level <u>**and**</u>
    - subject categories ⊆ object categories (read-down property)



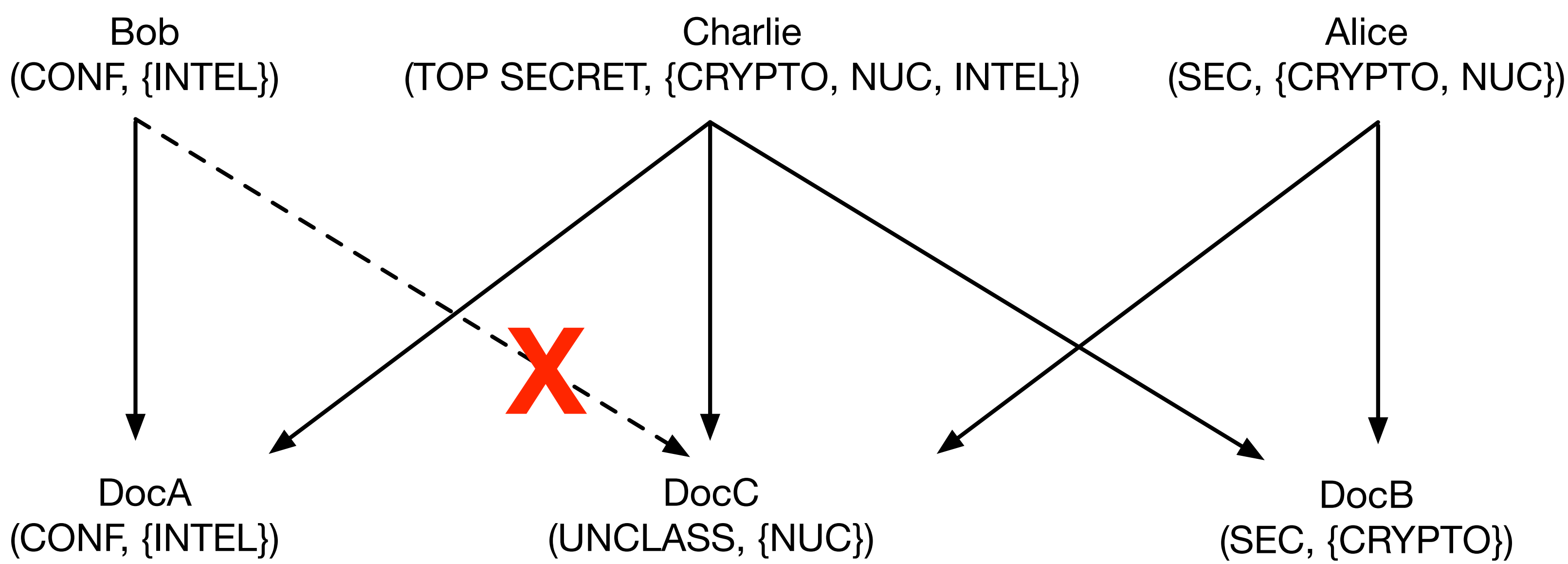
Evaluating Policy

- Access is allowed if:
  - subject clearance level ≥ object sensitivity level ***and***
  - subject categories ⊆ object categories (read-down property)



