Database Security: Data Access Control

Overview

Q: Who is responsible for security of database applications?

Variant 1: Database software developers, in software layer.

Variant 2: Variant 1 is not always feasible. In such cases, access control is facilitated inside the DBMS.

Security in Databases

Secrecy: information should not be disclosed to unauthorized users.

Integrity: only authorized users should be allowed to modify data.

Availability: authorized users should not be denied services.

Breaches of Security

Database security can be breached as a result of many different *activities*. Among them are:

- Theft and fraud;
- Loss of confidentiality/secrecy;
- Loss of privacy;
- Loss of integrity;
- Loss of availability;

Threats. A **threat** is a situation of event, *intentional or unintentional*, thay may adversely affect a system.

Threats to database security. Below is the table of threats to database security and their relationship to the activities breaching security.

Threat	Theft/fraud	Loss of secrecy	Loss of privacy	Loss of integrity	Loss of availability
Use of access means of others	√	√	√		
Unauthorized amendment of data	√			√	
Unauthorized copying of data	√	√	√		
Program alteration	√			√	√
Mix of normal and confidential output	√	√	√		
Wiretapping	√	√	√		
Illegal entry by hacker	√	√	√		
Blackmail	√	√	√		
Creation of a "trapdoor"	√	√	√		
Theft of data, programs, equipment	√	√	√		√
Failure of security mechanisms		√	√	√	
Staff shortages/strikes			√	√	
Inadequate staff training		√	√	√	√
Disclosure of unauthorized data	√	√	√		
Electronic interference/radiation				√	√
Data corruption due to power loss/surge				√	√
Fire/flood/bomb				√	√
Physical damage to equipment				√	√
Cable disconnection				√	√
Introduction of viruses				√	√

Countermeasures: facilities used and activities conducted to alleviate **threats**.

- Authentication
- Authorization
- Access control
- Views
- Backup and recovery
- Integrity
- Encryption
- RAID arrays

Authentication: mechanism that determines whether a user is who (s)he claims to be.

- User accounts.
- Software user accounts.
- Passwords.
- Separate from OS accounts/ Joint with OS accounts.

Authorization: the granting of a right of privilege that enables a subject to have legitimate access to a system or a system's object.

Access Control in Databases

Access control: a mechanism for granting, maintaining and verifying users' authorizations.

- Discretionary Access Control: a system of data access permissions initiated and controlled by DBMS users.
- Mandatory Access Control: a system of universal data access rules obeyed by DBMS.

Discretionary Access Control

Privileges: data access rights possessed by DBMS users.

System Privileges: the rights to perform a particular action or to perform an action on any schema objects of a particular type. Oracle 11i has 166 system privileges[1] Examples:

- `CREATE USER`: the right to create a new user account.
- `CREATE ANY PROCEDURE`: the reight to create stored PL/SQL procedures.
- `CREATE TABLESPACE`: the right to create a new table space.

Object Privileges: the rights to perform a particular action on a specific table.

- `SELECT`: the right to view stored data.
- `INSERT`: the right to insert data.
- `DELETE`: the right to delete data.
- `UPDATE`: the right to modify existing data.
- `REFERENCES`: the right to create tables with foreign keys to the data.

These priveleges are typically asserted at the level of individual relational tables. If necessary, the privileges can be asserted at the level of individual columns of relational tables[2]

Authorization in Discretionary Access Control

In a standard DBMS (e.g., Oracle), each database access is associated with a user name. Conventions establish initial set of privileges any user has over the data managed by the DBMS.

Further privileges can be acquired (or lost) via special-purpose granting and revocation commands passed to the DBMS.

GRANT

To give a user a new privilege, an SQL `GRANT` command is used. The syntax is:

`GRANT` *privileges* `ON` *object* `TO` *users* [`WITH GRANT OPTION`]

Here,

- *privileges*: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REFERENCES`. These grant permissions on the entire *object* (typically, a relational table).
- `INSERT(Attribute-List)`, `UPDATE(Attribute-list)`, `REFERENCES(Attribute-List)` grant privileges only for access of the specified attributes from the *object*.
- `WITH GRANT OPTION`, when included, allows the user obtaining the privilege to grant this privilege to other users in turn.

[1] To find the list of privileges, execute the following SQL command: `SELECT name FROM sys.system_privilege_map;`

[2] Other priveleges exist in relational DBMS, however they refer to functionality not covered in this course.

