Date _____ Day

CHAPTER # 05

- **SQL INJECTION ATTACK**

→ SQLi is an attack that exploits security vulnerability in the database layer.

→ Attackers extract or manipulate 's data

→ Steps involved in SQLi Attack

1. Hackers finds a vulnerability in a custom web app & injects an SQL cmd to a db. The cmd is injected into traffic that will be accepted by the firewall.
2. The webserver recvs the malicious code & sends it to the web app server
3. The web app server ~~recvs the~~ sends it to DB server
4. The DB server executes the malicious code on DB.
5. The DB returns data from ~~credit~~ table.

→ Injection Technique

```
var ShipCity;
ShipCity = request.get ("ShipCity");
var sql = " select * from OrdersTable where
          ShipCity = ' " + ShipCity + " ' ";
```

Date _____ Day _____

If (ShipCity = Be

If user enters "Karachi" then query becms

```
Select * from OrdersTabe where
ShipCity = 'Karachi'
```

But if user entered → 'Karachi' ; DROP table OrdersTable -- → added this to indicate that lines after this needs to be commented

→ Now the query becomes

```
Select * from OrdersTable where ShipCity
= 'Karachi' ; DROP table OrdersTable --
```

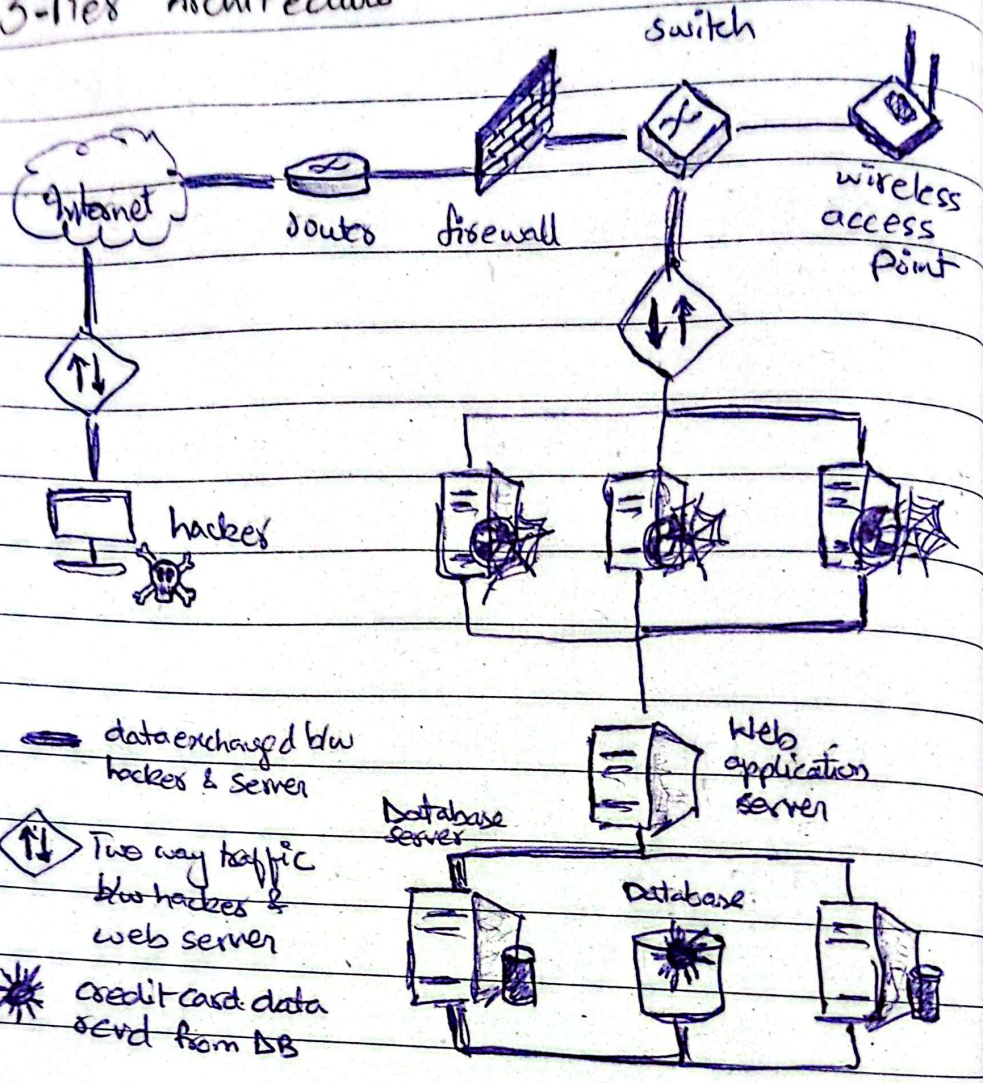
; → Seperates two commands

DROP table OrdersTable -- → This will be executed seperately

→

Date _____ Day _____

▷ 3-Tier Architecture



▲ Typical SQL Injection Attack

▷ Main SQLi Attack Avenues

① User Inputs

→ Attacker supplies malicious SQL through form field URL parameters (GET), or POST bodies

Date _____ Day _____

↦ Application directly concatenates input into SQL queries without doing any validation on input

→ Defenses:
- ↳ Use parameterized queries
- ↳ Use strong input validation
- ↳ Use ORMs that avoid raw SQL

② Server Variables (HTTP / Network Headers)

→ Attacker injects SQL via forged HTTP headers or ~~other env vars that the app logs~~

→ Why it works: Headers are under attacker control and often assumed trusted by server-side code

→ Defenses:
- ↳ Sanitize & treat server variables as untrusted input

③ Second-Order Injection

→ Malicious data is stored safely (partially sanitized) but later used unsafely in a different context, causing injection when the data is re-used in a new query

