# Hacking and SQL Injections

**Disclaimer: This post goes into a bit of SQL logic.**

A friend of mine asked me a very interesting question the other day which I was quite unqualified to answer but did anyway. He asked:

*“If someone knows how to code, does it mean that he/she knows how to hack?*”

I answered "no" pretty quickly without putting much thought into it since I dabble in some coding for data manipulation myself but did not have the slightest clue when it comes to "hacking". So, weeks past and I started to write a lot of SQL scripts for work (in Microsoft Access unfortunately) which got me thinking:

*"SQL is such an unexciting and task-specific language, what other exciting things can people do with it?"*

What I found led me to have second thoughts regarding the answer I gave my friend. Before we delve any further, we have to make sure that you understand 2 important terms/concepts which are integral to this post.

## What is SQL?

"Structured Query Language" or more commonly known as SQL (pronounced as sequel), is basically a computer programming language designed for handling relational databases. If you are unsure about what relational/linked databases are, it is basically different tables filled with data that are linked together by some unique ID.

SQL allows the manipulation of these databases as well as getting filtered results through the use of queries. A typical SQL query for getting filtered results from a database looks like this:

**SELECT** [Column name 1], [Column name 2],...[Column name **N]**   
**FROM** [Table name]   
**WHERE** [Condition];

or if we use some less-generic names, it would look something like:

**SELECT** name

**FROM** students

**WHERE** id = 5;

Note that this is just a very simple example and a lot more can be done with more complicated logic, but that is left for another post. Notice that an important syntactical feature about SQL is that the symbol “;” represents the end of the script as shown above. Now that we have a slight idea on what SQL is and what it's used for, let's move on to our second concept/term.

**What is hacking?**

According to Technopedia, "*Hacking generally refers to unauthorized intrusion into a computer or a network. The hacker may alter system or security features to accomplish a goal that differs from the original purpose of the system*". So, in a way, if a person can get sensitive information (such as employee details or passwords) out of a database that is supposed to be secure, we can say that the database got hacked. Contrary to popular belief, not all hackers are trying to get nuclear launch codes from the white house.

**SQL Injection**

A SQL Injection (SQLi) refers to an attack wherein an attacker can execute malicious SQL statements that control a web application’s database server. “*Malicious SQL statements”*may not make much sense now but will be explained later on. It is one of the oldest, most prevalent and dangerous of web application vulnerabilities. It really should not work anymore in this day and age but somehow, still does. We will begin to explain this hack through the use of an example. Let's say I have a web page for a store I am running in which the customer can search for products names to get the respective prices.

INSERT PICTURE HERE

Let us first try to understand what is actually happening when a customer types something into the search bar and clicks submit? Firstly, a POST request is sent from the website to the server (PHP for example), and the server constructs SQL query based on whatever the customer entered in the field.

Once the query is processed, the relevant lists and tables gets returned to the customer on the website. So, in essence, what he/she is typing into the search bar is part of a bigger piece of SQL code which looks something like this:

INSERT PICTURE HERE

Firstly, let me explain what the % (ampersand) symbol means. It acts as a wildcard for any number of characters before/after the word, depending on where you place it. So "%Nail% “would mean any product name that contains the word nail. Now, let's say we were to input nail into the search bar, the full query behind the scenes would look like:

**SELECT** \*   
**FROM** [Product Details]   
**WHERE** [Product Name] LIKE "%Nail%";

And a possible output could look something like:

|  |  |  |
| --- | --- | --- |
| **Product ID** | **Product Name** | **Price** |
| 50 | Nails – Silver | $1.20 |
| 32 | Nail gun | $65.00 |
| 6 | Nails – Copper | $1.10 |

Which is exactly what a customer would expect from the search! What if however, we try to break the system and input something like " (A single quotation mark) in the search bar?

INSERT PICTURE HERE

If website churns out a blank page with a message saying that there are no products called ", then the website is probably safe from an SQL injection. If however, a message displaying some sort of *"Server Error Warning"* pops up, it means that there is a way for us as customers to input some malicious SQL code and get more information than the store intended to provide, or in other words, "*We're in"*. What happened is that, instead of using " as a character, it was actually used as a [control structure](https://www.chegg.com/homework-help/definitions/basic-control-structures-3), or part of the source code. So, behind the scenes, the SQL code that the server reads are:

**SELECT** \*

**FROM** [Product Details]   
**WHERE** [Product Name] **LIKE** "%"**%"**;

As you can see, the first % is perfectly fine when it is surrounded by a pair of quotation marks, the issue comes when the second % is not, which is why a "Server Error" message pops up. So, what can we do in order to get some sensitive information? Firstly, we need to fix the malformed query. One way to do this is to use the “;” symbol to close the query and comment out anything else behind. Let's enter this into the search bar:

INSERT PICTURE HERE

I know this looks absolutely ridiculous and cryptic, but let's look at the underlying code behind the interface:

**SELECT** \*

**FROM** [Product Details]   
**WHERE** [Product Name] **LIKE** "% **";--** %"

Firstly, we can see that the first % still has a pair of quotation marks surrounding it so it is syntactically sound. After that, the “;” symbol ends the command and -- makes it so that anything after it becomes a comment and not executable code. So, in essence, what the server is going to execute is:

