Security:
Chapter 12

CIS 399, Week 5
Chapter Overview

Creating user accounts and roles
Controlling access to objects through roles
Limiting access to data through views and application controls
Encrypting data
Establishing and Monitoring security audits
Security Overview

• Confidentiality:
  only authorized people are allowed to see the data

• Integrity:
  the data stored accurately represents the concepts it is recording (prevent corruption by modification or failure)

• Availability:
  the system must be available when it is needed (backups, timely administration)
User Authentication

Who are you? Can you prove who you are?

• Challenge–response: username/password
  – Easy, powerful, but many problems

• Single sign–on
  – Requires complex configuration, additional hardware/software resources
    • Oracle Internet Directory (OID)
      – Supports the Lightweight Directory Application Protocol (LDAP)
      – Can use credentials from any LDAP authentication server
Creating Users

1. CREATE USER <username> IDENTIFIED BY <password> DEFAULT TABLESPACE <tablespace name>;
2. ALTER USER <username> QUOTA <quota> ON <tablespace name>;
3. GRANT CREATE SESSION TO <username>
4. GRANT <role name(s)> TO <username>

Example:
CREATE USER tsmith
    IDENTIFIED BY q1w2e3r4 <-- something secure
    DEFAULT TABLESPACE khuck;  <-- use YOUR space
ALTER USER tsmith QUOTA unlimited ON khuck;
GRANT CREATE SESSION TO tsmith;
GRANT webuser TO tsmith;
User Roles

You shouldn’t grant/revoke privileges to individual users
  – Reduces complexity when maintaining account privileges

When adding new users, only have to grant them the appropriate roles

When adding new objects, only have to grant access to the appropriate roles
Granting System and Object Privileges

GRANT <privilege | role> [, <privilege>] …] 
| ALL PRIVILEGES
ON <object>
TO <user> [IDENTIFIED BY <password>] 
[, <user2>] …]
| PUBLIC
[WITH ADMIN OPTION]
Common System Privileges

ALTER SESSION
ALTER SYSTEM
ALTER USER
AUDIT ANY
BACKUP ANY TABLE
CREATE PROCEDURE
CREATE ROLE
CREATE SEQUENCE
CREATE TABLE
CREATE TRIGGER
CREATE TYPE
CREATE USER
CREATE VIEW
DROP USER
FORCE TRANSACTION

Full list: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/statements_9013.htm#i2077938
# Common Object Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Table</th>
<th>View</th>
<th>Sequence</th>
<th>Procedure, Function, or Package</th>
<th>User Defined Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTE</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DEBUG</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FLASHBACK</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ON COMMIT REFRESH</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY REWRITE</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFERENCES</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Separation of Duties

Provide the the *SMALLEST* set of permissions possible to each role so those users can do their jobs effectively

1. Identify the primary objects in your database

2. Identify the primary roles needed by users of the database, keeping in mind the importance of separation of duties

3. Determine the permissions that each role needs for each object
Controlling User Access to Objects

By default, users have NO permissions

1. Test the system with each role, and make sure the user can do their job
2. Try to attack the system as that user/role, and make sure the system stops them

Be careful with the WITH ADMIN clause…
## New Roles for the database

<table>
<thead>
<tr>
<th>Database Object</th>
<th>Agent role</th>
<th>Manager role</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agents</td>
<td>Select</td>
<td>All</td>
</tr>
<tr>
<td>ContactReason</td>
<td>Select</td>
<td>All</td>
</tr>
<tr>
<td>CustAgentList</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Customers</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>LicenseStatus</td>
<td>Select?</td>
<td>All</td>
</tr>
<tr>
<td>Listings</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Properties</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>SaleStatus</td>
<td>Select</td>
<td>All</td>
</tr>
<tr>
<td><strong>Forms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AgentContacts</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>AgentListings</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Agents</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ContactReason</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Customers</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>LicenseStatus</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ListingActivity</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Listings</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Management</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Properties</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>SaleStatus</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Search4</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Startup</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Object</th>
<th>Agent role</th>
<th>Manager role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AgentSales</td>
<td>Maybe for self</td>
<td>All</td>
</tr>
<tr>
<td>CustomerList</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ListingsAndOffers</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>SalesByCityAndAgent</td>
<td>Maybe for self</td>
<td>All</td>
</tr>
<tr>
<td>SalesByMonthAndAgent</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

