

[Academy home](#)[Back to all topics](#)

SQL injection



> UNION attacks

SQL injection UNION attacks

When an application is vulnerable to SQL injection, and the results of the query are returned within the application's responses, you can retrieve data from other tables within the database. This is commonly known as a SQL injection UNION attack.

The `UNION` keyword enables you to execute one or more additional `SELECT` queries and append the results to the original query.

```
SELECT a, b FROM table1 UNION SELECT c, d FROM table2
```

This SQL query returns a single result set with two columns, containing values from columns `a` and `b` in `table1` and columns

For a `UNION` query to work, two key requirements must be met:

- The individual queries must return the same number of columns.
- The data types in each column must be compatible between the individual queries.

To carry out a SQL injection UNION attack, make sure that your attack meets these two requirements. This normally involves finding

- How many columns are being returned from the original query.
- Which columns returned from the original query are of a suitable data type to hold the results from the injected query.

Determining the number of columns required

When you perform a SQL injection UNION attack, there are two effective methods to determine how many columns are being returned.

One method involves injecting a series of `ORDER BY` clauses and incrementing the specified column index until an error occurs. For example, if you inject a string within the `WHERE` clause of the original query, you would submit:

```
' ORDER BY 1--  
' ORDER BY 2--  
' ORDER BY 3--  
etc.
```

This series of payloads modifies the original query to order the results by different columns in the result set. The column in an `ORDER BY` clause that you don't need to know the names of any columns. When the specified column index exceeds the number of actual columns in the result set, an error occurs.

```
The ORDER BY position number 3 is out of range of the number of items in the select list.
```

The application might actually return the database error in its HTTP response, but it may also issue a generic error response. In either case, as long as you can detect some difference in the response, you can infer how many columns are being returned from the original query.

The second method involves submitting a series of `UNION SELECT` payloads specifying a different number of null values:

```
' UNION SELECT NULL--  
' UNION SELECT NULL,NULL--  
' UNION SELECT NULL,NULL,NULL--  
etc.
```



If the number of nulls does not match the number of columns, the database returns an error, such as:

```
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of
```

We use `NULL` as the values returned from the injected `SELECT` query because the data types in each column must be compatible. `NULL` is convertible to every common data type, so it maximizes the chance that the payload will succeed when the column count matches.

As with the `ORDER BY` technique, the application might actually return the database error in its HTTP response, but may return a successful response if the number of nulls matches the number of columns, the database returns an additional row in the result set, containing null values. The response depends on the application's code. If you are lucky, you will see some additional content within the response, such as additional data. However, values might trigger a different error, such as a `NullPointerException`. In the worst case, the response might look the same as the original response. This would make this method ineffective.

LAB

PRACTITIONER

SQL injection UNION attack, determining the number of columns returned by the query →

Database-specific syntax

On Oracle, every `SELECT` query must use the `FROM` keyword and specify a valid table. There is a built-in table on Oracle called `DUAL`. The injected queries on Oracle would need to look like:

```
' UNION SELECT NULL FROM DUAL--
```

The payloads described use the double-dash comment sequence `--` to comment out the remainder of the original query following the injected query. The sequence must be followed by a space. Alternatively, the hash character `#` can be used to identify a comment.

For more details of database-specific syntax, see the [SQL injection cheat sheet](#).

Finding columns with a useful data type

A SQL injection UNION attack enables you to retrieve the results from an injected query. The interesting data that you want to retrieve is often found in one or more columns in the original query results whose data type is, or is compatible with, string data.

After you determine the number of required columns, you can probe each column to test whether it can hold string data. You can do this by placing a string value into each column in turn. For example, if the query returns four columns, you would submit:

```
' UNION SELECT 'a',NULL,NULL,NULL--
' UNION SELECT NULL,'a',NULL,NULL--
' UNION SELECT NULL,NULL,'a',NULL--
' UNION SELECT NULL,NULL,NULL,'a'--
```

If the column data type is not compatible with string data, the injected query will cause a database error, such as:

```
Conversion failed when converting the varchar value 'a' to data type int.
```

If an error does not occur, and the application's response contains some additional content including the injected string value, then the column can hold string data.

LAB

PRACTITIONER

SQL injection UNION attack, finding a column containing text →

Using a SQL injection UNION attack to retrieve interesting data

When you have determined the number of columns returned by the original query and found which columns can hold string data, you can use a SQL injection UNION attack to retrieve interesting data. Suppose that:

- The original query returns two columns, both of which can hold string data.
- The injection point is a quoted string within the `WHERE` clause.
- The database contains a table called `users` with the columns `username` and `password`.



In this example, you can retrieve the contents of the `users` table by submitting the input:

```
' UNION SELECT username, password FROM users--
```

In order to perform this attack, you need to know that there is a table called `users` with two columns called `username` and `password`. You have to guess the names of the tables and columns. All modern databases provide ways to examine the database structure, and you can use these to find the names of the tables and columns.

LAB

PRACTITIONER

SQL injection UNION attack, retrieving data from other tables →

Read more

- [Examining the database in SQL injection attacks](#)

Retrieving multiple values within a single column

In some cases the query in the previous example may only return a single column.

You can retrieve multiple values together within this single column by concatenating the values together. You can include a separator character to make the values more readable. For example, on Oracle you could submit the input:

```
' UNION SELECT username || '~' || password FROM users--
```

This uses the double-pipe sequence `||` which is a string concatenation operator on Oracle. The injected query concatenates together the `username` and `password` fields, separated by the `~` character.

The results from the query contain all the usernames and passwords, for example:

```
...
administrator~s3cure
wiener~peter
carlos~montoya
...
```

Different databases use different syntax to perform string concatenation. For more details, see the [SQL injection cheat sheet](#).

LAB

PRACTITIONER

SQL injection UNION attack, retrieving multiple values in a single column →

Want to track your progress and have a more personalized learning experience? (It's free!)

SIGN UP

LOGIN



Find SQL injection vulnerabilities using Burp Suite

TRY FOR FREE

Burp Suite

[Web vulnerability scanner](#)
[Burp Suite Editions](#)
[Release Notes](#)

Vulnerabilities

[Cross-site scripting \(XSS\)](#)
[SQL injection](#)
[Cross-site request forgery](#)
[XML external entity injection](#)
[Directory traversal](#)
[Server-side request forgery](#)

Customers

[Organizations](#)
[Testers](#)
[Developers](#)

Company

[About](#)
[Careers](#)
[Contact](#)
[Legal](#)
[Privacy Notice](#)

Insights

[Web Security Academy](#)
[Blog](#)
[Research](#)



[Follow us](#)

© 2023 PortSwigger Ltd

