

Unit-3

Database Security:

Recent Advances in Access Control:

- Access Control is a method of limiting access to a system or resources.
- Access control refers to the process of determining who has access to what resources within a network and under what conditions.
- It is a fundamental concept in security that reduces risk to the business or organization.
- Access control systems perform identification, authentication, and authorization of users and entities by evaluating required login credentials that may include passwords, pins, bio-metric scans, or other authentication factors.

How Access Control works:

- Access control involves determining a user based on their credentials and then providing the appropriate level of access once confirmed.
- Credentials are used to identify and authenticate a user include passwords, pins, security tokens, and even biometric scans.
- Example Authentication Factors: Password/Pin, Bio-metric (fingerprint, retina-scan), Key-Card, etc
- Multifactor authentication (MFA) increases security by requiring users to be validated using more than one method.
- Once a user's identity has been verified, access control policies grant specified permissions, allowing the user to proceed further.

Three Abstraction Layers of Access Control:

- **Policy:** High-level rules for granting or denying access.
- **Model:** Formal representation of policies.
- **Mechanism:** Enforcement of the model on a system.



1. Policy – *What is allowed or denied?*


- **Definition:** A **policy** is a set of high-level **rules** that define what kind of access is permitted or prohibited.
- **Purpose:** Specifies *who* can do *what* to *which resources* under *what conditions*.
- **Example:**
 - "Only doctors can access patient medical records."
 - "Interns cannot modify billing information."



Think of it as the "*what and why*" of access control.


2. Model – *How is the policy formally defined?*

- **Definition:** A **model** is a **formal framework** or mathematical abstraction that represents the rules laid out in the policy.
- **Purpose:** Translates real-world policies into well-defined, structured logic that can be reasoned about and analyzed.
- **Common Models:**
 - **DAC (Discretionary Access Control):** Access based on user identity.
 - **MAC (Mandatory Access Control):** Access based on classification levels.
 - **RBAC (Role-Based Access Control):** Access based on user roles.

 This is the "*how to structure the rules*" layer—logic behind the rules.

3. Mechanism – *How is the model implemented?*

- **Definition:** A **mechanism** is the **software or hardware component** that enforces the access control model.
- **Purpose:** Ensures the model's rules are actually applied in practice during system execution.
- **Examples:**
 - Operating system's permission checks.
 - Database systems evaluating ACLs.
 - Middleware authenticating user credentials and checking roles.

 This is the "*how it's enforced*" layer—implemented in actual code/systems.

Access Control Models

1. DAC
2. MAC
3. RBAC
4. CBAC

Discretionary Access Control (DAC)

What is DAC?

Discretionary Access Control (DAC) is a type of access control model where the owner of a resource (like a file or database) has the discretion (freedom) to decide who else can access it, and with what permissions.

Key Characteristics of DAC

1. **User-Centric:** Access is granted based on user identity.
 2. **Resource Ownership:** The creator or owner of the resource decides who can access it.
 3. **Flexible but Risky:** Users can share access, making it convenient—but also vulnerable to misuse.
 4. **Common in Commercial Systems:** DAC is widely used in systems like Windows, Linux, and Unix.
-

Basic Concepts

DAC typically involves:

- **Subjects:** Users or processes requesting access (e.g., Alice, a program).
- **Objects:** Resources being accessed (e.g., a file, database table).
- **Actions:** What the subject wants to do (e.g., read, write, execute).

These are usually defined in the form of:

(subject, object, action)

Access Matrix Model (Core of DAC)

Imagine a table (called an Access Matrix) where:

- **Rows = Users (subjects)**
- **Columns = Files or objects**
- **Cell = Allowed actions (like read, write)**

	File1	File2
Alice	read, write	read
Bob	read	—

Alice can both read and write File1, but Bob can only read it.

🧩 How DAC Works

Let's say Alice creates a file. She can:

- Read or write the file.
- Grant Bob read access.
- Later, revoke Bob's access if she wants.

This ability to delegate is why it's called discretionary—Alice has discretion over her file.

🔒 Mandatory Access Control (MAC)

✅ What is MAC?

Mandatory Access Control (MAC) is a strict access control model where the **system (not the user)** determines access permissions based on **predefined security policies**.

In MAC, access decisions are **not left to the discretion of users**. Instead, they're controlled by **security labels** and **rules** set by a central authority (e.g., a system administrator or security officer).

🔒 Key Characteristics

1. **System-enforced rules** — users cannot override them.
2. **Security Labels** assigned to both:
 - **Subjects** (e.g., users or processes)
 - **Objects** (e.g., files or database records)
3. Access is granted based on a **dominance relationship** (e.g., Clearance vs Classification).
4. Mostly used in **military, government, or high-security environments**.