→ Inputs may pass initial checks; later code assumes stored data is safe.

→ Defenses:
- ↳ Always parameterize every query (even if data comes from DB)

Date _____ Day _____

④ Cookies

→ Client-controlled cookies are modified to contain SQL payloads, the server builds SQL queries using cookie value

→ Example: `auth = admin'--` stored in cookie & used directly in SELECT ... WHERE user = ' + cookie.

→ Defenses:

↳ Treat cookies as untrusted input

③ Physical User Input / non-web inputs

→ Any external input channel (barcodes, QR codes) can carry SQL payloads that are later stored or used in queries

→ Eg: A csv doc scanned which contains a row having value "; DROP table Orders; --

→ Applications often assumed these data to be safe therefore fails to validate

→ Defenses:

↳ Sanitize & validate every external data source.

Day _____

▷ SQLi Attack Types

① Inband (uses same channel for injection & results)

↳ Attackers inject SQL and see results directly in the application response

↳ It has 3 types

i- Tautology

↳ Inject SQL that makes a WHERE condition always TRUE

↳ Ex: name = '' OR 1=1 -- → return all rows

↳ Bypass login checks or retrieve full tables

ii- End of Line Comment

↳ Use comment tokens (--, #, /* ... */) to neutralize the rest of the query

↳ Often combined with tautology to ignore pswd checks

iii- Piggybacked queries

↳ Append extra queries to the intended query

↳ Ex: '; DROP table users ; --

② Inferential (blind) — attacker learns data by observing behavior

↳ No data is directly returned ; attacker infers result by how the application behaves

Day

Date

↳ Following are the types incorrect
i- Error-based (illegal/logically queries)
↳ Trigger DB errors to learn schema or types (eg: causing a type mismatch or syntax error)
↳ Rely on verbose error messages
ii- Blind SQLi
↳ Boolean-based: Injects queries that evaluate to TRUE/False; Attacker observes page differences
Ex: AND (SUBSTR(password,1,1) = 'a') → if page shows normal content → 1st charcter is "a".
↳ Time-based: Use time delays (SLEEP()) to infer truth by response latency.
Ex: IF (condition, SLEEP(5), 0) → 5s delay → condition is True.

