The Utilization of Sandboxing to Prevent SQL Injection Cyber-Attacks

A Technical Report Submitted to the Department of Engineering and Society

Presented to
The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, Computer Science

Gabriel Edwards

Spring 2022

Advisors

Rosanne Vrugtman, Department of Computer Science

Daniel G. Graham, Department of Computer Science

The Utilization of Sandboxing to Prevent SQL Injection Cyber-Attacks

CS4991 Capstone Report, 2022

Gabriel Edwards
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
gse3ez@virginia.edu

Abstract

SQL injection attacks are one of the most common forms of cyber-attack, with millions of them threatening major databases across the internet each day. To combat this problem and isolate potential threats, I propose applying the concept of sandboxing to database queries. The general method of sandboxing a database is to clone (copy) the database before applying an SQL query to it, apply the query to the separate cloned database, and then compare the two databases to determine if any changes were made. The goal of the proposed research would be to determine both the effectiveness of sandboxing in detecting and preventing an array of SQL injection queries, and the impact sandboxing would have on the average load and response times of a website.

The effectiveness of sandboxing in detecting injections would vary depending on its intended application. SQL injections are very easy to detect in search-based queries, but the process would be less reliable for table update queries. In terms of the effect on a website's usability, sandboxing is likely to cause a notable delay in response time. Further research comparing this solution to others that are already in use would provide important insight into whether the effectiveness of sandboxing is worth the increased response time it causes.

1. Introduction

In 2012, the hacking group Team GhostShell stole records from over 100 universities worldwide [7]. The names, addresses, emails, usernames, passwords, and even more sensitive info were stolen from over 36,000 unsuspecting, helpless university students, like those at the University of Virginia. The vulnerability that allowed this access, which security personnel across the world were unprepared for, was simple yet devastating: SQL injection. SQL injection attacks are one of the most common forms of cyber-attacks today, having comprised approximately two thirds of all cyber-attacks worldwide between 2017 and 2019 [6]. These attacks target SQL databases, where information is stored in tables and can be added, updated, retrieved from, deleted from, etc. using queries statements [2].

Typically, websites will ask for information from the user (like a username or password), which is then used within a query to access some information in a given table. Injection attacks work by "injecting" statements into these queries instead, which trick the website into accessing information that it was not given the proper credentials for.

There already exist several methods of countering SQL injections, which employ a range of techniques and often attempt to handle the issue at various stages of querying. Examples of these include: using prepared statements, which bind query parameters to specific data types (so that commands are treated as plaintext); validating input, such as ensuring an email address given to complete a query is in valid form; and requiring login credentials to update a table [3].

2. Related Works

Research in this area has primarily focused on methods of attack prevention that address the queries themselves, rather than their effects. One of the oldest and more widely used of these methods is to use prepared statements, as discussed by Tankic (2018). Prepared statements are simply prewritten statements executed by code, where variables given by the user are bound to pre-determined data types. This differs from the focus of this paper in that potentially malicious commands are converted to plaintext (and neutered) before the query is executed, as opposed to examining the result of the query.

A more recently proposed version of query sanitation comes from research by Abikoye et al. (2020), which involves using the Knuth-Morris-Pratt string matching algorithm to simply compare a given query to known SQL injection commands, and reject matching queries. This differs from this research in the same way as prepared statements; however, it involves a similar requirement of comparing strings, and suggests the potentially effective KMP algorithm.

In addition to query sanitization methods, the strategy of Object-Relation Mapping, which is very similar to this research, involves copying a given table to another structure, where queries are then safely performed on that structure. It differs, however, in that tables are copied into an entirely separate data structure, as opposed to simply a second table. Creating a mapping from a relational database to a data structure is said by Lorenz et al. (2017) to be very difficult, and this research hopes to negate this issue.

3. Proposed Design

The research in this paper is focused on applying a new method of preventing SQL injection attacks, and determining its viability against multiple forms of injection.

3.1 Types of SQL Injection Attacks

SQL injection attacks can take several different forms, meant to produce different results. The first type is those that directly manipulate the data in a given table, for instance removing an entry or deleting the entire table. The second type is those which cause a query to return different results than expected. One example of this is that of forcing a table of products on an online retailer’s website to display a table of user’s login information instead. The final type addressed by this proposal is those which cause error messages to be displayed on the webpage, which could potentially give an attacker valuable information about database structure.

3.2 Proposal Overview

The crux of the proposed solution is the idea of sandboxing. Sandboxing is the practice of running a particular software along with all dependent software in a separate environment. The purpose is to limit the impact of an exploited vulnerability in that software; the attacker cannot reach any important data or computer systems, since there is no path from the vulnerable software. This proposal suggests that to prevent SQL injections, a given table that is meant to be accessed could be copied into a separate database, where it would then be queried. In this way, any alterations made to the table by an injected query would not affect the original data. Following this, the two tables would be compared entry by entry to determine if the query was malicious.