Basic Concepts

- **Subject:** An active entity (e.g., a user or a process).
- **Object:** A passive entity (e.g., file, document, DB row).
- **Security Labels:**
 - Include a **security level** (e.g., *Top Secret*, *Confidential*, *Unclassified*).
 - Can also include **categories** (e.g., *Medical*, *Admin*).



Dominance Rule:

A subject **can access** an object **only if its label "dominates"** the label of the object.

1. No Read Up (NRU)

A subject **cannot read** an object **at a higher level** than their clearance.

2. No Write Down (NWD)

A subject **cannot write** to an object **at a lower level** than their clearance.



Real-World Analogy

Imagine a **military office**:

- Alice has **Secret clearance**.
- Bob has **Top Secret clearance**.
- A file is labeled **Top Secret**.

Access

Allowed?

Alice reads Top Secret file	✗ No (No Read Up)
Alice writes Top Secret file	✗ No (No Write Down)
Bob reads Secret file	✓ Yes
Bob writes to Secret file	✗ No (No Write Down)



Role-Based Access Control (RBAC)

✓ What is RBAC?

Role-Based Access Control (RBAC) is an access control model where **permissions are assigned to roles**, and **users are assigned to roles**.

Instead of assigning permissions directly to each user, you assign them to roles — and users gain permissions by being **members of those roles**.

✓ It answers: "What role does this person have, and what can people in that role do?"

🔄 How It Works

RBAC separates **user identity** from **access permissions** using **roles** as an intermediary:

1. **Users** ➡ assigned to ➡ **Roles**
2. **Roles** ➡ have ➡ **Permissions**
3. ✓ Users gain permissions **through their role(s)**

👛 Example Scenario: Hospital System

User	Assigned Role	Permissions Given by Role
Alice	Doctor	View patients, write prescriptions
Bob	Nurse	View patients
Charlie	Admin Staff	Manage appointments, update billing
David	Doctor	Same as Alice (inherited from Doctor role)

🧩 Core RBAC Components

1. **Users** – The people who use the system.
2. **Roles** – Job functions or responsibilities (e.g., Doctor, Manager, Student).
3. **Permissions** – Allowed actions (e.g., read file, write prescription).
4. **Sessions** – Temporary mappings of a user to a subset of their roles.

▲ Role Hierarchies (Optional but Powerful)

- Roles can be **structured hierarchically**, meaning:

- Senior roles inherit permissions from junior ones.

Example:

SeniorDoctor → Doctor → Intern

- A **SeniorDoctor** has all permissions of a **Doctor**, who has all permissions of an **Intern**.

CBAC — Credential-Based Access Control

✓ What is CBAC (Credential-Based)?

Credential-Based Access Control (CBAC) is an access control model where **access decisions are based on the user's credentials**, rather than their identity (like in DAC) or their role (like in RBAC).

A **credential** is a digitally verifiable proof of some attribute — like a certificate that proves you're a doctor, or a membership badge.

Example of Credentials

- Digital certificates (X.509)
 - University ID card
 - Professional license
 - Age/Location proof
 - Attribute assertions like:
"isDoctor", "isEmployee", "hasSecurityClearance: TopSecret"
-

Why CBAC?

In **open, distributed systems** (e.g., the internet), you often:

- Don't **know** the user beforehand.
- Don't want to **rely on usernames/passwords**.

- Want access decisions based on **what the user can prove**.

That's where CBAC comes in.

How CBAC Works

1. A user tries to access a resource.
2. The system **asks for credentials** (e.g., digital certificate).
3. Access is granted **if the submitted credentials satisfy the access policy**.

Real-World Example

Imagine an online pharmacy:

- Only **licensed pharmacists** can access the order system.
- The system doesn't care if the user is Alice or Bob.
- It checks if the user can **present a valid "pharmacist license credential"**.

If yes → access granted.

If not → access denied.

Access Control Models for XML

Why Access Control for XML?

XML (eXtensible Markup Language) is widely used to represent **structured, hierarchical data** (e.g., in web services, databases, configurations).

But traditional access control models like DAC, MAC, and RBAC aren't well-suited because XML is:

- **Hierarchical** (tree-structured)
- **Semi-structured** (schema may not be fixed)
- May require **fine-grained access** (e.g., to specific elements/attributes)

- ✓ Need: Enforce access not just on files or tables, but down to specific **elements or attributes** in an XML document.

