Page 11 / Union Clause

Union Clause

So far, we have only been manipulating the original query to subvert the web application logic and bypass authentication, using the OR operator and comments. However, another type of SQL injection is injecting entire SQL queries executed along with the original query. This section will demonstrate this by using the MySQL Union clause to do SQL Union Injection.

Union

results from multiple SELECT statements. This means that through a UNION injection, we will be able to SELECT and dump data from all across



Next, let us see the output of the ships tables:

```
mysql> SELECT * FROM ships;
1 rows in set (0.00 sec)
```

```
• • •
                                                                                               Union Clause
  | CN SHA | Shanghai
| SG SIN | Singapore
| Morrison | New York
| ZZ-21 | Shenzhen
  4 rows in set (0.00 sec)
```

As we can see, UNION combined the output of both SELECT statements into one, so entries from the ports table and the ships table were combined into a single output with four rows. As we can see, some of the rows belong to the ports table while others belong to the ships table.

Note: The data types of the selected columns on all positions should be the same.

Even Columns

A UNION statement can only operate on SELECT statements with an equal number of columns. For example, if we attempt to UNION two queries that have results with a different number of columns, we get the following error:

```
mysql> SELECT city FROM ports UNION SELECT * FROM ships;
```

return the same number of columns, we can use the UNION operator to extract data from other tables and databases.



| Table of Contents | |
|---------------------------------|--|
| Introduction | |
| Databases | |
| Intro to Databases | |
| Types of Databases | |
| MySQL | |
| ☞ Intro to MySQL | |
| ☺ SQL Statements | |
| ☺ Query Results | |
| ☺ SQL Operators | |
| SQL Injections | |
| Intro to SQL Injections | |
| ☺ Subverting Query Logic | |
| Using Comments | |
| 19 Union Clause | |
| Union Injection | |
| Exploitation | |
| Database Enumeration | |
| © Reading Files | |
| ☞ Writing Files | |
| Mitigations | |
| Mitigating SQL Injection | |



Skills Assessment - SQL Injection Fundamentals

Closing it Out

Un-even Columns

We will find out that the original query will usually not have the same number of columns as the SQL query we want to execute, so we will have to work around that. For example, suppose we only had one column. In that case, we want to SELECT, we can put junk data for the remaining required columns so that the total number of columns we are UNIONing with remains the same as the original query.

For example, we can use any string as our junk data, and the query will return the string as its output for that column. If we UNTON with the string "junk", the SELECT query would be SELECT "junk" from passwords, which will always return junk. We can also use numbers. For example, the query SELECT 1 from passwords will always return 1 as the output.

Note: When filling other columns with junk data, we must ensure that the data type matches the columns data type, otherwise the query will return an error. For the sake of simplicity, we will use numbers as our junk data, which will also become handy for tracking our payloads positions, as we will discuss later.

Tip: For advanced SQL injection, we may want to simply use 'NULL' to fill other columns, as 'NULL' fits all data types.

The products table has two columns in the above example, so we have to UNION with two columns. If we only wanted to get one column 'e.g. username', we have to do username, 2, such that we have the same number of columns:

```
Code: sql

SELECT * from products where product_id = '1' UNION SELECT username, 2 from passwords
```

If we had more columns in the table of the original query, we have to add more numbers to create the remaining required columns. For example, if the original query used SELECT on a table with four columns, our UNION injection would be:

```
Code: sq!

UNION SELECT username, 2, 3, 4 from passwords-- '
```

This query would return:

```
Union Clause

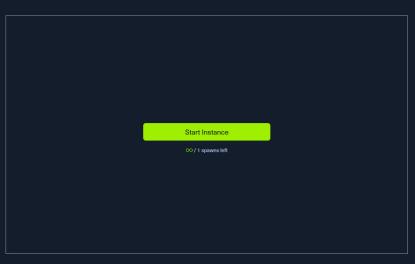
mysql> SELECT * from products where product_id UNION SELECT username, 2, 3, 4 from passwords--'

| product_1 | product_2 | product_3 | product_4 |

| admin | 2 | 3 | 4 |
```

As we can see, our wanted output of the 'UNION SELECT Username from passwords' query is found at the first column of the second row, while the numbers filled the remaining columns.





Waiting to start...



