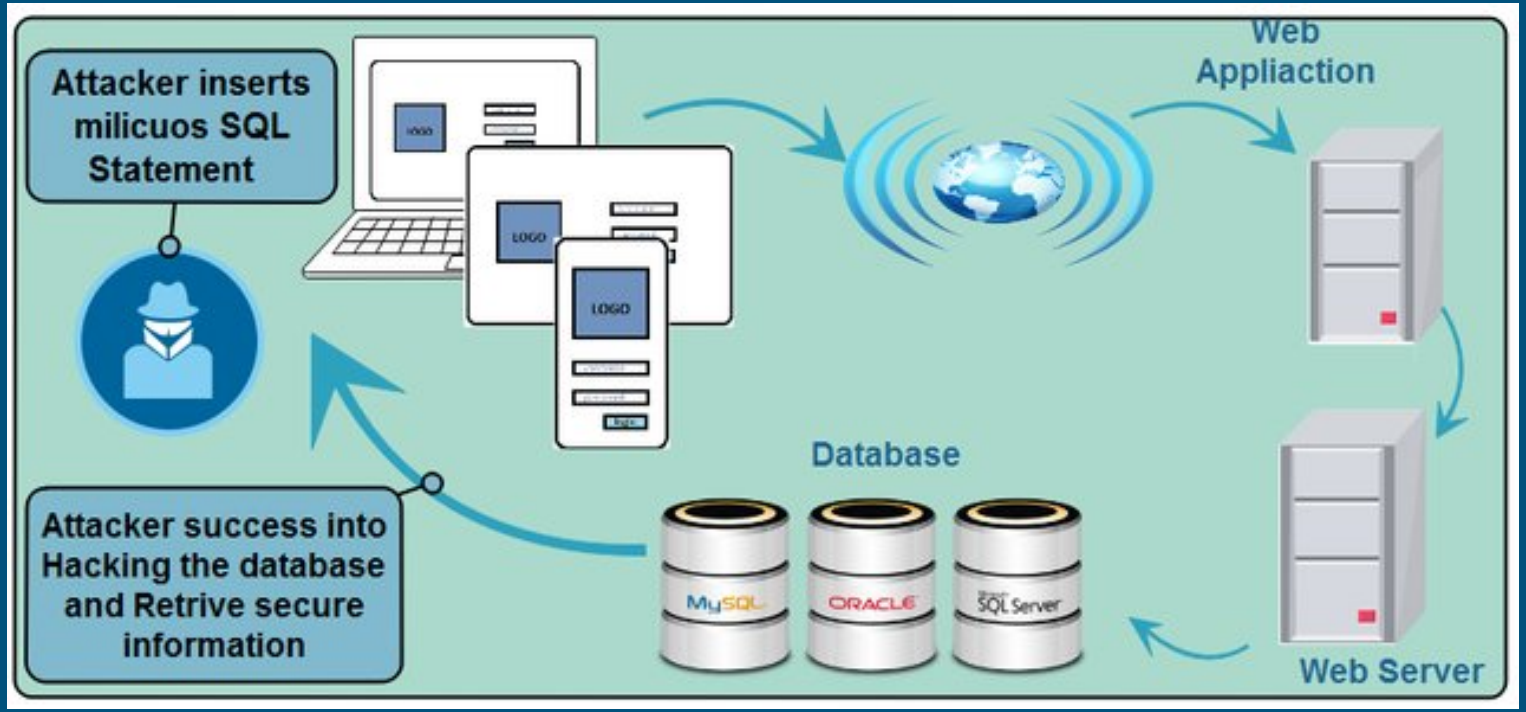

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

SQL InjectionAttacks

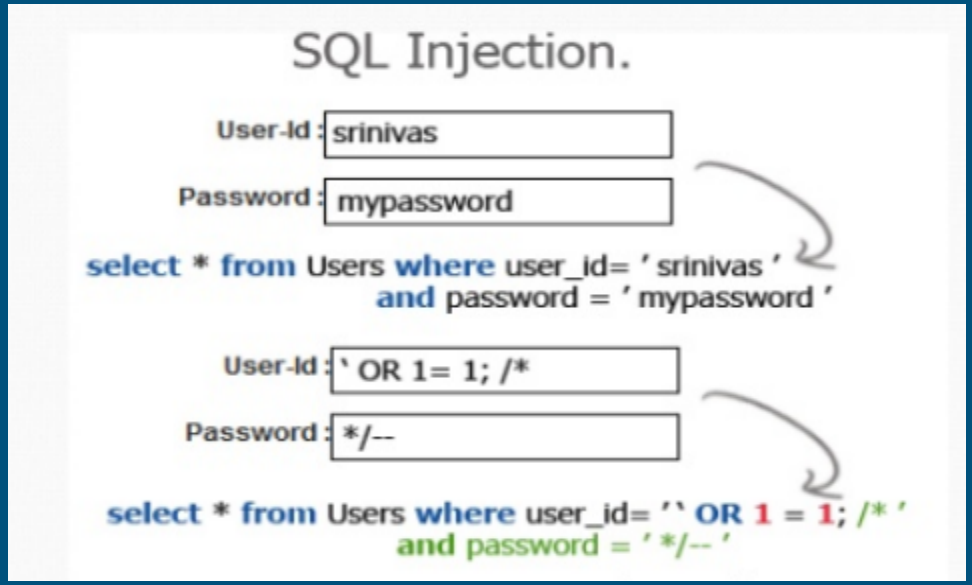

Fig 1: Example of SQL injection attacks

SQL Injection Mechanism

- Injection through user input
- Injection through cookies
- Injection through server variables
- Second order Injection

