Cybersecurity Update 1

Ester Aguilera, Jingru Dou, Kelsey Knowlson, Victoria Lien,
Bryce Palmer, Emely Seheon, Lasair Servilla

Due: February 25, 2024

Table of Contents

1 Problem Statement **2**

2 Motivation **2**

3 Methodology **2**
3.1 Work to Date 3
3.2 Future Plans 3

4 Intermediary Findings **3**
4.1 SQL Vulnerabilities 4
4.2 MySQL 4
4.3 Prevention Strategies 5
4.3.1 Input Validation 5
4.3.2 Parameterized Queries 5

5 Group Member Roles **6**

6 Expected Results **6**

References **7**

1 Problem Statement

SQL is a ubiquitous back-end database management language that is used across a plethora of platforms to organize and store a variety of confidential information. The specific challenge at hand is to identify potential security vulnerabilities within a typical SQL framework by implementing an instance of a SQL database as way to test its security features.

Due to SQL's pervasive use across a diverse range of industries from consumer shopping to student and employee databases, it is crucial to uphold robust security features to prevent possible exploitation.

2 Motivation

As of now, SQL injection is still a reliable and strong attack route that can compromise sensitive data by taking advantage of vulnerabilities in database-driven systems. This concern is raised as organizations increasingly rely on databases to store sensitive information; the exploitation of SQL injection vulnerabilities poses a significant risk to data integrity and system security. Because so many organizations rely on database systems to handle vital data, a successful SQL injection attack might have disastrous repercussions, from system failures to data leaks.

Through the implementation of strong input validation procedures and a thorough understanding of SQL injection vulnerabilities, this project aims to further the more general objective of improving cybersecurity safeguards in database-driven environments. Furthermore, with new advancements in this area, the potential impact of SQL injection attacks has increased, making it more important than ever for cybersecurity efforts to stay ahead of sophisticated threats.

3 Methodology

In order to test for vulnerabilities in a legal manner we will create a simplified version of a web interface to a SQL database following the standard practices for safe guarding such systems. An example standard being the sanitizing of user input by treating it as a parameter rather than a string. Once we

have a functioning service we will use our knowledge of how the system was set up to strategically attempt to break it. Breaking the system would mean gaining information from the database that was private or in a unintended manner.

3.1 Work to Date

At this point each team member has contributed to research related to the understanding and set up of our controlled SQL system. The preliminary set up of the front-end web interface and the back-end SQL database has been started. We have chosen to use the free community version of MySQL with the Python connector package to interface with the server. We also have a plan in place for how each team member is going to move forward on the project.

3.2 Future Plans

In the future we will continue the setup of the test system. This includes finishing the back-end work of interfacing with the SQL database. We will also complete the front-end which includes a web based UI following HTTP protocol to relay user interactions to the SQL interface. The front-end and back-end distinctions depart slightly from the traditional understanding to include the HTTP protocol under the front-end so as to better delineate the work to our needs. Once the system has been put in place each of our team members will work towards shoring up the security until we are in consensus that all standard practices have been implemented. It is at this point that we will work together to uncover new vulnerabilities.

4 Intermediary Findings

The initial phase of our project has primarily been focused on researching SQL injection vulnerabilities, SQL databases, and preventative measures against attacks and exploitation of SQL databases.

4.1 SQL Vulnerabilities

Unvalidated user inputs allow attackers to manipulate queries and potentially gain unauthorized access to sensitive and confidential information. SQL injection attacks occur when an attacker injects malicious content or SQL queries through user input into an application. These attacks have the potential of extracting sensitive data, manipulating or deleting existing information, and in some instances execute commands to the operating system.[1]

Some common approaches of detecting SQL injection vulnerabilities include:

- Testing with boolean conditions such as `OR 1=1`. Exploits in logical conditions are common SQL injection techniques.
- Testing with a single quote ' character. A vulnerable application may exhibit errors when a single quote is passed in user input.
- Testing with SQL specific syntax. SQL syntax, when executed, should return the same value as the original input. If it evaluates to a different value, it may indicate a vulnerable system.[2]

4.2 MySQL

We have decided to employ MySQL for constructing our SQL database. MySQL is an open-source relational database management system. Databases are data repositories for software that securely store information, ensuring the data can be accessed in subsequent instances with convenience. A relational database stores data into distinct tables rather than consolidating all information in a single table. This approach is optimized for speed and efficiency.