Week 4, Summer 2006  
CIS 399 - Intro to Oracle and SQL  
Page 12
Creating, listing Roles

CREATE ROLE <role name>;
GRANT <role name> TO <user name>;

DESCRIBE dba_role_privs; --as sysdba
SELECT granted_role FROM dba_role_privs; --as sysdba

SELECT * FROM user_role_privs;
SELECT Role FROM session_roles;
Granting and Revoking Permissions

See /space/ordata/classfiles/Chapter12/SQL/RealtyPermissions.sql

Make sure you run the following two scripts, first:

/space/ordata/classfiles/Chapter11/SetupDatabases/RedwoodRealty/BuildRedwood.sql
/space/ordata/classfiles/Chapter11/SetupDatabases/RedwoodRealty/RedwoodSQL811.sql
Enforcing Privileges Through Views and Procedures

GRANT command can limit access to individual columns, but not to individual rows.

Views can limit access to rows and columns.

See /space/ordata/classfiles/Chapter12/SQL/Views.sql
Restricting Access to Selected Rows and Columns

CREATE OR REPLACE VIEW MarcouxListings
AS
SELECT * FROM Listings
WHERE ListingAgentID = 10041;

GRANT SELECT ON MarcouxListings
TO MarcouxK;

See /space/ordata/classfiles/Chapter12/SQL/Views.sql
Restricting Updates Through Procedures

If we do this:
GRANT INSERT ON MarcouxListings TO MarcouxBK;

...then Kai Marcoux can do this...:
INSERT INTO MarcouxListings (ListingID, PropertyID, ListingAgentID, SaleStatusID) VALUES (99,773,10041,101);

...but he can also do this!
INSERT INTO MarcouxListings (ListingID, PropertyID, ListingAgentID, SaleStatusID) VALUES (99,773,\textbf{12963},101);
Restricting Updates through Procedures

Solution: only allow INSERT/UPDATE/DELETE operations through PROCEDUREs that can validate the data.

See

/space/ordata/classfiles/Chapter12/SQL/Procedure.sql
Protecting Data with Encryption

Encryption mixes plain-text data with a key value and produces scrambled, encrypted text that ‘hides’ the original data.

**Single key encryption**: The same key is used to encrypt and decrypt the data (faster, but need to control the key), used for storing data.

**Dual key encryption**: One key is used to encrypt the data, the other is used to decrypt the data (slower), used for transmitting data.
Securing Internet Transactions

Many public and private database applications are developed for use over the web.

All transactions are vulnerable to eavesdropping attacks.

Secure Sockets Layer (SSL) requires a digital security certificate, and uses a public-key infrastructure (PKI).

Web browser encrypts a message with the public key, and the server decrypts it with its private key – the certificate ensures that the key combination you are using is authentic.
Public Key Encryption

Client Browser

https://server

Server's public key
Data encrypted with server's public key

Data decrypted with server's private key

Public key
Private key

Certificate Authority

Security certificate

Server

Database

Data
Encrypting Selected Data in the Database

Sensitive information needs to be encrypted, in case the data is stolen

Use the dbms_crypto package to encrypt strings in the database

See /space/ordata/classfiles/Chapter12/SQL/Encrypt.sql

But, the keys have to be kept secure... how?
Protecting Source Code with Wrap

When you want to protect your stored procedures from “prying eyes”, obfuscate it with wrap

It is a one-way conversion…

Syntax (from the command line):

```
wrap iname=<input file> [oname=<output file>]
```
Auditing the Database

• Can identify undesirable actions and track which people performed those actions
• Can serve as a deterrent, if known
• Can provide a trail for investigators and legal action
• Can provide detailed data on how the DB is used – useful for tuning the database
• Can help in redesigning security permissions by detecting problems
Auditing Features

• Auditing records the name of the operation, the object affected, the user performing the operation, and the date and time of the action.