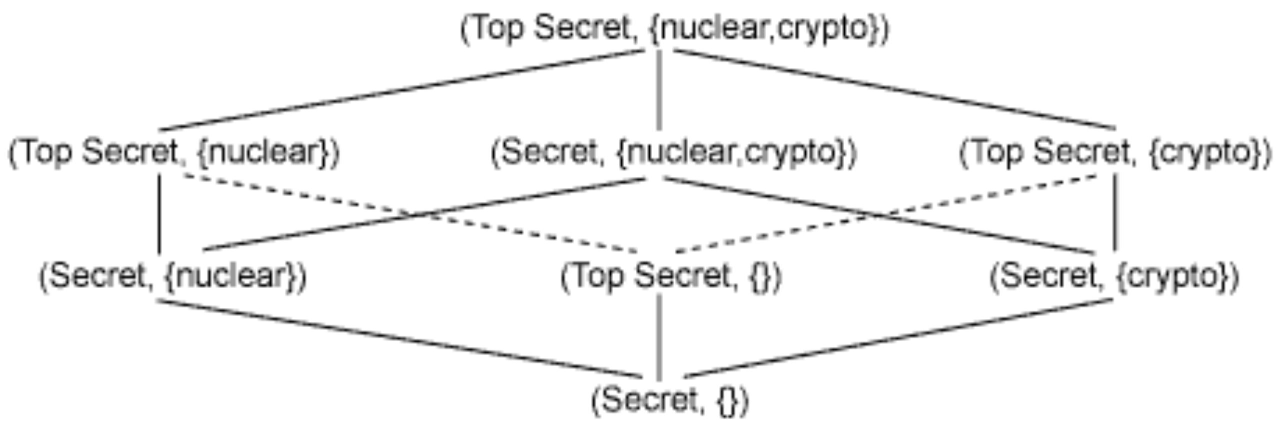
Evaluating Policy

- Access is allowed if:
  - subject clearance level ≥ object sensitivity level ***and***
  - object categories contained in subject categories (read-down property)

