Australian National University

Database Security – Part 1

Introduction

Australian National University

Introduction

- **Why database security?**
  - Databases often store **data that are sensitive** in nature.
  - Databases need to preserve **data integrity**.
  - . . .

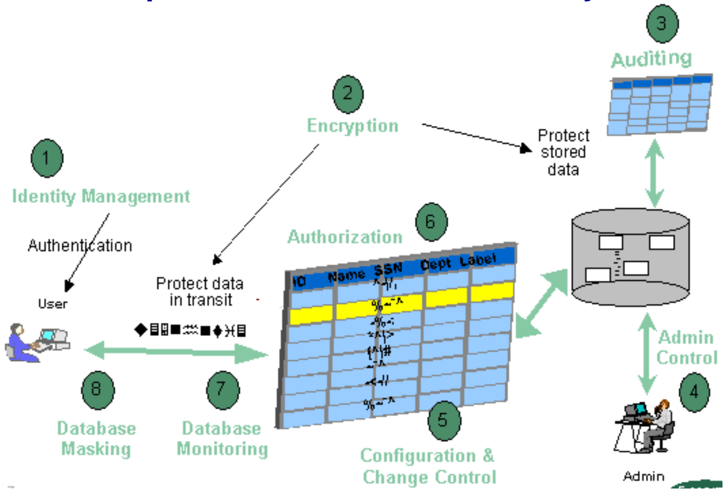
Example: Consider a payroll database, it must be ensured that:

- Salaries may not be disclosed to arbitrary users of the database;
- Salaries can only be modified by users that are properly authorized.

Introduction

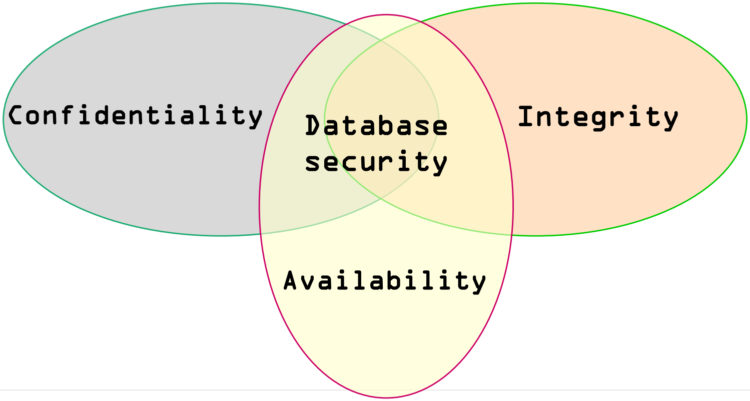
- The protection which database security gives is usually directed **against two cases**:
    - Stop users without database access from having any access;
    - Stop users with database access from performing actions on the database which are not required to perform their duties.

Comprehensive Database Security[1]



1 Oracle database 11g security: data masking, Jonathan Penn, Forrester Research

Main Objectives of Database Security



Threats to Databases

- A potential **breach of security** that, if successful, will have a certain impact on databases.

Threats to Databases

- A potential **breach of security** that, if successful, will have a certain impact on databases.
  - **Loss of confidentiality**: data should not be accessible to those who do not have legitimate access rights,

    e.g., *a student is not allowed to view grades of other students.*

Threats to Databases

- A potential **breach of security** that, if successful, will have a certain impact on databases.
    - **Loss of confidentiality**: data should not be accessible to those who do not have legitimate access rights,

      e.g., *a student is not allowed to view grades of other students.*
    - **Loss of integrity**: data should not be corrupted, through intentional or accidental acts,

      e.g., *students are allowed to see their grades, yet not allowed (obviously) to modify them.*

Threats to Databases

- A potential **breach of security** that, if successful, will have a certain impact on databases.
    - **Loss of confidentiality**: data should not be accessible to those who do not have legitimate access rights,

      e.g., *a student is not allowed to view grades of other students.*
    - **Loss of integrity**: data should not be corrupted, through intentional or accidental acts,

      e.g., *students are allowed to see their grades, yet not allowed (obviously) to modify them.*
    - **Loss of availability**: data should remain accessible to those who have legitimate access rights,

