DBS_MidSem_2024

Exported on: 2/23/2026, 2:20:36 AM

User

Answer Q1)

Gemini

In the Bell-LaPadula (BLP) model, the **object hierarchy function** typically organizes objects into a tree structure (like a standard filesystem). This ensures that an object's security clearance is inherited or constrained by its parent.

When the hierarchy deviates from a tree and becomes a general graph—specifically one with **multiple inheritance (DAG)** or **cycles**—the fundamental security properties of "No Read Up" and "No Write Down" face significant logical and enforcement challenges.

---

1. The Core Problem: Security Label Ambiguity

In a tree, every object has exactly one parent. In a graph, an object can have multiple parents or even be its own ancestor (in a cycle). This creates a conflict in determining the effective security level of an object.

Case A: Multiple Parents (Directed Acyclic Graph)

Imagine a system where Object C is a "child" of both Object A (Top Secret) and Object B (Secret).

- **The Conflict:** If a user has "Secret" clearance, can they access Object C?
  - Through Parent B, they should have access.
  - Through Parent A, they should be denied access because $Secret < TopSecret$.
- **The Result:** This leads to **non-deterministic access control**. To fix it, the system must choose a "Maximum" or "Minimum" policy, which often results in over-classification (blocking legitimate access) or under-classification (risking data leakage).

---

2. The Problem of Cycles (Graph with Cycles)

Cycles create a logical paradox in the BLP model's "star property" ($\star$-property).

Example of a Security Cycle:

1. **Object A** (Top Secret) is made a child of **Object B** (Secret).
2. **Object B** (Secret) is then made a child of **Object A** (Top Secret).

- **The Problem:** By the principle of hierarchy, the clearance required for the child should generally be at least as high as the parent to prevent information flow from a high-level container into a lower-level container.
- **The Leak:** If a cycle exists, information can flow from Object A to Object B and back to Object A. In the BLP model, this effectively **collapses the security levels**. If $A \leq B$ and $B \leq A$, then A must equal B.
- **The Consequence:** The distinction between "Secret" and "Top Secret" is lost within that cycle. A user with only Secret clearance could potentially gain access to Top Secret data because the hierarchy no longer provides a clear "up" or "down" direction for the flow of information.

---

3. Summary of Resulting Problems

Problem	Description	Security Impact
Label Inconsistency	Multiple paths to an object provide conflicting security labels.	High: Leads to unauthorized "Read Up."
Transitive Leakage	Information flows through a cycle back to a lower-level node.	Critical: Violates the ⋆-property (No Write Down).
Inheritance Deadlock	The system cannot compute the "Least Upper Bound" for an object.	Operational: System may default to "Deny All" or crash.

Export to Sheets

Conclusion

The Bell-LaPadula model relies on a **strict partial ordering** of security levels. A tree structure perfectly mirrors this order. When the hierarchy becomes a graph with cycles, the "partial ordering" breaks, making it mathematically impossible to enforce the "No Read Up, No Write Down" rules without manually collapsing all nodes in a cycle into a single security level.

Would you like me to also tackle Q2 regarding the revocation problem in ACLs vs. Capabilities?

User

Answer Q2)

Gemini

The **revocation problem** is a classic challenge in access control, centering on how quickly and reliably a system can cancel a user's permissions once they are no longer authorized.

The difficulty of revocation varies significantly depending on whether the system uses **Access Control Lists (ACLs)** or **Capabilities**.

---

1. Revocation in ACLs vs. Capabilities

Access Control Lists (Object-Centric)

In an ACL-based system, each object (like a file) maintains a list of users and their permitted actions.

- **The Process:** To revoke access, the administrator simply goes to the specific object and removes the user’s name from its list.
- **The Advantage:** Revocation is **immediate and straightforward**. Since the "gatekeeper" is at the object itself, the next time the user tries to access it, the system checks the list, sees the name is gone, and denies entry.

Capabilities (User-Centric)

In a capability-based system, each user holds a "ticket" or "key" (the capability) for every object they can access.