③ Out-of-band (OOB)
↳ It is a type of SQLi where the attacker doesnot rcv a response from the attacked application on the same communication channel but instead it is able to cause the applicati to send data to a remote endpoint that they control.

Date _____ Day _____

↳ It is only possible if the server that you are using has cmds that trigger DNS or HTTPS request.

▷ SQLi Countermeasure

① Use parameterized queries / prepared statements (never concatenate user input into SQL)
② Use ORM or safe DB libraries that seperate code from data
③ Validate & normalize input
④ Avoid verbose DB error messages to users ; log them privately
⑤ Use Web Application Firewall (WAF) and monitoring for suspicious patterns
⑥ Regularly test with safe / authorized security testing

▷ SQLi Detection

① Signature Based
- Looks for known attack patterns
- Fast & accurate for known attacks
- Needs constant update & misses new/changed attacks

② Anomaly - based
- Learns what "normal" looks like, than flags anything unusual.

Date Day

- can catch new or unknown attacks

③ Code Analysis
- Runs many automated test attacks against the app to find SQLi holes
- Find vulnerabilities before attackers do
- Needs good test coverage & can miss logic bugs.

- DATABASE ACCESS CONTROL

→ Database access control means managing who can access what in a database
→ The system first authenticates users
→ Then it controls what each user can do — like read, insert, create, update, delete
→ Access can be given at diff levels — to whole DB, to a table or even specific rows/col.
→ Three main types of administration:
① Centralized — Only a few admins can give or remove access
② Ownership-based — The person who creates a table can give/remove access to others
③ Decentralized — The owner of the table may grant/revoke authorization rights to other users.

Day

- SQL Based Access Definition

→ SQL provides GRANT & REVOKE to assign & remove access rights on DB objects

GRANT <privileges> → SELECT, INSERT, UPDATE ...
ON <object> → DB, table, schema ...
TO <user | role | PUBLIC>
[WITH GRANT OPTION]

↳ allows the grantee to re-grant that privilege to others

Ex: GRANT SELECT ON Employees TO Samroze
GRANT SELECT (salary) ON Employees TO payroll_role
GRANT accountant_role TO Samroze
REVOK SELECT ON payroll FROM Samroze

→ The optional IDENTIFIED BY clause is used to specify pswd that must be used when creating users/roles

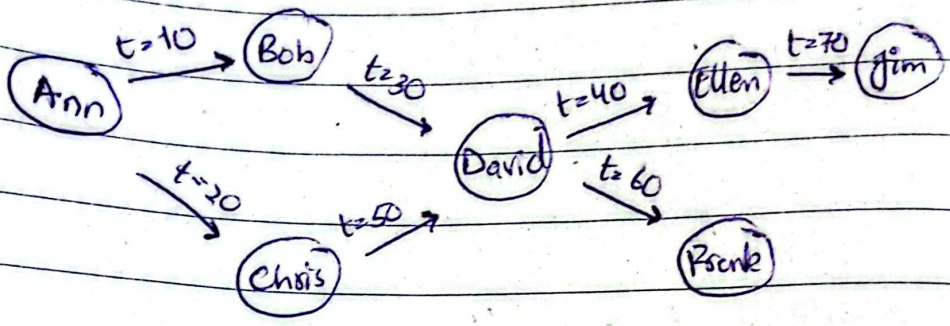
- Cascading Authorization

↳ It happens when a user who has an access right with the GRANT OPTION passes that right to others and those others pass it further down.