      e.g., *a lecturer is allowed to change grades of students.*

Control Measures

1. **Access control**
   - Restrict access to the database system,

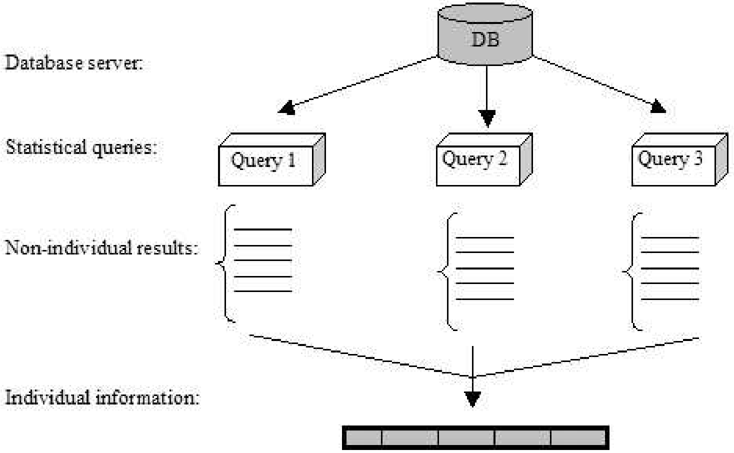
     e.g., *user accounts and passwords*.
2. **Inference control**
   - Ensure that data that users are not authorized to access cannot be inferred from statistical or summary data,

     e.g., *know the average salary of a department, but don't know the salary of a particular person*.
3. **Flow control**
   - Prevent data to flow into unauthorized users,

     e.g., *avoid covert channels*.
4. **Data encryption**
   - Protect sensitive data during storage and transmission,

     e.g., *passwords and credit card information*.

Inference Attack[2]



[2] Inference Attacks to Statistical Databases: Data Suppression, Concealing Controls and Other Security Trends, 2000

Inference Attack - Examples

- An extensive data re-identification experiment run in 1990 by the United States Government:

  87% of 248 million US citizens could be uniquely identified based on the combination of gender, date of birth and a five-digit ZIP code.

Inference Attack - Examples

- Suppose that we have a database which contains information of employees, including names, ages and salaries, and only allows aggregation queries. If we happen to know that Peter is the oldest employee in the company, can we infer the salary of Peter through aggregation queries?

Inference Attack - Examples

- Suppose that we have a database which contains information of employees, including names, ages and salaries, and only allows aggregation queries. If we happen to know that Peter is the oldest employee in the company, can we infer the salary of Peter through aggregation queries?

  (1) We could repeatedly ask: "*How many employees are there whose age is greater than X?*" until the answer is 1

  (2) Then we could ask: "*what is the average salary of all employees whose age is greater than X?*".

Australian National University

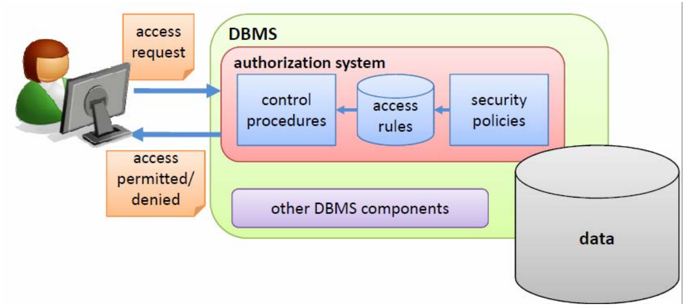
Database Security – Part 2

Access Control

Australian National University

Access Control

- **Access Control** refers to any means of controlling access to resources in a database.
- Can be seen as **the combination of authentication and authorization** plus **additional measures**, such as IP-based restrictions.



Authentication vs. Authorization

- **Authentication** is the process by which a system can identify users.
  - Who are the users?
  - Are the users really whom they represent themselves to be?

  – Identified by username/password, a smart card, a PIN number, a secret code sent in a letter, a fingerprint scan, and so on.

- **Authorization** is the process by which a system determines what level of access a user (who **is already authenticated**) has to secured resources.
  - Is a user authorized to access or modify a table?
  - ...

Main Approaches to Access Control

1. **Discretionary access control** (**DAC**)
    - Based on the concept of **access privileges** for giving users such privileges.
    - SQL support DAC; most commercial DBMSs also support DAC.
2. **Mandatory access control** (**MAC**)
    - Based on **system-wide policies** that cannot be changed by individual users.
    - SQL doesn’t support MAC but some DBMSs support MAC.
3. **Role-based access control** (**RBAC**)
    - Based on **roles** (can be used with DAC and MAC).
    - SQL support privileges on roles; many DBMSs support RBAC.