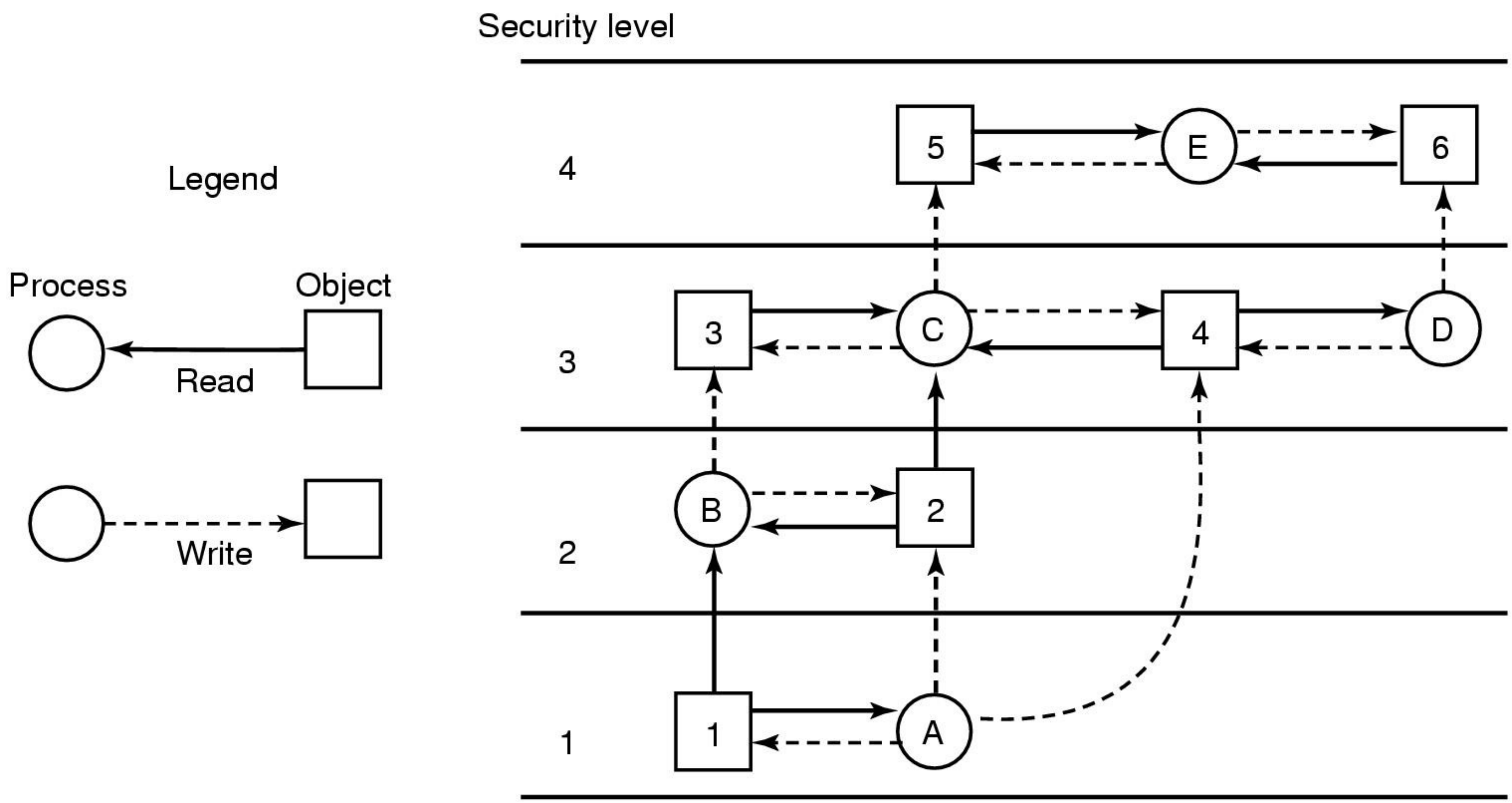


Bell-LaPadula Model

- **The Simple Security Property**: A subject running at security level k can read only objects at its level or lower. (*no read up*)
- **The * Property**: A subject at security level k can write only objects at its level or higher (*no write down)*



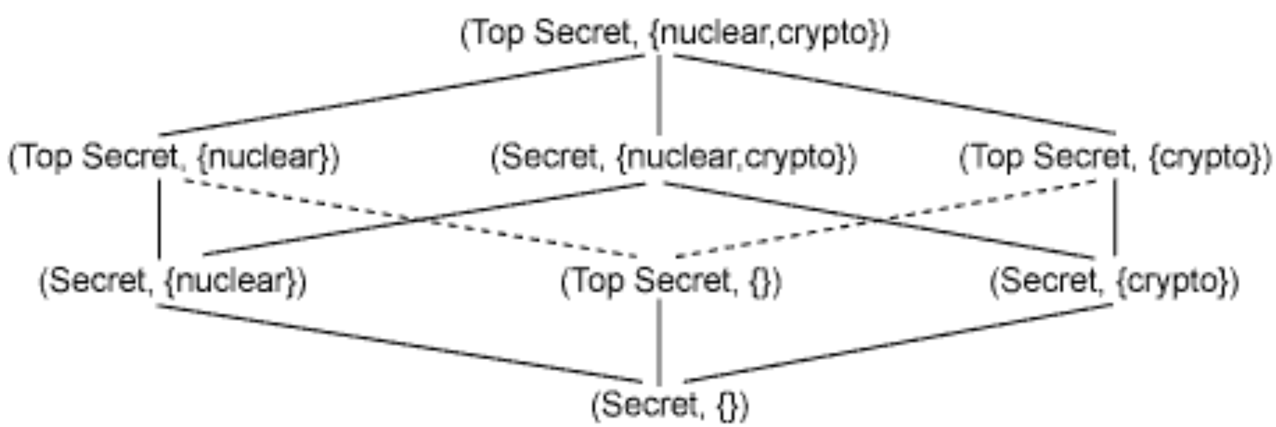
Evaluating Policy



Using Bell-Lapadula, we can reason about permissible information flows in a system.

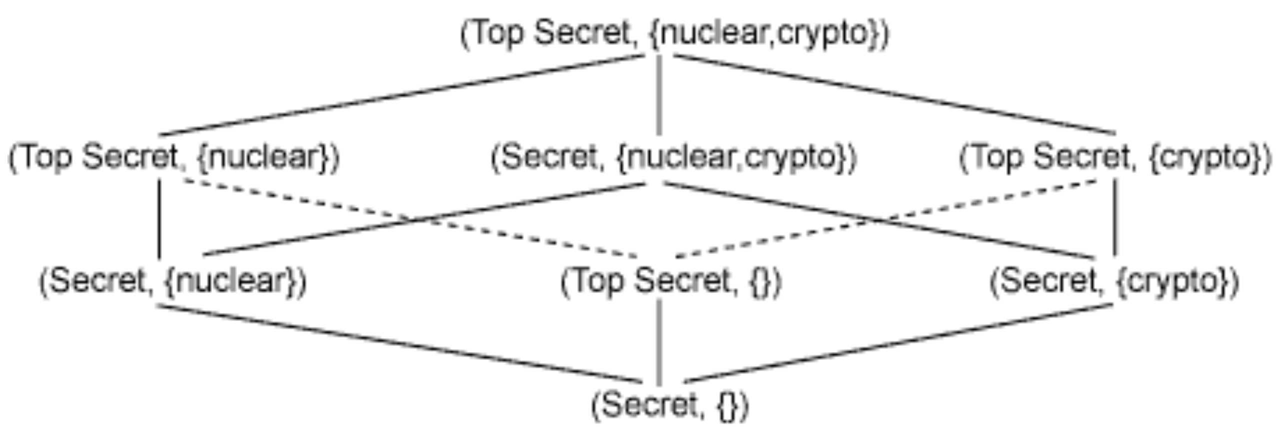
Biba Integrity Model

- Bell-LaPadula provides confidentiality. What about integrity?
- Biba model provides Integrity guarantees in a manner analogous to Bell-Lapadula's secrecy levels.
- Integrity prevents inappropriate modification of data.