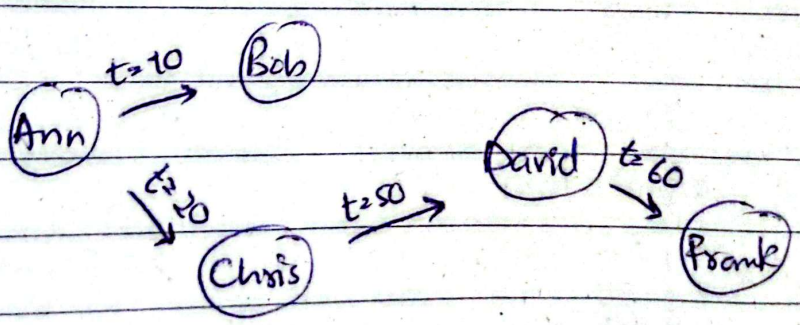
↳ So permission spread (cascade) from one user to many.

Date _____ Day _____

→ Same is for revokation of rights.
"When a user's privilege is revoked, all the rights that originated from that user's grant are also revoked."



→ Ab Agr Bob, David se access rights leta hai to Ellen or Jim dono ke rights bhi revoke hojayenge q k Ellen or Jim ko jo David se rights mile hai wo Bob wale hai. Jabke Frank k isliye revoke nhi honge q k uske wale rights Chris k hai.



Date _____ Day _____

Role Based Access Control

1. Application Owners
   - Owns database tables & data for a specific app
2. End-user (Non-owners)
   - Uses the app to access data but doesn't own it
3. Administrator
   - Manages the database system, roles and security

→ RBAC in a Database must support
- Create / Delete role
- Give permission to roles
- Assign or remove users from roles

→ RBAC makes DB security organized, scalable, and easier to manage by grouping permissions into roles instead of assigning them one by one to users.

Date _____ Day _____

INFERENCE

→ A user who has permission to see some data can figure out (infer) other data they are not authorized to see — just by combining or analyzing the allowed data.

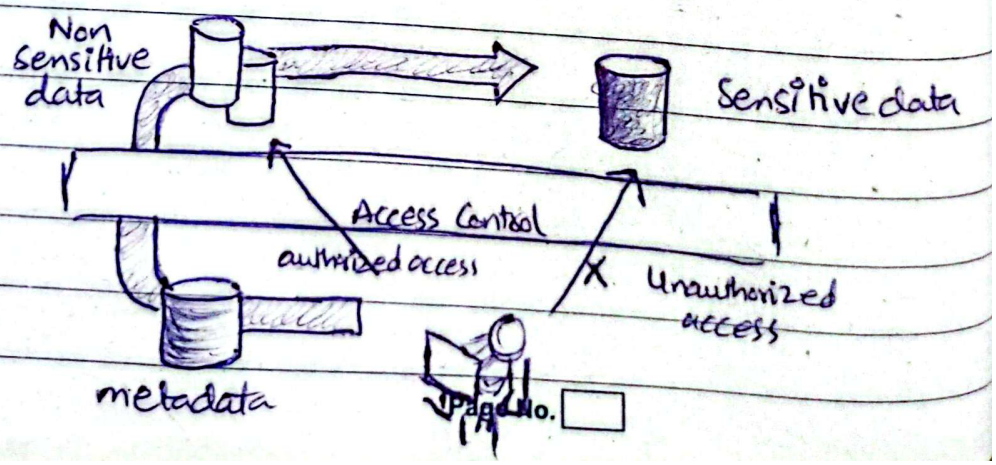
→ Combining non-sensitive data can reveal sensitive info

→ Or data relationships (metadata) can help guess hidden facts

- Prevention Technique

① During Database Design
  - Split tables (eg: seperate sensitive col)
  - Use stricter access rules (eg: RBAC roles)

② At Query Time
  - Detect when a user's query might reveal forbidden info & block or modify it



Day

• DATABASE ENCRYPTION

→ Since DB contains an orgs most valuable data, it needs strong protection — firewalls, authentication, access control & finally encryption.