Discretionary Access Control (DAC)

- Called **discretionary** because it allows a subject to grant other subjects privileges to access objects of the subject at its own discretion.
- DAC governs the access of subjects (e.g. accounts, etc.) to objects (relations, views, etc.) **on the basis of subjects' privileges**.
- SQL supports DAC through the `GRANT` and `REVOKE` commands.
  - `GRANT` gives privileges to users;
  - `REVOKE` takes away privileges from users.

Australian National University

Specifying Privileges - Grant

- The **syntax** of the GRANT command:

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

Examples: Consider the relation schemas

SUPPLIER(<u>id</u>, sname, city, rating)
RATINGSTANDARD(<u>no</u>, description)

1. `GRANT SELECT ON SUPPLIER TO Jerry;`
2. `GRANT INSERT, DELETE ON SUPPLIER TO Tom;`
3. `GRANT UPDATE (rating) ON SUPPLIER TO Tom;`
4. `GRANT REFERENCES (no) ON RATINGSTANDARD TO Bob;`

Australian National University

Specifying Privileges - Views

- Views provide **an important mechanism for discretionary authorization**.
- The **syntax** of creating a view:

```
CREATE VIEW view_name AS
      SELECT attribute_list
        FROM table_list
      [WHERE condition]
   [GROUP BY attribute_list [HAVING group_condition]]
   [ORDER BY attribute_list];
```

- Creating a view requires `SELECT` privilege on all relations involved in the view definition.

Specifying Privileges - Views

- **Example:** Consider the relation schema:

  SUPPLIER(<u>id</u>, sname, city, rating)

  How to give Bob read access to SUPPLIER *for suppliers in Paris (only), but not to supplier ratings?*

Specifying Privileges - Views

- **Example:** Consider the relation schema:

  SUPPLIER(id, sname, city, rating)

  How to give Bob read access to SUPPLIER *for suppliers in Paris (only), but not to supplier ratings?*

```
Step 1:  CREATE VIEW SUPPLIER-PARIS AS
              SELECT id, sname, city
              FROM SUPPLIER
              WHERE city='Paris';

Step 2:  GRANT SELECT ON SUPPLIER-PARIS TO Bob
```

  Users of this view only see part of SUPPLIER (**horizontal subset** by applying `city='Paris'` and **vertical subset** by excluding `rating`).

Revoking Privileges - Revoke

- The **syntax** of the REVOKE command:

```
REVOKE [GRANT OPTION FOR] privileges ON object FROM users
```

Examples: Still consider the relation schema

SUPPLIER(id, sname, city, rating)

1. `REVOKE INSERT, DELETE ON SUPPLIER FROM Peter;`
2. `GRANT SELECT ON SUPPLIER TO Bob;`

   Bob is working on the task ... and done!

   `REVOKE SELECT ON SUPPLIER FROM Bob;`

Delegating Privileges

- **Can we pass on privileges to others?**
  - We are the object owner;
  - We have received the privilege with `GRANT OPTION`.

Example: Tom, the owner of SUPPLIER, wants to give Bob the right to grant his SELECT privilege on SUPPLIER to other users for one month.

```
GRANT SELECT ON SUPPLIER TO Bob WITH GRANT OPTION;
```

```
One month later ...
```

```
REVOKE GRANT OPTION FOR SELECT ON SUPPLIER FROM Bob;
```

Delegating Privileges - Recursive Revocation

- The privileges of an object can be given to a user *with* or *without* the `GRANT OPTION`

```
GRANT SELECT ON Supplier TO Bob;
```

```
GRANT SELECT ON Supplier TO Bob WITH GRANT OPTION;
```

- A user can only revoke privileges that he or she has granted earlier, with two optional keywords in `REVOKE` command:
    - `CASCADE`: revoking the privilege from a specified user also revokes the privileges from all users who received the privilege from that user.
    - `RESTRICT`: revoking the privilege only from a specified user.

