Security Requirements

- **Confidentiality**
  - Users must not be able to see data they are not supposed to
  - E.g., a student can’t see other students’ grades

- **Integrity**
  - Users must not be able to modify data they are not supposed to
  - E.g., only instructors can assign grades

- **Availability**
  - Users must be able to see and modify data they are allowed to

Definitions

- **Security policy**
  - Specifies who (what user) is authorized to do what

- **Security mechanism**
  - Way to enforce a chosen security policy

Terminology

- **Users**
  - Data = Objects (tables, views, stored procedures, etc.)

Two important functions needed to achieve security

- **Authentication (AuthN)**
- **Authorization (AuthZ)**

Authentication

- Establishing the identity of the user, or who the user is

Users present authentication credentials

- Username/Password combination – “what user knows”
- Digital certificates (cryptographic tokens) – “what user has”
- Biometrics – “what user is”

Authentication mechanism not covered by SQL 92/99, but resulting “authorization id” has specified handling.

Some credential types stronger than others

- For high-security applications, multi-factor authentication
  - E.g., password + fingerprint

Authorization

- Once we know who the user is, what can s/he access?
  - What objects (tables, views, etc.) the user allowed access to?
  - What kind of operations is the user allowed to perform?
    - Read-only, modify, append
  - Authorization also referred to as access control

Two main categories of access control

- **Discretionary**: object owner decides authorization policy for its objects (Unix system, SQL)
- **Mandatory**: system-wide rules that dictate who gets to access what (multi-level security, Bell-LaPadula) (we won’t cover)

Discretionary Access Control

- Based on the concept of access rights or privileges
  - Privileges for objects (tables, views, etc.)
  - Mechanisms for granting and revoking privileges

Object creator automatically gets all privileges on it

- DBMS keeps track of who subsequently gains and loses privileges
- DBMS ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed
  - Example: user joe creates a table in a schema he owns; he can now alter it, drop it, insert to it, etc. User sally can’t access it unless joe arranges that access.
The System-privileged users (DBAs)

- How did user joe come to own the schema?
  - Like superusers in UNIX, each database has users with System privileges.
  - System-privileged users are called DBAs (database admins)
  - DBAs create users (including more DBAs) and schemas for them to own.
  - "create user" is not covered by the SQL standard, allowing variation between products:
    - Oracle "create user" also creates a schema of same name owned by the new user.
    - Mysql separate "create user" and "create database", which creates a schema of any specified name (we use "username@db") that arranges access to the new schema for the new user.
  - The SQL standard (SQL92, SQL-2003) assumes that each user has a "authorization_id", obtained somehow.

GRANT Command: Entry SQL92

```
GRANT privilege_list ON object TO user_list [WITH GRANT OPTION]
```

- The following privileges can be specified:
  - SELECT
  - INSERT(col-name)
  - UPDATE
  - DELETE
  - REFERENCES (col-name)
  - CREATE, ALTER, and DROP for tables by the SQL standard, but most databases provide ways for ordinary users (non-DBAs) to do CREATE, ALTER, etc., when given permission.
  - Oracle and mysql provide an ALTER privilege for a table, which can be given to ordinary users (to use on tables outside their own schema).
  - Mysql additionally provides CREATE and DROP table privs that can be given to ordinary users. Oracle calls these CREATE ANY TABLE and DROP ANY TABLE system privileges.

Examples

- GRANT INSERT, SELECT on Sailors TO Horatio
  - Horatio can query Sailors or insert tuples into it.
  - If the granting user has schema "sally", Horatio can "select * from sally.sailors;"

- GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
  - Yuppy can delete tuples, and also authorize others to do so.
  - Yuppy can "grant delete on sally.sailors to horatio"

- GRANT UPDATE (rating) ON Sailors TO Dustin
  - Dustin can update (only) the rating field of Sailors tuples

- GRANT SELECT ON sally.Sailors to PUBLIC (executed by sally or a DBA)
  - Like yelp._db tables, any user can read them.

Grant on views

- Need select priv on all base tables to create a view
  - View will be dropped if needed select priv dropped
  - Additional privs given on base tables automatically provides additional privs on (updatable) views

- Can grant privs on views to other users, regardless of their privs on base tables
  - This provides an important way to provide part of data to others
  - Does depend on the continuing select priv of the user who did the grant

- Grant on views: Example
  - User joe, with table Sailors in his schema, does:
    create view bestsailors (name, age, rating) as select s.name, s.age, s.rating from sailors where rating = 10;
    grant select on bestsailors to sue;
  - Now user sue can access bestsailors but not sailors:
    select * from joe.sailors;
    Error: Either no such table or no privileges on table select * from joe.bestsailors;

  Note: this could be Oracle, with schemas based on username, or mysql, with schema/database joe, also user joe.
**REVOKE Command**

```
REVOKE [GRANT OPTION FOR] privilege_list ON object
   FROM user_list [CASCADE | RESTRICT]
```

- **REVOKE**: Revokes privileges
- **CASCADE**: when a privilege is revoked from X, it is also revoked from all users who got it solely from X
  - Such a revoked privilege is said to be **ABANDONED**
  - A graph with the granting relationship is maintained
- **RESTRICT**: if revoke would cause some privilege to be abandoned, it is NOT executed

**Authorization Graph (FYI)**

- Keeps track of active authorization on objects
- Each authorization ID (user) corresponds to a node
- Granting a privilege adds labeled edge to graph
- Removing privilege deletes one or more edges from graph
- Special "System" node that originates all privileges
- Note: it is possible to have multiple edges between same pair of nodes (with same direction)!