Biba Integrity Model

- **The Simple Integrity Property:** A subject running at integrity level k must not read an object at a lower integrity level (*no read down*)
- **The * Integrity Property**: A subject at security level k can only write objects at its level or lower. (*no write up)*



Integrity + Secrecy?

- What happens if we want both?
- What if we turned Biba and Bell-Lapadula on simultaneously?

LOMAC

- Low Water-Mark Mandatory Access Control (LOMAC)
- Supports transitions between different integrity levels for more flexibility in protection
- Let's consider just 2 security levels:
  - Level 1 for low integrity objects (e.g., downloaded files)
  - Level 2 for high integrity objects (e.g., system binaries)
- User processes start at Level 2

LOMAC Example

1. O1 is level 2 object
2. J starts at level 2 (can access)
3. J downloads content containing a Trojan Horse; i.e., J reads from Level 1 data.
4. J is demoted to level 1
5. Trojan Horse uses buffer overflow attack to take control of J.
6. J can no longer access O1 because of demoted integrity level.

An alternate metaphor for integrity

Consider a bank reconciling its daily transactions

- Today's deposits D, withdrawals W, yesterday's balance YB, today's balance TB
- Reconciliation will only be valid when D + YB – W satisfies TB; if valid, day's transactions become trusted.
- Integrity thus defined as a set of constraints and data is in a consistent or valid state when these constraints are satisfied
- A well-formed transaction moves the system from one consistent state to another
- But: who examines and certifies that transactions happen correctly?

Clark-Wilson Integrity

- Integrity Verification Procedures (IVPs) verify the initial integrity of high integrity Constrained Data Items (CDIs).
- CDIs are only modified by Transformation Procedures (TPs).
- Low integrity Unconstrained Data Items (UDIs) can become high integrity through *certification* by a TP in order to become a CDI.
- If all of the above are correct, then the integrity of the computation is preserved *even though the system handles low-integrity data!*

CW Certification Rules 1&2

- CR1: When any IVP is run, it must ensure all CDIs are in a valid state.
- CR2: For some associated set of CDIs, a TP must transform those CDIs in a valid state into a (possibly different) valid state.
  - Defines the relation that associates a set of CDIs with a particular TP as certified
  - In bank example, TP is the reconcile function, CDIs are the accounts

CW Enforcement Rules 1&2

- ER1: The system must maintain the certified relations and must ensure that only TPs certified to run on a CDI manipulate that CDI.
- ER2: The system must associate a user with each TP and set of CDIs. The TP may access those CDIs on behalf of the associated user. The TP cannot access that CDI on behalf of a user not associated with that TP and CDI.
  - System must maintain and enforce the certified relation
  - System must also restrict access based on user ID (allowed relation)

CW User Rules

- CR3: The allowed relations must meet the requirements imposed by the principle of separation of duty.
- ER3: The system must authenticate each user attempting to execute a TP.
  - The type of authentication is not defined here
  - Authentication isn't required before using the system, just before using a TP

CW Untrusted Input Rules

- <u>CR5</u>: Any TP that takes as input a UDI may perform only valid transformations, or no transformations, for all possible values of the UDI. The transformation either rejects the UDI or transforms it into a CDI.
  - In a bank: numbers entered at a keyboard are UDIs so they cannot be input to TPs
  - TPs must validate numbers (certify them to become CDIs) before using them
  - If validation process fails, TP rejects UDI

CW Other Rules

- Logging (CR4: TP must include enough information to reconstruct transaction)
- Separation of duty (ER4: only certifier of TP can change entities associated with the TP, no certifier can execute TP)

Issues with CW

- Certification: All IVPs and TPs have to be validated as correct
- But how do you define that something is correct?
- Model is nice for its goals but heavyweight to implement
- Ongoing research to develop lightweight alternatives