Delegating Privileges - Recursive Revocation

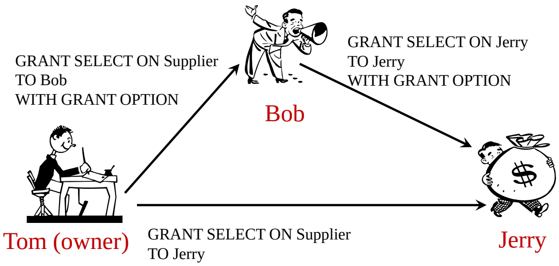
- If a user receives a certain privilege from multiple sources, and the user would lose the privilege only after all sources revoke this privilege.
- **Example:**

```
1.GRANT SELECT ON SUPPLIER TO Bob WITH GRANT OPTION;  (by Tom)

2.GRANT SELECT ON SUPPLIER TO Jerry;  (by Tom)

3.GRANT SELECT ON SUPPLIER TO Jerry WITH GRANT OPTION;  (by Bob)

4.REVOKE SELECT ON SUPPLIER FROM Bob CASCADE;  (by Tom)
```

- **Questions:**
  1. Will Bob lose the SELECT privilege on SUPPLIER?
  2. Will Jerry lose the SELECT privilege on SUPPLIER?

Delegating Privileges - Recursive Revocation

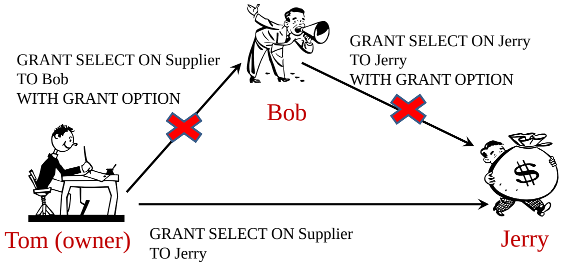
- **Example:**



```
1.GRANT SELECT ON Supplier TO Bob WITH GRANT OPTION;  (by Tom)
2.GRANT SELECT ON Supplier TO Jerry;  (by Tom)
3.GRANT SELECT ON Supplier TO Jerry WITH GRANT OPTION;  (by Bob)
```

Delegating Privileges - Recursive Revocation

- **Example:**



```
4.REVOKE SELECT ON Supplier FROM Bob CASCADE;  (by Tom)
```

1. Bob will lose the privilege.
2. Jerry won't lose the privilege.

Delegating Privileges - Propagation

- There are techniques to limit the propagation of privileges. But not implemented in most DBMSs and not part of SQL.

  - Limiting **horizontal propagation**: limits that an account given the `GRANT OPTION` can grant the privilege to at most n other accounts;

  - Limiting **vertical propagation**: limits the depth of the granting privileges.

Mandatory Access Control (MAC)

- Restrict access to objects based on the **sensitivity of the information** contained in the objects and the formal **authorization** of subjects to access information of such sensitivity.
  - Sensitivity of the information (e.g., security classes) top secret (TS), secret (S), confidential (C), unclassified (U).

$$TS \geq S \geq C \geq U$$

  - Authorization (e.g., clearances)

Example:

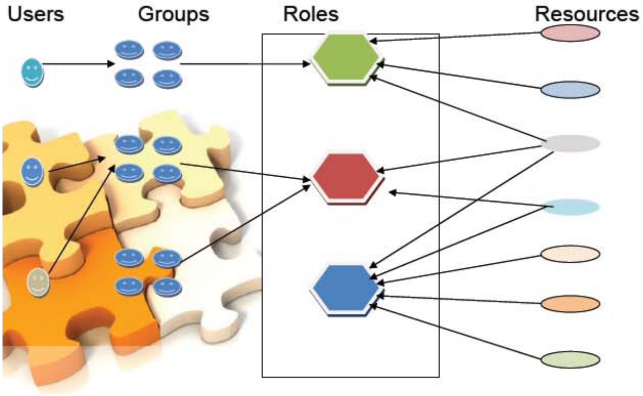
id	sname	city	rating	security class
1	S1	Paris	4	secret (S)
2	S2	Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

Australian National University

Role-Based Access Control (RBAC)[1]

- Access rights are grouped by **roles**, and the use of resources is restricted to individuals assigned to specific roles.



[1] Comprehensive Approach to Database Security, Ajoy S. Kumar, 2008

Australian National University

Database Security – Part 3

SQL Injection

A Survey

- A survey by GreenSQL (http://www.greensql.com, released on April 3, 2012) shows that **the most critical database security concerns** are:

51%	**SQL injection** attacks from internal and external users
31%	**Internal threats**, including unauthorized database access, database administrator errors, and data exposure to non-privileged internal users
18%	**Regulatory compliance**

- Surveyed more than six thousand users – IT administrators, DBAs, data security professionals and consultants.