• Audit records can be stored in the database, or written to a file (but the file output has less information)

• Can be used to audit successful and/or unsuccessful operations

• Can even audit the actions of DBAs
## Oracle auditing types

<table>
<thead>
<tr>
<th>Type of Auditing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statement Auditing</strong></td>
<td>Tracks activity by the type of statement. For example, AUDIT TABLE tracks several DDL statements regardless of the table involved. A broad-based tracking but it can be applied to individual users or to everyone.</td>
</tr>
<tr>
<td><strong>Privilege Auditing</strong></td>
<td>Monitors the use of system privileges such as AUDIT CREATE TABLE. Can be assigned to track individual users or everyone.</td>
</tr>
<tr>
<td><strong>Schema Object Auditing</strong></td>
<td>Records the usage of specific statements on individual objects. For instance, AUDIT SELECT ON customers records a note every time someone retrieves data from the customer table. Always applies to all users.</td>
</tr>
<tr>
<td><strong>Fine-Grained Auditing</strong></td>
<td>The most detailed specifications, you can create any Boolean condition based on individual columns or specific rows of data. Or example, you might track all orders with a value &gt; 100,000.</td>
</tr>
</tbody>
</table>
Enabling Auditing

-- as sysdba:
ALTER SYSTEM set audit_trail=DB_EXTENDED
    SCOPE=SPFILE;

-- Restart the database
-- confirm auditing is on
Show parameter audit_trail;

-- enable auditing for an object
AUDIT <SQL statement> [ON <schema object>] [BY <user>] [WHENEVER [NOT] SUCCESSFUL]
Auditing Examples

AUDIT INSERT ON Agents;
AUDIT DELETE ON Agents;

INSERT INTO Agents (AgentID, FirstName, LastName) VALUES (99, 'Barbara', 'Feldon');
DELETE FROM Agents WHERE AgentID=99;

NOAUDIT INSERT ON Agents;
NOAUDIT DELETE ON Agents;
Viewing Audit Trails

```sql
SQL> SET linesize 150
SQL> COLUMN NTIMESTAMP# FORMAT DATETIME
ERROR 1065 (01000): Illegal FORMAT string "DATETIME"
SQL> COLUMN UserID FORMAT A15
SQL> COLUMN OBJ$NAME FORMAT A20
SQL> COLUMN SQLTEXT FORMAT A40
SQL> SELECT TO_CHAR(NTimestamp#, 'YYYY-MM-DD HH24:MI:SS') AS Time, UserID, OBJ$NAME, SQLTEXT FROM sys.aud$;

<table>
<thead>
<tr>
<th>TIME</th>
<th>USERID</th>
<th>OBJ$NAME</th>
<th>SQLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-02-10 23:02:12 REDWOOD</td>
<td>AGENTS</td>
<td>INSERT INTO Agents(AgentID, Firstname, Lastname) VALUES (99, 'Barbara', 'Feldon')</td>
<td></td>
</tr>
<tr>
<td>2005-02-10 23:02:40 REDWOOD</td>
<td>AGENTS</td>
<td>DELETE FROM Agents WHERE AgentID=99</td>
<td></td>
</tr>
</tbody>
</table>

SQL>
```
Disabling Auditing

-- To turn off auditing:
-- as sysdba
ALTER SYSTEM set audit_trail=NONE
  SCOPE=SPFILE;
-- delete the audit rows from the database
DELETE FROM sys.aud$;
Creating Triggers for Auditing

Same as Chapter 4 example

See /space/ordata/classfiles/Chapter12/SQL/AuditTrigger.sql
Fine-Grained Auditing

Similar to trigger code, but far less effort and more controls
Handled through the DBMS_FGA package
Records are stored in the DBA_FGA_AUDIT_TRAIL view
Example: record changes to the employee table done by people who are not in the HR department
FGA Example

DBMS_FGA.ADD_POLICY (  
Object_schema => 'HRM',  
Object_name => 'Employee',  
Policy_name => 'Monitor_Employee_Table',  
Audit_condition => 'Department' <> "HRM",  
Audit_column => 'Salary',  
Statement_types => 'select, insert, update, delete');