Examples. User ST10 is issuing the following command:

```
GRANT SELECT ON Goods TO ST44;
```

Now, user ST44 has access to table Goods created by user ST10. In particular, user ST44 can now run the following query:

```
SELECT * FROM ST10.Goods;
```

(note that in Oracle, the convention is to specify the table name as userName.tableName.)

User ST10 is issuing the following command:

```
GRANT UPDATE(Food, Flavor, Price) ON Goods TO ST44 WITH GRANT OPTION;
```

User ST44 can now perform the following operations:

```
UPDATE ST10.Goods
SET Flavor = 'Chocolate', Food = 'Cake', Price=15.45
WHERE Gid = 10;

GRANT UPDATE(Price) ON ST10.Goods TO ST45;
```

As a result of the last operation, ST45 can now do the following:

```
UPDATE ST10.Goods
SET  Price=16.95
WHERE Gid = 10;
```

REVOKE

To take a privilege away, the REVOKE command is used.

> REVOKE [GRANT OPTION FOR] *priveleges* ON *object* FROM *users* { RESTRICT | CASCADE }

- REVOKE *privileges* ... revokes the specified privileges, **including** any GRANT OPTIONs on these priveleges that the users may have.
- REVOKE GRANT OPTION FOR *privileges* revokes only the GRANT OPTION, leaving the privilege itself intact.
- CASCADE: revokes privilege from the named user, and from any user who obtained the privilege *solely through* the current user.
- RESTRICT: rejects the command, if revoking privileges to named user(s) results in the need to revoke other privileges.

Problems with discretionary access control

Trojan Horse attacks. User who can access the database application software, but cannot access data, can create a new table, grant another user access to it, and insert malicious code, which would populate the created table with data selected from the tables the user does not have direct access to.

No row-based access control. GRANT provides equal access to ALL rows in a given table.

Ownership. Users have access to information they have created. In some applications, this is not desirable.

Mandatory Access Control: Bell-LaPadula Model

Mandatory Access Control is enforced when a higher level of security needs to be achieved. In particular, Mandatory Access Control.

Class. The mandatory access control schema (the Bell-LaPadula model) is based on associating with each subject a *security clearance* and with each object a *security class*, and in providing a mapping between security clearances and security classes.

In most simple cases, *security clearances* and *security classes* co-inside, and in such cases, we use *class(O)* and *class(S)* to denote the security class of an object O (a database table, for example, or a tuple in a table) and a security clearance of a subject S (a DBMS user).

The list of security classes/clearances **must be** partially ordered. In standard applications, it will be completely ordered.

The access rules are:

Simple Security Property: Subject S is allowed to read object O only if $class(S) \geq class(O)$.

***-property:** Subject S is allowed to write object O only if $class(S) \leq class(O)$.

Loopholes

Multilevel tables : tables that reveal different tuples (content) to users with different access privileges.

Example. Consider an example of a database using the **Bell-LaPadula mandatory access control** approach at the tuple level. Each tuple in the database tables has a special attribute, `SecurityClass`.

Conisder the set of security classes: $TS > S > C > U$ (Top Secret, Secret, Confidential, Unclassified.)

Consider the following table `SalesData`:

ReceiptNo	Customer	Item	Quantity	Paid	SecurityClass
4325	'Mary Lake'	756401	1	178.99	C
6374	'Albert Stover '	535623	1	27.79	C
3463	'Tom Cruise'	46367323	5	5,000.00	TS

Users of classes C will see only the first two tuples returned when the

```
SELECT * FROM SalesData;
```

query is issued.

Suppose user Bob with class C wants to run the following command:

```
INSERT INTO SalesData VALUES(3463, 'Lorem Ipsum', 444444, 4, 4.00);
```

This command will fail, because receipt `3463` is in the table with a higher security class.

Bob now knows that the `SalesData` table contains tuples Bob cannot access, and that at least one such tuple if for a receipt with number `3463`.

Polyinstantiation. Presence of data objects that appear to have different values to users with different clearances. Facilitated by the use of `securityClass` attribute as part of the primary key of the table.

In our example, **Polyinstantiation** would allow Bob's update request to succeed creating the following table:

ReceiptNo	Customer	Item	Quantity	Paid	SecurityClass
4325	'Mary Lake'	756401	1	178.99	C
6374	'Albert Stover '	535623	1	27.79	C
3463	'Tom Cruise'	46367323	5	5,000.00	TS
3463	'Lorem Ipsum'	444444	4	4.00	C