SQL Injection

- SQL injection is mostly known as an attack for **web applications** (considered as **one of the top 10 web application vulnerabilities** of 2007 and 2010 by the Open Web Application Security Project).
- SQL injection can be used to attack any type of SQL databases.
- Web applications often access a database by
  1. Connect to the database;
  2. Send SQL statements and data to the database;$\hookleftarrow$ **SQL Injection!**
  3. Fetch the result and display data from the database;
  4. Close the connection.

Australian National University

What is SQL Injection?

- In **SQL injection attacks**, hackers inject a string input through the Web application which changes the SQL statement to their advantages.
- It can harm the database in various ways:
  - change the database content,
  - retrieve sensitive data in the database like credit card or passwords,
  - execute system level commands that may cause the system to deny service to the application,
  - ...

What is SQL Injection?

- In **SQL injection attacks**, hackers inject a string input through the Web application which changes the SQL statement to their advantages.
- It can harm the database in various ways:
  - change the database content,
  - retrieve sensitive data in the database like credit card or passwords,
  - execute system level commands that may cause the system to deny service to the application,
  - ...

A credit card payment processing company called CardSystems had an SQL injection attack in June 2005, in which 263,000 credit card numbers were stolen from its database. CardSystems lost large amounts of business and its assets were acquired by another company.

SQL Injection – Example

- The following query is issued by a simplistic authentication procedure:

```
SELECT * FROM users WHERE name='jake' and password='passwd';
```

SQL Injection – Example

- The following query is issued by a simplistic authentication procedure:

```
SELECT * FROM users WHERE name='jake' and password='passwd';
```

- The attacker can change the SQL statement as follows:

```
SELECT * FROM users WHERE name='jake'
                   AND password='p' OR 'x'='x';
```

```
SELECT * FROM users WHERE name='jake'
                   AND password='p'; DROP TABLE users; --;';
```

SQL Injection – Example

- The following query is issued by a simplistic authentication procedure:

```
SELECT * FROM users WHERE name='jake' and password='passwd';
```

- The attacker can change the SQL statement as follows:

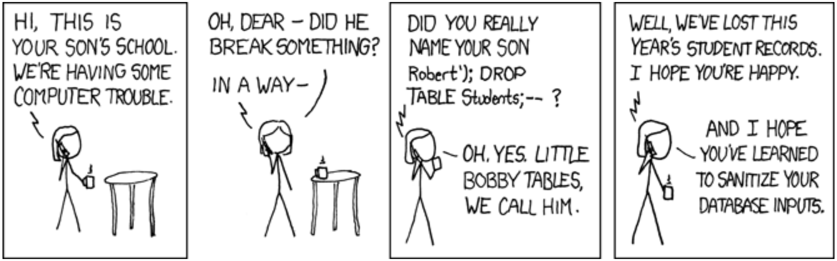
```
SELECT * FROM users WHERE name='jake'
                  AND password='p' OR 'x'='x';
```

```
SELECT * FROM users WHERE name='jake'
                  AND password='p'; DROP TABLE users; --;';
```

- Because the query here is constructed from strings, the use of quotes has turned the original WHERE condition into a condition **that is always true**.

Australian National University

SQL Injection – Another Example


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

link to this comic: http://xkcd.com/327/

SQL Injection – Protection Techniques

- Protection against SQL injection attacks can be achieved by applying certain rules to all Web-accessible procedures and functions.
  - **Parameterized queries** is used to improve security by preventing SQL injection (best solution). For example,

```
PreparedStatement stmt=conn.prepareStatement(
      "SELECT * FROM users WHERE name=? and password=?");
stmt.setString(1, user_name);
stmt.setString(2, user_passwd);
```

  - **Input validation** is used to remove or escape characters from input strings. For example,

```
"SELECT * FROM users WHERE name = '" + escape(user_name) +
      "' and password= '" +escape(user_passwd) +"'"
```

    In this case, user_name can't be `p' OR 'x'='x`.

Summary

- Database security is critical.
  - Ensure that only authenticated users can access the system (i.e., **authentication**).
  - Ensure that authenticated users can access only data/interfaces that they are authorized to access (i.e., **authorization**).
- SQL injection attacks are an important security threat. Nevertheless, avoiding SQL injection vulnerabilities is much easier than you might think.
  - SQL Injection (`http://www.w3schools.com/sql/sql_injection.asp`)
  - Testing for SQL Injection (`https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)`)
  - SQL Injection Prevention Cheat Sheet (`https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet`)
  - "**Foundations of Security: What Every Programmer Needs To Know**" by Neil Daswani, Christoph Kern, and Anita Kesavan