**Module 2: SQL Injection**

Zane Milo Deso

Southern New Hampshire University

CS-405: Secure Coding

**SQL Injection Post-Mortem**

**Introduction**

As a Computer Science undergraduate attending Southern New Hampshire University (SNHU), I have been tasked with summarizing my experiences during an assignment involving detection and prevention of SQL injection. In this paper I will briefly define SQL injection, outline my approach, why it works to prevent SQL injection, identify trouble areas I encountered during the assignment, and how I overcame the obstacles.

**SQL Injection**

SQL injection is a form of cyber-attack that involves maliciously using database query methods to gain access to or even alter the database. The attack involves altering the input that would normally be used to return records normally, or log in, etc. For example, a bad actor may add additional characters and symbols to the string input of the query as to alter the execution of that query at the database, or some middle layer, level. By altering the input, the database will misconstrue the intent of the query and allow for actions not otherwise possible, based on the system's purpose and architecture.

**Approach to Detect and Prevent SQL Injection**

Initially, I planned on using prepared statements to prevent and detect malicious injections attacks. Though, after implementation, I found that I was limited in scope due to the query arguments used in this assignment lacking the parameter required by the sqlite3 methods to operate functionally. I spent time squabbling about attempting to get it to work, yet to no avail it was time to shift gears.

After a quick google search, I decided to refer to regex to help find patterns that would be found within SQL injections. Shortly after, I integrated a function to check from suspicion of SQL injection, placed a call to this function within the assigned function, followed by a return false statement. This was not an easy one to divert to, as I felt it was wrong to do this type of detection since there are, without doubt, patterns that will cause injection attacks that go undetected by the regex function. For sake of brevity, I opted to keep this approach and will try to reimplement the properly prepared statement in future code for proof of concept and secure coding practices.

**Why does the Approach work?**

The approach simply works due to finding the underlying pattern of characters and symbols used by bad actors to attack the database. The regex library and pattern matching aided in detecting common SQL query language to help discern from a normal input or a malicious one. By incorporating the full stop return false statement, I am forcing the program to disallow any further input, thereby dissallowing the SQL injection attack.

**Problems Encountered**

As mentioned, I encountered an issue when I attempted to implement a prepared statement pattern. Although, due to the constraints of the function I found myself in, I would have had to code a large amount of code to dynamically scrap input prior to processing to identify the potential “input” variable, which in this assignment was simulated.

**Overcoming the Obstacle**

To handle the limited scope, I opted to use regex pattern matching via a custom suspicion function. This function was called within the query function to be edited and used to match common SQL query language, and potentially malicious injections followed by a full stop return false. This fully ensures no injection will be executed by the database.

**Conclusion**

In conclusion, SQL injection remains a leading security threat across the globe. By taking a Secure Software Development Life Cycle approach, developers may deny hackers the ability to inject dangerous code into their databases and software. Though limited in scope by the assignment, I still attempted to incorporate what I believe is the way to dynamically catch SQL injection, with the code now commented out, and implemented a custom regex pattern matching approach to ameliorate the debacle. Even if this is not the perfect approach, since the reminder of the queries will not be executed, it still disallows any potential threats to the database itself. This was overall a strong learning experience, and one I will carry with me.