→ Disadvantages of DB Encryption

① Key Management
- ↳ Every authorized user needs a decryption key
- ↳ Managing who gets which key is complex, especially with many users & apps

② Inflexibility
- ↳ Searching or filtering records becomes harder when data is encrypted
- ↳ You might have to decrypt everything before searching

→ Levels of Encryption
- ▷ Database level → Whole DB encrypted
- ▷ Record level → Specific rows
- ▷ Attribute level → Certain cols (like pswd)
- ▷ Field level → individual cells

Date Day

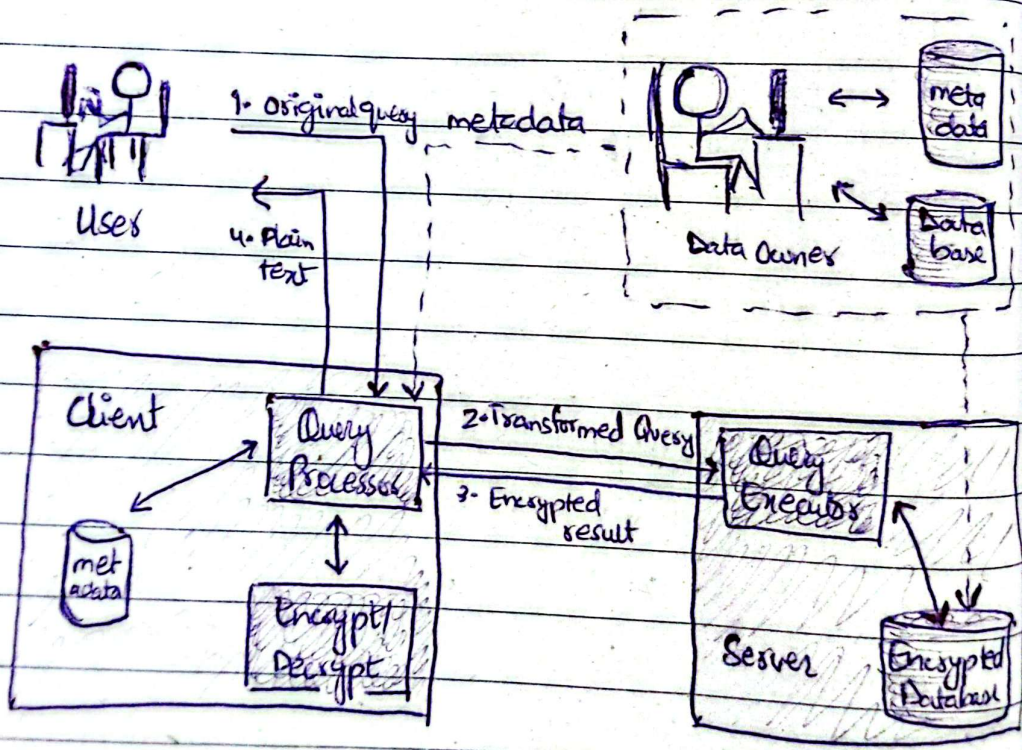
→ Four Entities are involved

① Data Owner
↳ Creates and encrypts data before sending it to server

② Server
↳ Stores encrypted data (cloud provider)

③ Client (Frontend)
↳ Translates user queries → encrypted queries

④ User
Sends request & views decrypted results



▲ Database Encryption Scheme

Date _____ Day _____

→ Following steps are followed for info retrieval

① Uses makes a query (eg: "get details of employee with ID = 123")

② The client system (where the user's query is processed) encrypts that ID (123) using the secret key "k" into "11011101" befor sending it to server

③ The server, which only has the encrypted data, uses that encrypted ID to find the correct record and sends it back

④ The client then decrypts the data using its key & shows the readable result to user.

→ The server never sees any ~~user~~ unencrypted data or key, so even if it gets hack, the attacker only sees encrypted data.

→ This method can be slow & limited.