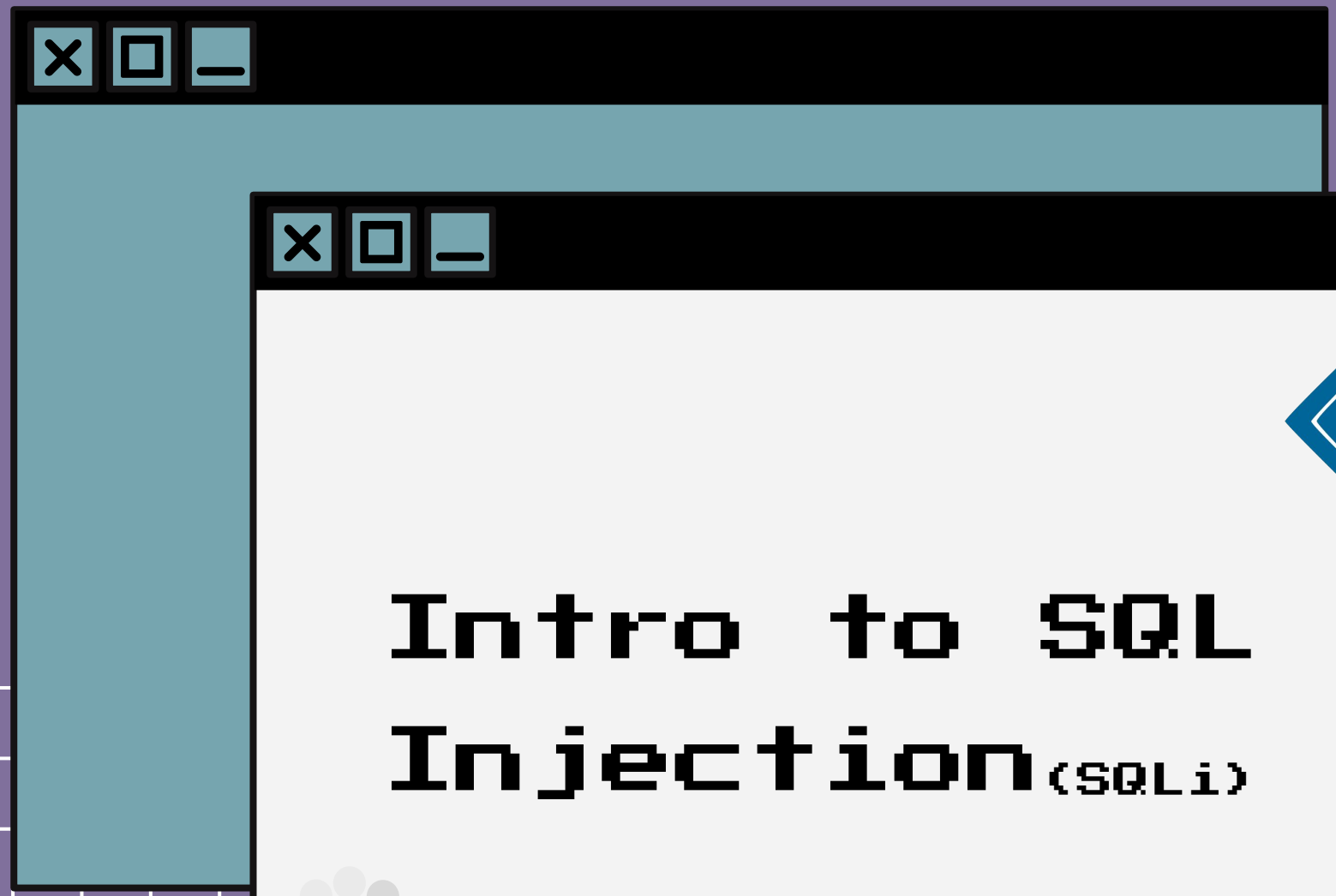
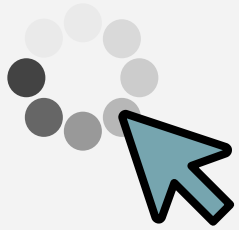
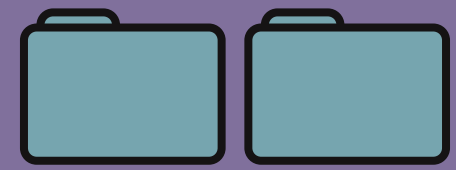


Intro to SQL Injection (SQLi)



by orestis karapiperis



What is SQL?

What is SQL?

SQL (Structured Query Language) is a programming language used to manage and interact with relational databases. It allows you to store, retrieve, and manipulate data in a structured way.

