# **SQL Injection Final Report**

Erich Elshoff

Oakland University

CSI 4480: Information Security Practices

Dr. Anyi Liu

December 5, 2021

**Disclaimer***: All attacks are done for informational and educational purposes only. Any recreation of these attacks against an unconsented party is illegal.*

## Intro and Goals

The goal of this project is to build a website with SQL vulnerabilities and subsequently launch a series of SQL injection attacks to penetrate the system. Through both manual and automatic attack styles, the victim website will be attacked with the goal of obtaining sensitive information in the database, as well as using various commands to add, delete, and modify the database. Through these attacks, I will learn how attackers can gain access to a database, how much damage they can do, and how to prevent it. As a follow up to the attack, and using knowledge gained from the attack tests, the website will be rebuilt with more robust defenses.

**Tools**

The front end for this project will be a simple website written in HTML. This site will consist of a table and search form that are linked to a database that contains information about various grocery store products. The back end will consist of php code that uses SQL commands to communicate with a MySQL server. The site will be hosted locally. This website will be created with intentionally vulnerable code that will allow us to exploit it. The database schema will also contain additional tables for testing purposes and as attack targets.

This project will utilize a third-party software, SQLMap. This software allows us to automatically launch SQL injection attacks. This is a useful penetration testing software that allows us to pinpoint the vulnerabilities of our website.

## The Problem

SQL is a database management language that has been around for decades. The first SQL injection attack was documented in 1997 and since has continued to be one of the most common and devastating types of attacks. Although this type of attack is well known and documented, SQL injections are still prevalent in 2021, ranking as the third most common web application attack on OWASP’s top ten list. The motivation for this project came from wanting to understand how this type of attack can be so common and to learn how to prevent it.

## The Victim

The victim consists of two components: the website and the database.

**Store Webpage** – This website consists of 3 unique pages that serve different testing purposes. Each page is responsible for acquiring user input via a search form, passing that information to the SQL server, then printing the results in a table, but each one takes a different approach.

1. **Index.php** – This page uses php and mysqli commands. It is vulnerable to SQL injections but doesn’t support multiple queries in one line.

### Text Description automatically generated

1. **Multi.php** – This page is identical to index.php, but calls the mysqli\_multi\_query() function, which overrides the limitation in index.php. This makes the website even more vulnerable to SQL injections, enabling the usage of other commands like ADD, DROP, and MODIFY.



1. **Secure.php** – This page uses prepared statements to sanitize data. It is not vulnerable to SQL injections.

### Text Description automatically generated

**Inj** – The database schema. Hosted on the MySQL Workbench database management software using Xampp to host a local Apache and SQL server. This schema consists of three tables that are not linked: the products table, which contains mock grocery store items and prices, the user table, which contains sensitive user information such as passwords and credit card credentials, and a test table. The database also contains an information schema table, which contains metadata about the database and will serve as a vector for attacks.

## Manual SQL Injection

The attacks in this section will be done manually using MySQL queries within the search form of the website. Because I am acting as both the victim and the attacker, I will have the advantage of knowing the database language being used as well as the code. Typically, an attacker will use trial and error to try to guess the SQL language being used. There are four types of SQL injection attacks that I will demonstrate and explain: error, union, blind, and Boolean.

**Error Based** - This type of injection attack attempts to inject commands that will result in the database producing error messages. These error messages can be used to determine various things about the database. 

By entering a single quote (‘) into the search form of the website, the SQL server responded with the above error message. This is bad news for the website, as an error message like this tells the attacker that the website is vulnerable. The appropriate response would be a message that alerted the user that no products like (‘) exist.

**Union Based –** This injection attack uses the UNION SQL operator to combine multiple SELECT statements. This can be exploited to generate responses from the SQL server that contain data that can be leveraged by the attacker.



The above SQL query is what the server will see when we use a union-based attack. The string “beans%’ UNION SELECT 1,2,3,4— “produced this query. It is using SELECT to both search for beans in the product table, but also selects columns 1,2,3, and 4 and prints those numbers in the table, seen below. Graphical user interface, application

Description automatically generated

This attack can be taken further, replacing the integer values with column names and specifiying the table. Doing so will print database information directly onto the webpage.

Chart

Description automatically generated with low confidence

**Time Based -** This is a blind SQL injection attack, meaning that the server wont print responses to the webpage like error and union-based attacks. To get a response, attackers use the SLEEP SQL command to try to get the server to wait for several seconds. If the attacker observes the website loading for the specified amount of time, they know the SQL query worked and the server is responding.



The above code is an SQL query that results from typing the string “beans%’ order by SLEEP(5)—“. This command will sleep the server for 5 seconds for every product that matches the “beans” keyword.

**Boolean Based –** This is another type of blind attack. Attackers will try to leverage Boolean logic to get information from the server. By combining a query with a Boolean statement, the attacker can determine if the server is responding or not. For example, if the attacker injects the string “beans%' AND 1=1—“ and it returns a product in the table, but beans%' AND 1=2—returns no product, the attacker can confirm that the injection was successful and the Boolean logic was being processed. This can be utilized to get information about the database. For example, injecting a string like,” beans%' AND (length(database())) = 3—“ will return a product result for beans if the length of the database name is equal to 3 characters and nothing if it is false. Since we know the database is called “inj”, 3 will return true but 6 will be false.