SQL Injection Attacks Intent

1. Determining database schema
2. Extracting data
3. Adding or modifing data
4. Identifying injectable parameters
5. Bypassing authentication

SQL Injection Attacks Types

- Tautologies
- Logical incorrect query
- Union query
- Piggy-backed queries
- Blind injection
- Timing attacks
- Alternet encoding

SQL Injection Attacks: Tautologies

- Injects SQL tokens to the conditional query statement to be evaluated always true

"SELECT * FROM employee WHERE userid = '112'
and password ='aaa' OR '1'='1'"

SQL Injection Attacks: Logical Incorrect Query

- Injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors.
- In this type of injection an attacker tries to gather information about the type and structure of the back-end database of a Web application.
- The attack is considered to be a preliminary step for further attacks.
- If an incorrect query is sent to a database, some application servers return the default error message.
- Through type error, one can identify the data types of certain columns

SQL Injection Attacks: Union Query

- Join injected query to the safe query by the word UNION and get data about other tables from the application

"SELECT Name, Phone FROM Users WHERE Id=$id"

By injecting the following Id value:

$id= 1 UNION ALL SELECT credit Card Number, 1 FROM Credit CarTable

SQL Injection Attacks: Piggy-backed Queries

- Normally the first query is legitimate query, whereas following queries could be illegitimate

"SELECT info FROM users WHERE login='doe' AND pin=0; drop table users"

SQL Injection Attacks: Blind Injection

- Attacker can steal data by asking a series of True/False questions through SQL statements instead of an error message.

“SELECT accounts FROM users WHERE login= 'doe' and 1 =0 -- AND pass = AND pin=0”

“SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=0”

SQL Injection Attacks: Timing Attacks

- Attacker gather information from a database by observing timing delays in the database's responses. This technique uses an if-then statement for injecting queries.

“declare as varchar(8000) select as = db_name() if (ascii(substring(as, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:5'”

SQL Injection Attacks: Alternet encoding

- Attackers modify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode.

“SELECT accounts FROM users WHERE login='' AND pin=0; exec (char(Ox736875746 46j776e))”

Second order injection

- Second-order SQL injection arises when user-supplied data is stored by the application and later incorporated into SQL queries in an unsafe way.
- To detect the vulnerability, it is normally necessary to submit suitable data in one location, and then use some other application function that processes the data in an

- SQL injection attacks that delay execution until a secondary query are known as "second order".
- If we create an account with same username of an existing one and a suffix -, it comments out password portion.
- Example:
- https://bertwagner.com/posts/how-to-steal-data-using-a-second-order-sql-injection-attack/
- https://haiderm.com/second-order-sql-injection-explained-with-example/
- Solution: parameterized query and input sanitization

Second Order Injection example

```
1. UPDATE users
2. SET password='123'
3. WHERE username='haider'--' and password='abc'
```

Now as the username in WHERE clause is **“haider’ –“**, Not that After — the query is discarded as comments, as — is used to start comments in SQL . The query ends up like:-

```
1. UPDATE users
2. SET password='123'
3. WHERE username='haider'
```

Input validation

- Simple input check can prevent many attacks.
- Always validate user input by checking type, size, length, format, and range.
- Test the content of string variables and accept only expected values.
- Reject entries that contain binary data, escape sequences etc.
- When you are working with XML documents, validate all data against its schema as it is entered.

Parameterized query

- The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access.
- This method uses two steps to incorporate potentially tainted data into SQL queries:
    - first, the application specifies the structure of the query, leaving placeholders for each item of user input;
    - second, the application specifies

- Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure.

Parameterized query exmple

```
String custname = request.getParameter("customerName");
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Evading Signature Based Detection

Evading ' OR 1=1 signature

- ' OR 'unusual' = 'unusual'
- ' OR 'something' = 'some'+'thing'
- ' OR 'text' = N'text'
- ' OR 'something' like 'some%'
- ' OR 2 > 1
- ' OR 'text' > 't'
- ' OR 'whatever' IN ('whatever')
- ' OR 2 BETWEEN 1 AND 3

SQLIA Detection or Prevention tools: CANDID

- Dynamically mines the programmer-intended query structure on any input
- Detects attacks by comparing coming input against the structure of the actual query issued.

Detection or Prevention tools: SQL Guard

- Checked at runtime which is expressed as grammar that only accepts legal queries.
- SQL Guard examines the structure of the query before and after the addition of user-input based on the model.

Detection or Prevention tools: SQL Check

- Checked at runtime which is expressed as grammar that only accepts legal queries.
- SQL Check the model is specified independently by the developer
- Use a secret key to delimit user input during parsing by the runtime checker.

Detection or Prevention tools: SQL_IDS

- Focus on writing specifications for the web application
- Describe the intended structure of SQL statements are produced by the application,
- Automatically monitoring the execution of these SQL statements for violations with respect to these specifications