MySQL is cross-platform compatible, making it so that development and deployment are flexible and well-suited for diverse computing environments. It is also scalable, accommodating applications ranging from small to large. MySQL is optimized for performance, achieved by utilizing query optimization and caching mechanisms, ensuring the efficiency of database operations. It also incorporates security features to protect the confidentiality of data through features such as data encryption and user authentication protocols. Moreover, MySQL supports a plethora of programming languages, including C, Python, C#, and Java. This versatility will enable individuals to

contribute effectively to the project despite having different programming backgrounds.[3]

4.3 Prevention Strategies

In obstructing potential exploits against our SQL database, we have chosen two preventive techniques: input validation and the incorporation of parameterized queries. The implementation of these strategies is crucial for ensuring the robustness of the security of our database and the potential mitigation of risks from SQL injection attacks.

4.3.1 Input Validation

Input validation is an essential procedure. Its main objective is to stop corrupted data from entering the database and potentially causing problems with components that function at a deeper level. It is strongly advised to stop possible attacks as soon as user requests are processed in order to improve security. Before any data is processed further by the program, input validation is a useful technique for identifying illegal input.

Concerning validation strategies, both syntactic and semantic input validation are required. Ensuring the proper syntax of structured input fields is the main goal of syntactic validation; it is necessary to verify that phone and social security numbers are formatted correctly. Semantic validation, on the other hand, makes sure that values are accurate within the framework of the particular business needs. Examples of this include confirming that a price is within the expected range or that the start date comes before the end date.[4]

4.3.2 Parameterized Queries

Parameterized queries are a security measure utilized to prevent SQL injection attacks in databases. In parameterized queries, parameters are utilized to denote user input instead of directly incorporating user input into query strings as in traditional SQL queries[5], and parameter values are supplied during execution.[6] By avoiding the direct incorporation of user input into the query, this allows for enhanced security and reduces risks of malicious attacks. This enables the database to distinguish between code and data, ensuring that the purpose of a query remains unchanged even if SQL commands are inserted by an attacker.[7] Furthermore, parameterized queries offer the

advantage of being reusable and adaptable with dynamic data, where values are unknown until the statement is executed.[8]

5 Group Member Roles

Project Lead: Lasair Servilla
Front-end Developers: Emely Seheon, Victoria Lien
Back-end Developers: Bryce Palmer, Kelsey Knowlson
Full-stack Developers: Ester Aguilera, Jingru Dou

*All team members will work on research related to best practices and SQL injection.

6 Expected Results

The successful implementation of this project is anticipated to generate substantial insights into enhancing the security of SQL databases through an amalgamation of rigorous research and practical implementations. The project will delve deep into the intricate workings of SQL databases, particularly focusing on the vulnerabilities susceptible to exploitation, notably by SQL injection attacks.

Drawing upon the findings from research, the project aims to develop a robust security framework tailored to mitigate the risks posed by SQL injection attacks. This framework will encompass multifaceted strategies, including but not limited to encryption techniques, access control mechanisms, input validation strategies, and parameterized queries.

Furthermore, the project will involve the development of a secure testing environment to assess the effectiveness of the implemented security measures. Ultimately, the expected outcome is a fortified defense mechanism that not only safeguards against exploitation but also preserves the integrity and confidentiality of data stored within the SQL database.

References

[1] k. OWASP. "Sql injection." (2024), [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection (visited on 02/25/2024).

[2] PortSwigger. "Sql injection." (2024), [Online]. Available: https://portswigger.net/web-security/sql-injection (visited on 02/25/2024).

[3] Oracle. "What is mysql?" (2024), [Online]. Available: https://www.oracle.com/mysql/what-is-mysql/ (visited on 02/25/2024).

[4] C. S. S. Team. "Input validation cheat sheet." (2024), [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (visited on 02/25/2024).

[5] L. VILEIKIS. "Parameterized queries in sql – a guide." (2023), [Online]. Available: https://www.dbvis.com/thetable/parameterized-queries-in-sql-a-guide/ (visited on 02/25/2024).

[6] SQL-Server-Team. "How and why to use parameterized queries." (2019), [Online]. Available: https://techcommunity.microsoft.com/t5/sql-server-blog/how-and-why-to-use-parameterized-queries/bap/383483 (visited on 02/25/2024).

[7] O. C. S. Series. "Sql injection prevention cheat sheet." (2024), [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (visited on 02/25/2024).

[8] G. D. Solutions. "Parameterized sql queries." (2024), [Online]. Available: https://www.ge.com/digital/documentation/historian/version72/c_parameterized_sql_queries.html#:~:text=The%20benefit%20of%20parameterized%20SQL,SQL%20queries%20for%20each%20case. (visited on 02/25/2024).