Key Functions of SQL:

- Data Querying: Fetch specifc information using commands like **SELECT**.
- Data Manipulation: Modify existing data with **INSERT**, **UPDATE**, and **DELETE**.
- Data Defnition: Defne database structure with commands like **CREATE TABLE**.

Basic SQL Concepts

1. Database
A collection of organized data that can be easily accessed, managed, and updated.

2. Table
The fundamental structure within a database. It consists of rows (records) and columns (fields).

Table: Users

ID	Username	Password	Email	Role
1	alice	alice123	alice@example.com	Admin
2	bob	securePass	bob@example.com	User
3	charlie	passCharlie	charlie@sample.net	Moderator

Basic SQL Concepts

3. Query:
A request to access or modify data in the database.

4. Primary Key:
A unique identifier for each record in a table.

5. Common Commands:
- SELECT : Retrieve data from a table
- INSERT : Add new records
- UPDATE : Modify existing records
- DELETE : Remove records
- UNION : Combines multiple queries
- ORDER BY: Sorts the result set of a query based on one or more columns.

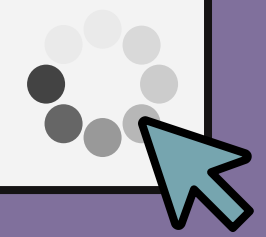
SQL Query Structure

```
SELECT column1, column2

FROM table_name        - - this is a comment

WHERE condition;
```

Query example:

```
SELECT * FROM Users WHERE Role = 'Admin';
```

ID	Username	Password	Email	Role
1	alice	alice123	alice@example.com	Admin



What is SQL Injection?

What is SQL Injection?

- Web attack category
- SQLi is a type of cyberattack where an attacker exploits vulnerabilities in a web application's input fields to inject malicious SQL code.
- This can manipulate the database's queries to extract, modify, or delete data, and even take control of the database server.

Why It Happens:

SQL injection occurs due to improper handling of user input. Specifically:

1. **Unsanitized Input:**
   User input is directly incorporated into SQL queries without proper filtering or validation.

2. **Dynamic SQL Queries:**
   Queries that concatenate user input directly into the SQL statement.

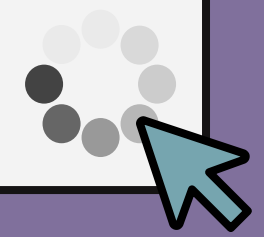
Why It Happens:

3. Lack of Parameterization:
The absence of parameterized queries or prepared statements leaves the system vulnerable.

4. Poor Error Handling:
Detailed error messages may reveal sensitive information about the database structure, aiding attackers.

Is this safe?

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



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

If an attacker enters:

username : admin'--

The query becomes:

```
SELECT * FROM users WHERE username = 'admin'--' AND password = '$password';
```

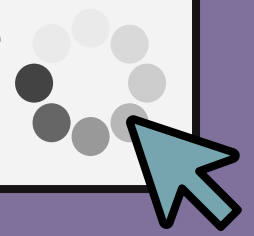
Thus, bypassing password check

Types of Vulnerabilities

1. **Input Fields**: Login forms, search boxes, or contact forms.

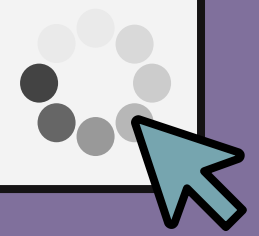
2. **URL Parameters:** Data passed through URLs (e.g., ?id=5).

3. **Cookies:** Maliciously crafted cookies can manipulate server-side SQL queries.



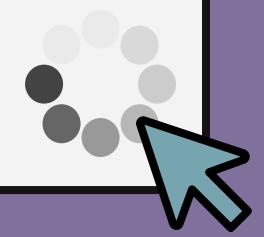
Basic Exploit Steps:

1. **Find Input Fields:** Login forms, search bars
2. **Test for Vulnerabilities:** Inject `' OR '1'='1`
3. **Extract Data:** Use crafted payloads



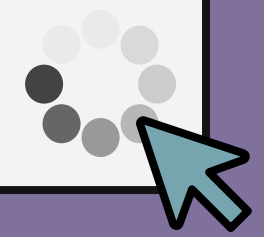
Why `' OR '1'='1` ?

1. **' (Single Quote):** Closes the username field input.
2. **OR:** SQL logical operator that evaluates if at least one condition is true.
3. **'1'='1:** A tautology (always true) because 1 is equal to 1.



Why `' OR '1'='1` ?

Basically, when an `OR` clause contains a condition that's always true (like `1=1`), it effectively makes the `WHERE` filter irrelevant for all rows in the table, since the entire `WHERE` clause evaluates to `TRUE`.