To address the second two types of SQLi attacks, the query results would also be sandboxed and examined. This is much more implementation-based, and would require building a sample of expected results with the widest coverage possible for the given task. The result of the query would then be compared to each expected result, using regular expression to determine if it contains the correct type of data. Also, a simple error handler would be written to catch unexpected errors and prevent them from being displayed directly on the given website.

3.3 Experimental Design

There are two questions about the proposed design that must be answered: what amount of SQL injection attacks are caught using this method; and how much time does it add to a given website operation? To test, a sample website would be written in the HTML language, connected to two SQL databases: one containing populated tables of user login information and fake product information, and the other containing two empty tables. The website would contain a field for the user to input a given product item name to display on the website, which would be meant to query the product table for an entry matching that name. Before querying, a timer would start, and both tables would be copied in the manner described. The query would then be executed on the copied product table. To determine if the table was manipulated, each entry of both tables would be compared with the original versions, where any deviations would be logged. To determine if login information (which would be targeted) is returned, the result would be converted to a string and compared to a list of unique identifier strings given to all product entries; any misses would be logged. Following these operations, the recorded time would also be logged. A sample of typical SQLi attack queries would then be executed on the database, for various table sizes ranging from 100 entries to 10 million entries.

4. Expected Results

The time addition of the proposed process is very straightforward to calculate. First, each entry in both tables would need to be copied between databases adding N operations (where N is the number of entries). Then, each table entry would need to be compared, adding N more operations. Finally, the result would need to be compared to the list of product names, adding M operations (where M is the number of products). Therefore, 3N + M operations are added, upper bounded by 4N. Both comparing and copying happen in constant (negligible) time, so this process would add a linear amount of time. This means that at small table sizes, the extra time would likely not be noticeable; at sizes approaching 10 million, they could potentially be.

It is likely that determining the effectiveness of the experiment would be less straightforward. It is expected that it would catch all of the potential data leak attacks possible. Regex comparison is a process that is known to be effective, and any login info returned by a query would certainly be flagged as not containing a unique product identifier. The issue is that this experiment cannot guarantee the effectiveness of the process for all use cases. This lies in the fact that there is an unknowable amount of possible query types and data that could be returned, all of which requiring unique testing.

5. Conclusion

The results expected from this proposal point to a promising proof of concept, with a reasonable level of effectiveness with small, simple cases. That said, real-world databases and injection attacks are wildly complex and diverse, much more so than those discussed here. Any application of sandboxing would likely be implementation-based, and need to cover an extremely wide range of attack forms. These qualities would be very difficult to design and test fully, and would likely cause a considerable time addition. Though this solution may not be particularly practical or groundbreaking, this proposal demonstrates that there are always other solutions somewhere to a given problem.

6. Future Work

The most immediate work still to be done is implementing the proposal; the expectations seem sound, but they must be tested. Following, testing must be expanded to more realistically sized and complex databases, as well as to a much wider variety (and intricacy) of SQL injection examples. As it's likely for this testing to uncover undesirable time additions at real world use cases, work into a cleverer and more efficient sandboxing algorithm will also be necessary.

References:

[1] Oluwakemi Abikoye, Abdullahi Abubakar, Ahmed Dokoro, Oluwatobi Akande, and Aderonke Kayode. 2020. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. *EURASIP Journal on Information Security*. 14. (Aug. 2020). DOI: https://doi.org/10.1186/s13635-020-00113-y

[2] Daniel Calbimonte. 2021. SQL Definition. (April 2021). Retrieved February 23, 2022 from https://www.sqlshack.com/sql-definition/#:~:text=Basically%2C%20SQL%20stands%20for%20Structured,%2C%20procedures%2C%20etc.).

[3] Evan Klein. 2019. How to Defend Your Business Against SQL Injections. (June 2019). Retrieved February 23, 2022 from https://logz.io/blog/defend-against-sql-injections/

[4] M. Lorenz, J. Rudolph, Guenter Hesse, M. Uflacker, and H. Plattner. 2017. Object-Relational Mapping Revisited - A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies. In *Proceedings of the Hawaii International Conference on System Sciences*, January 4, 2017, Manoa, Hawaii. DOI:10.24251/HICSS.2017.592

[5] Jasmine Tankic. 2018. SQL Database Performance using Prepared Statements and Stored Procedures. https://www.researchgate.net/publication/326786523_SQL_Database_Performance_using_Prepared_Statements_and_Stored_Procedures

[6] Jai Vijayan. 2019. SQL Injection Attacks Represent Two-Third of All Web App Attacks. (June 2019). Retrieved February 23, 2022 from https://www.darkreading.com/attacks-breaches/sql-injection-attacks-represent-two-third-of-all-web-app-attacks

[7] Jaikumar Vijayan. 2012. Group says it hacked systems at 100 major universities Harvard, Stanford, Penn among those hit; breached data mostly innocuous, analyst says. (October 2012). Retrieved February 23, 2022 from https://www.computerworld.com/article/2491950/group-says-it-hacked-systems-at-100-major-universities.html