Example XML Document

```
<Patient>

  <Name>John Doe</Name>

  <Diagnosis>AIDS</Diagnosis>

  <Insurance>

    <Provider>HealthFirst</Provider>

    <PolicyNumber>12345</PolicyNumber>

  </Insurance>

</Patient>
```

Different users might need:

- Doctors: full access
- Nurses: no access to `<Diagnosis>`
- Admins: only `<Insurance>` section

1. Subjects

→ Users or roles requesting access.

2. Objects

→ XML nodes (elements, attributes, subtrees).

3. Access Rights

→ Common actions:

- `read` (view node)
- `write` (modify node)
- `delete`, `append`, etc.

4. Access Rules

→ Defined using **XPath/XQuery expressions** to match parts of the XML tree.



Policy Components (Explained with Examples)

These components are used to define access control policies for XML data.

♦ 1. Target

What it is:

Specifies the **part(s) of the XML document** the policy applies to — usually expressed using **XPath** (a query language to navigate XML trees).

Example:

xml

CopyEdit

```
<Target>//Patient/Diagnosis</Target>
```

📌 This targets the `<Diagnosis>` element inside any `<Patient>` node.

♦ 2. Subjects

What it is:

Specifies **who the policy applies to** — could be individual users or user groups/roles (e.g., `doctor`, `nurse`, `admin`).

Example:

xml

CopyEdit

```
<Subject>role = nurse</Subject>
```

📌 This policy applies to all users with the role “nurse”.

♦ 3. Permission

What it is:

Specifies **what action** is allowed or denied on the targeted element.

Common permissions:


- **read**: View the data
- **write**: Modify the data
- **append**, **delete**, etc. (in some models)

Example:

xml

CopyEdit

```
<Permission>read</Permission>
```

 This refers to the read (view) operation.

♦ 4. Constraints (optional)

What it is:

Specifies **contextual conditions** under which the rule applies. This adds **dynamic filtering** based on:

- Time of day
- Location
- Device
- User attributes
- Request IP

Example:

xml

CopyEdit

```
<Constraint>time >= "09:00" AND time <= "17:00"</Constraint>
```

📌 Nurses can only read the diagnosis during working hours.

Another example:

xml

CopyEdit

```
<Constraint>request_ip = "192.168.1.*"</Constraint>
```

📌 Access only allowed from the hospital's internal network.

◆ 5. Effect

What it is:

Determines whether the policy **permits or denies** the action if all conditions match.

Values:

- Permit
- Deny

Example:

xml

CopyEdit

```
<Effect>Deny</Effect>
```

📌 If the conditions match, access is **explicitly denied**.

✅ Full Example Policy

xml

CopyEdit

```
<Policy>
```

```
<Target>//Patient/Diagnosis</Target>

<Subject>role = nurse</Subject>

<Permission>read</Permission>

<Constraint>time >= "09:00" AND time <= "17:00"</Constraint>

<Effect>Deny</Effect>

</Policy>
```

What this means:

If a **nurse** tries to **read** the **<Diagnosis>** element of a patient record **between 9 AM and 5 PM**, access will be **denied**.



Database Issues in Trust Management and Trust Negotiation



1. Introduction

- In open systems like the internet, users and services often don't know or trust each other in advance.
- Trust Management is the process of deciding whether to trust an entity (like a user or organization) based on credentials.
- Trust Negotiation enables gradual, automated exchange of credentials between untrusted parties to build trust over time.



Unlike static access control models, trust-based models work dynamically and rely on policies and evidence instead of identity.



2. Key Concepts

◆ Credentials

- Digitally signed assertions about a subject.
- Examples:

- "Alice is a licensed doctor."
- "This device is owned by Bob."
- Stored in distributed repositories or embedded in smartcards.

◆ Disclosure Policies

- Rules that control when and to whom a credential can be released.
- Example: "Only release my home address if the requester is an HR employee from the same company."

◆ Trust Management System

Handles:

- Credential discovery
 - Credential validation
 - Policy evaluation
-



3. Trust Negotiation



Definition

A protocol-driven interaction where two parties exchange:

- Access control policies
- Credentials
- Meta-policies (rules about policies)



Basic Flow

1. Party A requests access to a resource from Party B.
2. Party B requires credentials before granting access.
3. Party A replies with some credentials.

4. If needed, Party B may respond with its own requirements.
5. Iteratively builds trust → Access granted if policies are satisfied.

Goal:

Enable access without revealing unnecessary information.