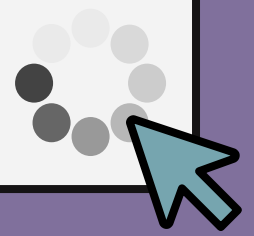


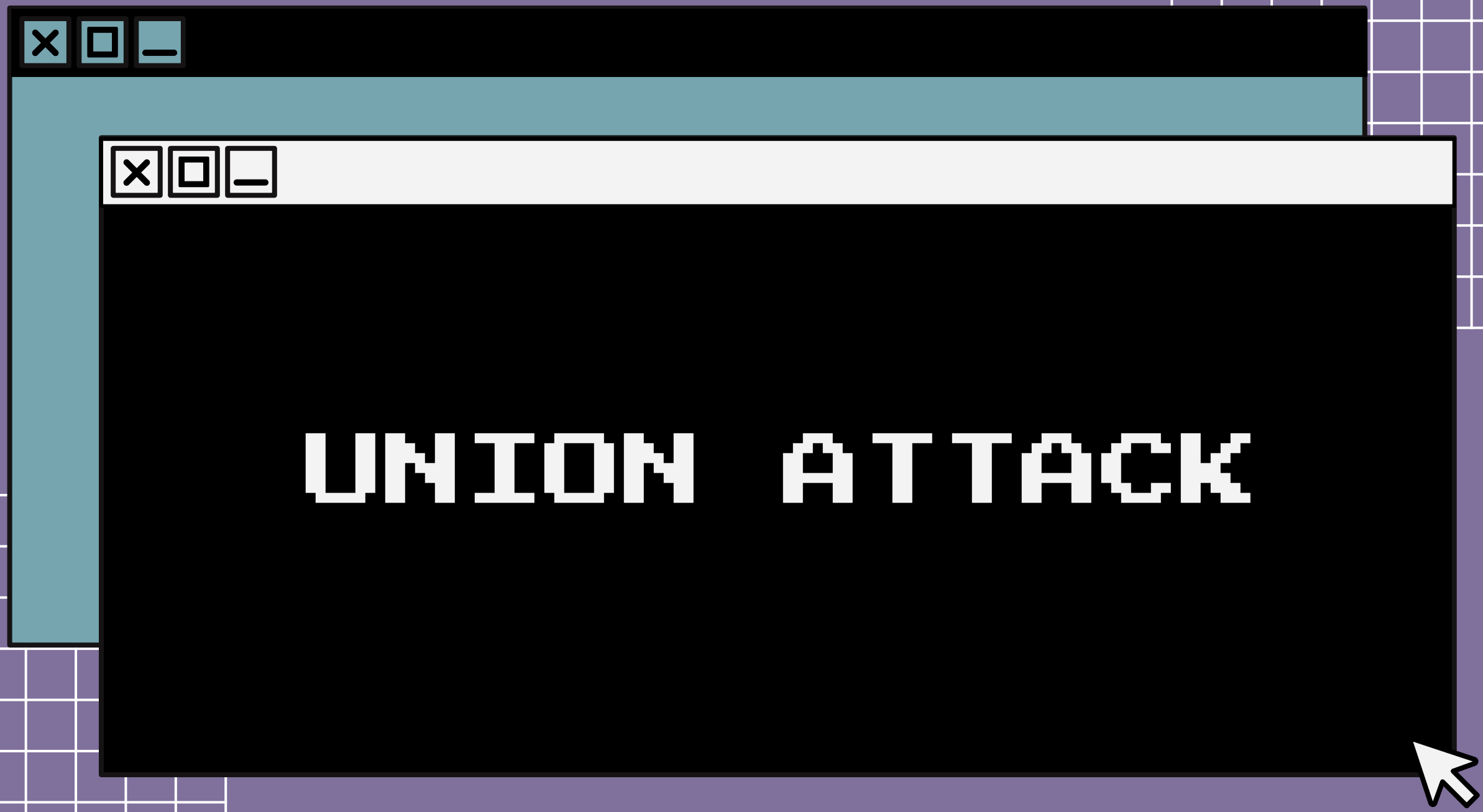
Examples

Portswigger labs:

1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

2. SQL injection vulnerability allowing login bypass




UNION ATTACK

UNION-based SQL Injection

- Leverages the `UNION` SQL operator to combine results from multiple queries into a single response.
- How it works:
  The attacker injects a `UNION SELECT` statement to extract data from a different table or columns.
- Requirements:
  - **Same Number of Columns:** Both queries must return the same number of columns.
  - **Compatible Data Types:** Columns must have compatible data types.

