**CPSC 4660 – Final Report**

**SQL Injection Attack Prevention**

**By: Amanda Munden**

**#001148507**

**Due: November 25, 2019**

**Background:**

SQL injection attacks are some of the most common attacks on databases. There are multiple SQL injection strategies that are used to accomplish different goals when attacking a database. Some of the most common types of attacks are using tautologies which can allow you to bypass authentication and retrieve data you would not normally have access to, using a Union query which allows you to return additional data from a query, and using Piggy-Backed queries which allows you to alter the database by removing tables or tuple, changing values or adding values. To test my different preventative techniques that I will be implementing in this project, I will be using attacks from all three strategies to see how accurately each of the prevention methods stops the different kinds of attacks.

**Summary of Component Being Implemented:**

I implemented three separate preventative techniques that are intended to reduce or eliminate SQL injection attacks. The first technique I implemented was “type or length detection” which I put into the HTML file to keep the user from inputting the wrong type of information and to limit how much could be put into the different input fields. The second technique that I implemented was “sensitive key word/character filtering” which I implemented in the PHP code to check for sensitive key words or special characters that could allow other SQL code to be run. Instead of returning an error message like I originally proposed, I decided to add a slash to the input on the backend as to have all the input treated as just standard input and not a piece of code that could be run. Finally, I implemented “tokenization” which was done using a “query parser method” to break down the expected input into tokens and then the actual input into tokens and compare the two sets of tokens. The goal of implementing three different techniques was to determine if a more complex strategy like “tokenization” better prevents SQL injection attacks than a more basic strategy like “type or length detection” or “sensitive key word/character filtering”.

**Implementation Details:**

For the implementation of this project, I started by creating a database that consisted of three tables which were student, faculty, and course. Inside these tables, I used….

I then implemented the front end where I created a menu webpage that had links to adding information to each of the tables and being able to search for information in each of the tables.

After everything was created, I conducted a test of all the SQL injection methods that I intended to use to make sure that the base case was susceptible to all of them.

Then I recoded the HTML and PHP documents to implement the three prevention strategies. First, I took the HTML files and introduced a stricter type for the information that could be input. I also added in a max length of characters that could be put in the input fields on the webpage.

**Evaluation Strategy:**

My evaluation strategy was to apply one of the prevention strategies to my test database and then try multiple kinds of attacks against the database and see how many of the attacks the strategy can prevent and how accurately it prevents the attacks. I will repeat this for all three prevention strategies to see which one can prevent the most attacks and if there is a difference in how well a simple strategy prevents attacks compared to a more advanced and complex strategy.

**Results of the Evaluation:**

**Conclusions:**

**References**

Yeole, A. S., Meshram, B. B. (2011). Analysis of different technique for detection of SQL injection. *ICWET '11: Proceedings of the International Conference & Workshop on Emerging Trends in Technology*. Publisher ACM.

Zhang, Haiyan, Zhang, Xiao. (2018). SQL Injection Attack Principles and Preventive Techniques for PHP Site. *CSAE '18: Proceedings of the 2nd International Conference on Computer Science and Application Engineering*. Publisher ACM.