4. Challenges in Databases

A. Efficient Credential Storage & Querying

- Need to store, index, and retrieve credentials quickly.
- Credentials might be:
 - Distributed across networks
 - Large in number (scalability challenge)

B. Query Rewriting for Policy Compliance

- When a user issues a query, it may be rewritten to:
 - Ensure only data matching credential-based policies is returned
 - Avoid disclosing restricted data

C. Policy Language Support

- Policies must be expressive but also efficient to evaluate.
- Languages like:
 - RT (Role-based Trust-management framework)
 - XACML (XML-based policy language)
 - Ponder, ASL, etc.

D. Credential Protection and Minimization

- Don't reveal more than necessary.

- Risk: Revealing a credential could help an adversary infer sensitive information.
- Solution: Use minimum disclosure strategies.

Security in Data Warehouses and OLAP Systems

What Is a Data Warehouse?

A Data Warehouse is a centralized system that stores large volumes of data collected from multiple sources.

- It's used mostly for analytics and decision-making, not for day-to-day operations.
- The stored data is usually historical and aggregated.

Example: A supermarket chain collects sales data from all its branches daily and stores it in a central warehouse to analyze trends.

What Is OLAP?

OLAP stands for Online Analytical Processing.

It allows users to perform fast, multi-dimensional analysis of data in a warehouse.

With OLAP, managers can ask complex questions like:

“What was the total revenue from electronics in all South Indian branches during the Diwali season over the past 5 years?”

Why Security Is Challenging in Warehouses and OLAP

Data warehouses:

- Combine data from many sources
- Contain highly sensitive information
- Are used for aggregated reporting, which can lead to inference attacks (explained below)

So, special attention is needed for data privacy, access control, and inference prevention.



Key Security Issues in Data Warehouses & OLAP

1. Inference Attacks

What is it?

An attacker gets access to high-level or aggregated data, then uses it to guess sensitive individual information.

 Example:

- A user queries:
“How many patients in the hospital are HIV positive?”
- Then runs a query filtered to a small group (like only one village).
- If the answer is "1", the user might infer the identity of that patient.



Prevention Techniques:

- Cell suppression: Hide cells with too few contributors (e.g., less than 3).
 - Query restriction: Don't allow overly specific queries.
 - Noise injection: Add small random changes to results.
-

2. Fine-Grained Access Control

What is it?

Not all users should see the same data. Access should be customized at a very detailed level, like:

- Specific rows (data about Mumbai only)
- Specific columns (exclude income, SSN)



Example:

A receptionist should see a patient's name and appointment time, but NOT their diagnosis or prescription history.



Techniques:

- Authorization views: Show only what the user is allowed to see.

- User-based policies: Define access per user role or credentials.
-

3. Multilevel Security (MLS)

What is it?

Each piece of data is labeled with a security classification, such as:

- Unclassified
- Confidential
- Secret
- Top Secret

Each user has a clearance level, and they can:

- Read data *only at or below* their level
- Write data *only at or above* their level



Example:

A manager with "Confidential" clearance cannot access "Secret" data.

This helps prevent accidental data leaks.

4. Security in Materialized Views

Materialized views are precomputed summary tables used to speed up queries.

Problem: They can leak sensitive data indirectly.



Example:

- Original table: Patient name and disease.
- View: Count of people with AIDS by ZIP code.
- A ZIP code with only 1 patient → identity revealed!



Solution:

- Control what gets materialized.
 - Don't allow views with low group counts.
 - Use view-level access policies.
-

5. Securing OLAP Cubes

An OLAP Cube is a multi-dimensional structure (e.g., sales by region, product, year).

- Each cube dimension may need separate access rules.
- Need to control:
 - Which measures a user can see (e.g., sales revenue, profit)
 - Which dimensions they can access (e.g., region, product)



Example:

A regional manager may only view data for South India, not all of India.



Solution:

- Cell-level security: Define access down to individual data points in the cube.
-

6. Data Integrity and Audit

Integrity: Make sure data hasn't been tampered with.

Auditing: Keep logs of who accessed what, and when.



Useful to catch:

- Unauthorized access
 - Query patterns suggesting misuse
 - Attempts to infer private data
-



Summary Table

Security Issue	Description	Real-World Example	Countermeasure
Inference Attack	Guessing private data via aggregates	Inferring 1 HIV patient in a ZIP code	Query limits, cell suppression
Fine-Grained Access	Custom visibility by user/role	Receptionist can't view diagnosis	Authorization views
Multilevel Security	Clearance-based access	Secret-level data hidden from staff	Classification labels
Materialized View Leak	Views reveal sensitive info	View with only 1 patient in a group	Thresholds, restricted views
OLAP Cube Exposure	Unauthorized cube drill-down	Manager sees all-region sales	Cell-level access control
Audit & Integrity	Track actions and prevent tampering	Check who queried patient stats at night	Logs, checksums