- **The Problem:** The system does not keep a central list of who has which key. If you want to revoke a user's access to "File A," you have to find and invalidate the specific "ticket" held by that user.
- **The Challenge:** Capabilities can be passed from one user to another (propagation). Finding every copy of a specific capability across a distributed system is computationally expensive and often results in **delayed revocation**.

---

2. Efficient Implementation of Revocation

To solve the "search and destroy" problem in capabilities, or to manage large-scale ACLs, two primary efficient methods are used:

A. Indirection (The "Alias" Method)

Instead of giving a user a direct capability to an object, the system gives them a capability to an **intermediate pointer** (an entry in a "Mapping Table").

- **How it works:** 1. User $\rightarrow$ Pointer $\rightarrow$ Object. 2. To revoke access, the administrator simply deletes or "nulls" the **Pointer**. 3. Even if the user still holds their capability, it now points to nothing.
- **Efficiency:** This allows for **instant revocation** without needing to track down every distributed copy of the user's capability.

B. Back-pointers (For ACLs)

In systems with millions of objects, finding every file a specific user has access to is slow.

- **How it works:** The system maintains a set of **back-pointers** or a "User-to-Object" index.
- **Efficiency:** When a user is fired, the admin doesn't have to scan the entire filesystem; they consult the index for that specific user and batch-delete their entries across all relevant ACLs.

Summary Comparison

Feature	Access Control List (ACL)	Capabilities
Revocation Ease	High (Easy to remove user from list)	Low (Hard to find all "keys")
Efficiency Solution	Global index / Back-pointers	Indirection / Revocable Pointers
Best For...	Centralized file systems	Distributed systems / Microservices

Export to Sheets

Since you're working through these security models, would you like me to help you draft a practice response for a "Capability vs. ACL" comparison table for your lab report?

User

yes

Gemini

Since you are working on your M.Tech in CSIS, comparing these two models is a common exam or lab report requirement. Below is a structured comparison table and a drafting guide you can use for your submission.

Comparison Table: ACLs vs. Capability Lists

Feature	Access Control List (ACL)	Capability List (C-List)
Orientation	**Object-centric**: Attached to the resource (e.g., File A).	**Subject-centric**: Held by the user (e.g., User 1).
Storage	Stored with the object's metadata.	Stored within the user's process space or token.
Revocation	**Easy**: Remove a user from the object's list.	**Difficult**: Must find and invalidate all held tokens.
Efficiency	Fast for checking "Who can access this file?"	Fast for checking "What files can this user access?"
Least Privilege	Harder to implement granular, temporary access.	Excellent for "fine-grained" temporary delegation.

Export to Sheets

Drafting Your Response: Revocation Implementation

When writing your lab report or exam answer, you can structure your explanation of "efficient implementation" using these three pillars:

1. **Indirection (The Mapping Table):** Explain that instead of a direct link between the user and the object, the system uses a **Revocation Table**. The user's capability points to an entry in this table, which then points to the object. To revoke, you simply delete the table entry.
    - **Analogy:** It's like a hotel key card. The card doesn't "know" the room; the door's computer knows which "code" is currently valid. Change the code at the door, and the card becomes useless.
2. **Back-pointers (User-to-Object Indexing):** For ACLs, searching every file on a server to remove one user is $O(n)$ where n is the number of objects. By maintaining a **global index** of "User → Object" pointers, revocation becomes $O(1)$ or $O(\log n)$.
3. **Time-limited Tokens (Leasing):** Mention **Expiration Dates**. By issuing capabilities that expire every 30 minutes, revocation happens automatically. To maintain access, the user must "renew" the lease, at which point the system checks if they are still authorized.

Pro-Tip for your Lab Report

If you are using **LaTeX** (specifically `IEEEtran` as you have previously), you can format the comparison as a `table` environment and use `itemize` for the implementation strategies to keep it clean and professional.

Would you like me to generate the LaTeX code for this table so you can drop it directly into your report?