Graphical user interface, text, application, email

Description automatically generated

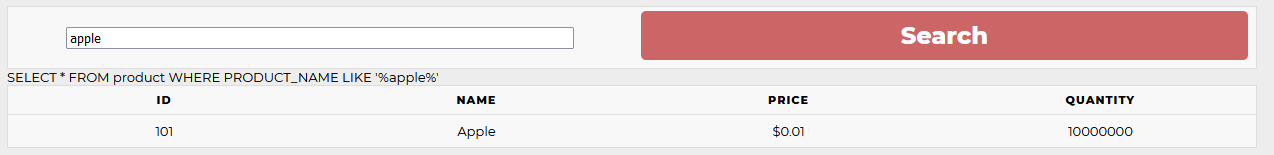
Graphical user interface, text, application

Description automatically generated

**Multi Query Attacks –** These attacks will be launched on the multi.php page. I will demonstrate how an attacker can use INSERT, DELETE, UPDATE, and DROP to modify the database in an SQL injection. Through the mysqli\_multi\_query() function we can request multiple queries in one line. The queries are separated by a “;”.

**INSERT** – By using an INSERT command, we can add an entry to the database with whatever values we want. For example, using the string below will add a very inexpensive apple to the table with 1 million units available.





**UPDATE** – We can modify an entry, for example, changing the price of the apple.



**DELETE –** We can remove that entry using a delete command.



**DROP –** Finally, we can drop the entire table if we wish



## Defense

As we have seen in the attack section of this project, the vulnerability of the website is in the way the php and mysqli code is written. Since the code is written to take user input and pass it to the SQL server, the point of vulnerability is in the way this data is being processed. If you look at the code snippet for index.php, you will notice that the k parameter, which holds the user input, is being put directly into the query that is sent the SQL server. This is where the problem lies because it allows the user to modify the intended SQL query.

To properly defend against this vulnerability, we need to separate the user input and the SQL query. To accomplish this, we will use prepared statements. Prepared statements work by creating a SQL query template that is sent to the server before any user input is supplied. The template will include a placeholder “?” that will later be used to add user input. The SQL server parses the template with the placeholder and then waits for the user data to be added later. When the user input is added later it will go into the placeholder spot. Since the SQL query has already been parsed and checked for syntax, there is no way to change it. For example, if we tried a union based attack on a prepared statement using the string “beans%' UNION SELECT 1,2,3,4—“, the SQL server will look for a product whose name is literally “beans%' UNION SELECT 1,2,3,4—“. The ‘ and – characters will have no effect on the query.

## Testing

In this project I used a third-party software, SQLmap, to do penetration testing on my webpages. This tool is useful both as an attacker and defender. As an attacker, it automatically can determine the languages, versions, database type, vunerable forms, and injection payloads within seconds. As a defender, it can be used to test the security of a webpage and to ensure that injection attacks are prevented.

I will showcase a simple demonstration of using SQLmap to breach a database. First, I ran a crawler through the website to find a vulnerable URL.



This found a URL that was injectable, so I followed up with a standard SQLmap call on the vulnerable URL. The results show the various vulnerabilities and database information.



**Text

Description automatically generated**

We can then use some options to dump the entire users table. -D selects the database and -T selects the table.



Graphical user interface, text

Description automatically generated with medium confidence

If we run the same crawl command on our secure webpage that uses prepared statements, we will get the following message that show that the form is secure.



## Challenges

There were a number of challenges I faced while completing this project. First, I needed to develop a strong understanding of how databases work and how the SQL query language worked. This was necessary to perform injection attacks, as using the correct syntax is critical to success. This knowledge allowed me to understand how these attacks work but also how to defend against them.

Another challenge was creating a webpage that was vulnerable to all types of SQL injections. At the start of the project I wanted to be able to perform SQL injection attacks of all types on my web page, but it wasn’t so simple. The SQL language disables multiple queries by default as a basic means to prevent injection attacks. This does stop some types of attacks like INSERT, DELETE, UPDATE, and DROP, which are impossible to do on one line. To tackle this challenge, I needed to create a modified version of the page that called a special override function to disable this feature.

## Conclusion

This project has shown that SQL injection attacks can be devastating, as they allow an attacker to gain access to precious information and manipulate the database. The solution is simple and effective, but the problem continues to affect web pages. In conclusion, it is critical to understand the potential weaknesses in an application and take the necessary steps to prevent them. The usage of penetration testing tools can also be crucial to ensure that these steps have been taken.

## References

*Prepared statements - manual*. php. (n.d.). Retrieved December 5, 2021, from <https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php>.

Team, H. E. D. U. (n.d.). *How to prevent SQL injection vulnerabilities: How prepared statements work*. HackEDU. Retrieved December 5, 2021, from https://www.hackedu.com/blog/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work.

*Welcome to the OWASP Top 10 - 2021*. OWASP. (2021). Retrieved November 16, 2021, from https://owasp.org/Top10/.