How to determine if access is allowed for an ID?
- There must be a path from System to that ID formed of privileges equal (or stronger) than the one required

**Authorization Graphs**

- Complex situations can occur—how is authorization computed by the database?
- Example in book, pg 700
- Table Sailors is in Joe's schema
  - Joe: Grant Select on Sailors to Art with grant option;
  - Joe: Grant select on sailors to Bob with grant option;
  - Now Art and Bob can access joe.sailors, and do grants
  - Art: Grant select on joe.sailors to bob with grant option;
  - Joe: revoke select on sailors to Art cascade.
- Question: Can Art and/or Bob still access sailors?
- Let's look at this with authorization graphs…

**Authorization Graph**

Joe: CREATE TABLE T … in a schema he owns

```
<table>
<thead>
<tr>
<th>Sys</th>
<th>ALLPRIV, Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td></td>
</tr>
<tr>
<td>Art</td>
<td></td>
</tr>
<tr>
<td>Cal</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>
```

Joe: GRANT SELECT ON T TO Art WITH GRANT OPTION

```
<table>
<thead>
<tr>
<th>Sys</th>
<th>ALLPRIV, Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>SELECT, Yes</td>
</tr>
<tr>
<td>Art</td>
<td></td>
</tr>
<tr>
<td>Cal</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>
```

Joe: GRANT SELECT ON T TO Bob WITH GRANT OPTION

```
<table>
<thead>
<tr>
<th>Sys</th>
<th>ALLPRIV, Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>SELECT, Yes</td>
</tr>
<tr>
<td>Art</td>
<td></td>
</tr>
<tr>
<td>Cal</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>
```
Art: GRANT SELECT ON T TO Bob WITH GRANT OPTION

Authorization Graph

Joe
Sys
Art
Cal
Bob

Joe: REVOKE SELECT ON T FROM Art CASCADE

Authorization Graph

Joe
Sys
Art
Cal
Bob

Bob retains SELECT priv on T, because he got it independently from Joe

More Surprising Sequence: with cycle in graph

Text, pg. 701-702
Joe: GRANT SELECT ON Sailors TO Art WITH GRANT OPTION
Art: GRANT SELECT ON Sailors TO Bob WITH GRANT OPTION
Bob: GRANT SELECT ON Sailors TO Art WITH GRANT OPTION
Joe: GRANT SELECT ON Sailors TO Cal WITH GRANT OPTION
Cal: GRANT SELECT ON Sailors TO Bob WITH GRANT OPTION
Joe: REVOKE SELECT ON Sailors FROM Art CASCADE

Authorization Graph

Joe
Sys
Art
Cal
Bob

Joe: CREATE TABLE T ...

Authorization Graph

Joe
Sys
Art
Cal
Bob

Joe: GRANT SELECT ON T TO Art WITH GRANT OPTION

Authorization Graph

Joe
Sys
Art
Cal
Bob

Art: GRANT SELECT ON T TO Bob WITH GRANT OPTION
Authorization Graph

Joe: GRANT SELECT ON T TO Art WITH GRANT OPTION

Cal: GRANT SELECT ON T TO Bob WITH GRANT OPTION

Joe: REVOKE SELECT on T FROM Art CASCADE

Art, Bob can still access T!
Art uses arc from Bob even though it originally depended on the now-revoked link from Joe to Art!
No “temporal order” memorized

Security at the Level of a Field!
- Can create a view that only returns one field of one tuple
  - Then grant access to that view accordingly
- Allows for arbitrary granularity of control, but:
  - Tediou to specify and maintain policies
  - Performance is unacceptable
  - Too many view creations and look-ups
Access Security/Authentication Examples

- Amazon can simply use password authentication
- Alice logs into her Amazon account: user authentication
- SSL (secure sockets layer) used to establish a session key
- Transmission of the password must be secure!
- SSL is set up first using just server authentication, so it's working by the time the user password is sent to Amazon.

We log into pe07 using ssh: SSL is in use here too

- So our Oracle password cannot be seen by other Internet traffic
- But other logged-in users on pe07 may see it if used on the command line (sqlplus user@//dbs3.cs.umb.edu/dbs3)
- To avoid this, use /nolog on the command line, then connect inside SQLPlus, as follows (from DatabaseSetup.html):

```
pe07$ sqlplus /nolog
SQL*Plus: Release 12.1.0.2.0 Production on ...
Copyright (c) 1982, 2014, Oracle. All rights reserved.
SQL> connect scott/tiger1@//dbs3.cs.umb.edu/dbs3
Connected.
```

Mandatory Access Control: FYI

- Based on system-wide policies that cannot be changed by individual users (even if they own objects)
- Each DB object is assigned a security class
- Each subject (user or user program) is assigned a clearance for a security class
- Rules based on security classes and clearances govern who can read/write which objects.

Many commercial systems do not support mandatory access control

- Some specialized versions do
  - e.g., those used in military applications

Bell-LaPadula Model (FYI)

- Security classes:
  - Top secret (TS)
  - Secret (S)
  - Confidential (C)
  - Unclassified (U):
    - TS > S > C > U

Each object (O) and subject (i.e. user) (S) is assigned a class

- $ can read O only if class(S) >= class(O) (Simple Security Property or No Read Up)
- $ can write O only if class(S) <= class(O) (*-Property or No Write Down)

Intuition

- Idea is to ensure that information can never flow from a higher to a lower security level
- The mandatory access control rules are applied in addition to any discretionary controls that are in effect