UNION injection in practice

- Example:
  textbox: id
  query: SELECT id, username, email FROM users WHERE id = '$id';

  Task: Expose the id's and passwords of admin users

  *Tip: The admin info is stored in **admin_users** table*

UNION injection in practice

query: SELECT id, username, email FROM users WHERE id = '$id';
Task: Expose the id's and passwords of admin users

- We see that the first query returns 3 columns. So we must ask for 3 columns too
- The query must be : SELECT id, password, NULL FROM admin_users

- So, we could enter this in the textbox, adding ' UNION:
  ' UNION SELECT id, password, NULL FROM admin_users'

  making the injected query as:

  SELECT id, username, email FROM users WHERE id = '' UNION SELECT id, password, NULL FROM admin_users'

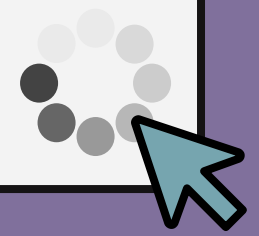
General Steps for UNION-Based Attacks

1. **Find the Number of Columns:** Use ORDER BY or UNION SELECT NULL, NULL,... to determine the column count.
2. **Identify Data Types:**
   Inject payloads with known data types to match the original query's structure.

General Steps for UNION-Based Attacks

3. Extract Data:

Replace each NULL in UNION SELECT with sensitive column names (e.g., passwords, emails).



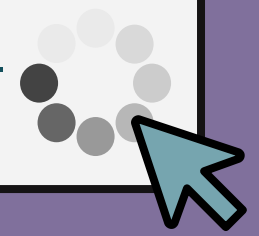
Portswigger Example

Lab: SQL injection attack, querying the database type and version on Oracle

https://portswigger.net/web-security/sql-injection/examining-the-database/lab-querying-database-version-oracle

SQLi Cheatsheet:

https://portswigger.net/web-security/sql-injection/cheat-sheet



BLIND SQLi

Can SQL be blind?

Blind SQL Injection is used when an application does not show query results directly but provides clues through behavior or responses (e.g., time delays, error messages).

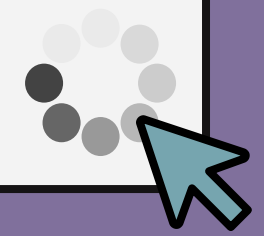
Types of Blind SQLi:

- Boolean-Based (Content-Based):
  True/false conditions trigger different responses.
  Example:
  ' OR 1=1 -- (Always true)
  ' AND 1=0 -- (Always false)

- Time-Based:
  Uses delays to infer if conditions are true.
  Example: ' OR IF(1=1, SLEEP(5), 0); --

Consequences of SQLi

1. **Data Breaches:** Access to sensitive data.
2. **Data Manipulation:** Alter or delete records.
3. **System Compromise:** Advanced attacks can take control of the server.



PREVENTING SQLi

How to prevent SQL Injection:

- Input Validation: Sanitize input data.
- Parameterized Queries: Use placeholders for variables.
- Use ORM: Safe interaction with the database using object-oriented code.
- Web Application Firewall (WAF): Filters malicious requests.

Input Validation (Sanitize Input Data):

Ensure that input data is clean and conforms to expected formats (e.g., email, username, etc.). Reject or sanitize any input that doesn't match the expected pattern.

Example: For instance, if expecting a numeric ID in the query, validate the input to ensure it is a number.

```php
$id = $_GET['id'];
// Validate that $id is numeric
if (is_numeric($id)) {
    $query = "SELECT * FROM users WHERE id = $id";
    // Execute the query...
} else {
    // Handle invalid input (e.g., error message)
    echo "Invalid input!";
}
```

Parameterized Queries (Prepared Statements)

Always use parameterized queries (or prepared statements) instead of directly embedding user input into SQL queries. This way, input is treated as data, not executable code.

Example: Using parameterized queries ensures user input is not directly executed as part of the SQL statement.

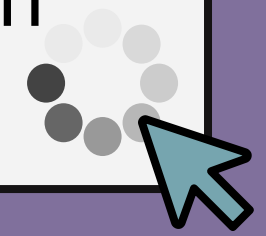
```php
// Establish DB connection...
$stmt = $mysqli->prepare("SELECT * FROM users WHERE username = ?");
$stmt->bind_param("s", $_GET['username']);  // "s" means the input is a string
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    echo $row['username'];
}
```

Practice time

https://play.picoctf.org/

Classroom Code: **CleCzOnr1** (its an omikron)

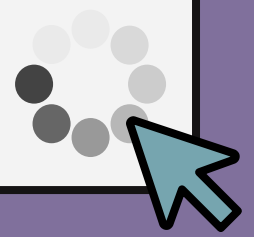
Practice -> Assignments -> SQL Injection



Practice time

Start with:

1. SQLiLite
2. More SQLi (hint: google sqlite_master)
3. SQL Direct (practice at home, PostgreSQL client needed: windows, debian: sudo apt-get install postgresql-client-common, then connect using given command)



Thank you for your attention!

IEEE

UNIVERSITY OF PIRAEUS