**SELECT** \*

**FROM** [Product Details]   
**WHERE** [Product Name] **LIKE** "%";

Which in English, translates to "*Give me everything from the product details table where the product name is anything*". Amazing isn't it? Who would've thought that typing ";-- in the search box will churn the list of all product details? Now that we have gotten our first foothold on the store's database system, what else can we do in order to get the private/sensitive stuff? (Technically the next step would be to identify what version of SQL the server is using because there are some syntactical differences between them. This can be done through extending the query further, but I will not go into it here.) Let me introduce you to the UNION command in SQL. What UNION basically does is it takes 2 tables with the same number of columns and just puts them on top of the other, which would be the only way we can get any other information out of the database as SQL runs only 1 query at a time. For example:

**SELECT** \*

**FROM** [Product Details]   
**WHERE** [Product Name] **LIKE** "%Nail%"

**UNION**

**SELECT** \*

**FROM** [Product Details]   
**WHERE** [Product Name] **LIKE** "%Screw%"

Would return something like this:

|  |  |  |
| --- | --- | --- |
| **Product ID** | **Product Name** | **Price** |
| 50 | Nails – Silver | $1.20 |
| 32 | Nail gun | $65.00 |
| 6 | Nails – Copper | $1.10 |
| 2 | Screws – 5cm | $0.30 |
| 43 | Screws – 3cm | $0.20 |

What that line of code did was just to stack the 2 results vertically. So instead of using UNION to stick Screw details below our Nail details, why not we try to stick some private information below our Nail data? First off, we have to identify some issues we currently have:

* We need to know what other tables there are within the database and their names
* We need to recognize that our query must output 3 columns of data in order for the UNION to work

Note that we do not technically know that the table we are getting product information from is even called [Product Details]. I just wrote it like that to make the code easier to read. Taking these points into consideration, we can try something like:

**SELECT** ?

**FROM** ?

**WHERE** ? **LIKE** "%Nail%"**UNION** ( **SELECT** [TABLE\_NAME], [TABLE\_SCHEMA], [1] **FROM** [INFORMATION\_SCHEMA.tables] ) ;--%";

Where the words in red are literally what you will type into the search bar and everything else is embedded in the website and cannot be seen by the customer. Firstly, recognize that syntactically, there are no errors as everything is properly wrapped with quotation marks and anything after the -- is not executable code. Secondly, notice that we are using the UNION command again but this time, we are joining our Nail data to 3 columns:

* Table names
* Table schema
* Placeholder column filled with 1s

INFORMATION\_SCHEMA is just a directory for the database that MySQL databases have. Sort of a table of tables. If we input the red part into the search bar and click submit, we should get some output like:

|  |  |  |
| --- | --- | --- |
| **Product ID** | **Product Name** | **Price** |
| 50 | Nails – Silver | $1.20 |
| 32 | Nail gun | $65.00 |
| 6 | Nails – Copper | $1.10 |
| Employee Details | INFORMATION\_SCHEMA | 1 |
| Product Details | INFORMATION\_SCHEMA | 1 |
| Sales Records | INFORMATION\_SCHEMA | 1 |
| Expense Records | INFORMATION\_SCHEMA | 1 |

Which is absolutely ridiculous because now that you have the names of the tables, you can just manipulate the query again to get any of the data that you might find interesting. The next step is to figure out what columns each table has. The reason we can't just ask SQL to select everything in the table is because of our UNION command, which requires the number of columns to be the same as our Nails output. Suppose we want to find out the columns in the "Employee Details" table. We can do this by writing:

**SELECT** ?

**FROM** ?

**WHERE** ? **LIKE** "%Nail%"**UNION** ( **SELECT** [COLUMN\_NAME], [1], [1] **FROM** [INFORMATION\_SCHEMA.columns] **WHERE** [TABLE\_NAME] =

"Employee Details" ) ;--%";

As before, only the code in red is typed into the search bar. Now we are still accessing the INFORMATION\_SCHEMA but selecting all the column titles within the database if and only if the table name is "Employee Details". Clicking submit would give us an output that may look something like:

Department

Finally, let's say we want to know the passwords and bank details for all their employees. We can just rewrite our search bar input into:

**SELECT** ?

**FROM** ?

**WHERE** ? **LIKE** "%Nail%"

**UNION**

(**SELECT** [Employee Name], [Employee Password], [Employee Bank Details]

**FROM** [Employee

Details]) ;--%";

Which will give us an output like:

|  |  |  |
| --- | --- | --- |
| Product ID | Product Name | Price |
| 50 | Nails – Silver | $1.20 |
| 32 | Nail gun | $65.00 |
| 6 | Nails – Copper | $1.10 |
| Alex | Alex123 | 123-456-7 |
| Jesse | Destiny2 | 543-192-2 |
| Jeremy | Ilovesneakers | 123-312-9 |

This is just catastrophic for the store if customers can easily access all their private data. Think about it, we have used something as simple as a search bar, a UNION command and some simple queries on a website to hack into the store's database! Note that this is an over simplified example of a *first order SQL injection* and there are many things the store can do to improve the security on its website, the easiest being to just ban the use of any symbols in the search bar. There are more complicated *second-order SQL injections* where hackers would use part of a malicious query as their username, which doesn't get flagged, and then inject the rest into the system by changing their password. (So brilliant right?) Going back to the question which my friend asked, I am now more convinced that if you know how to code, then you have everything you need to hack into something. The only thing lacking would be the mindset to find loopholes and weaknesses in the design of a system. That being said, performing an SQL injection is illegal and it should not work anymore as most companies have improved their web security over the years.

Just a quick note, if you have received any phishing e-mails scams you could potentially practice your SQL injection skills to hack into the scammer’s database to delete all their tables like a modern day virtual Batman.