

Cybersecurity Update 2

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

Due: March 31, 2024

Table of Contents

| | | |
|----------|-------------------------------------|-----------|
| 1 | Problem Statement | 2 |
| 2 | Motivation | 2 |
| 3 | Methodology | 2 |
| 4 | Intermediary Findings | 5 |
| 4.1 | SQL Vulnerabilities | 5 |
| 4.2 | MySQL | 5 |
| 4.3 | Prevention Strategies | 6 |
| 4.3.1 | Input Validation | 6 |
| 4.3.2 | Parameterized Queries | 7 |
| 4.4 | Password Hashing in MySQL | 7 |
| 5 | Expected Results | 7 |
| 6 | Group Member Roles | 8 |
| 6.1 | Lasair Servilla | 8 |
| 6.2 | Emely Seheon | 9 |
| 6.3 | Victoria Lien | 10 |
| 6.4 | Bryce Palmer | 11 |
| 6.5 | Kelsey Knowlson | 11 |
| 6.6 | Ester Aguilera | 12 |
| 6.7 | Jingru Dou | 12 |
| | References | 13 |

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 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.

In order to test different ways a SQL database can be attacked we determined a ticket selling site would provide one of the best solutions. This is because there would be several components that could be affected by an attack. For example there are the tickets for an event themselves. They have price, location, and if the ticket has already been sold. On the other side is user accounts which would have things like encrypted passwords and a list of tickets associated with the account. Given time constraints we could even add a database for saved payment methods, however the above components would be more than enough for testing purposes.

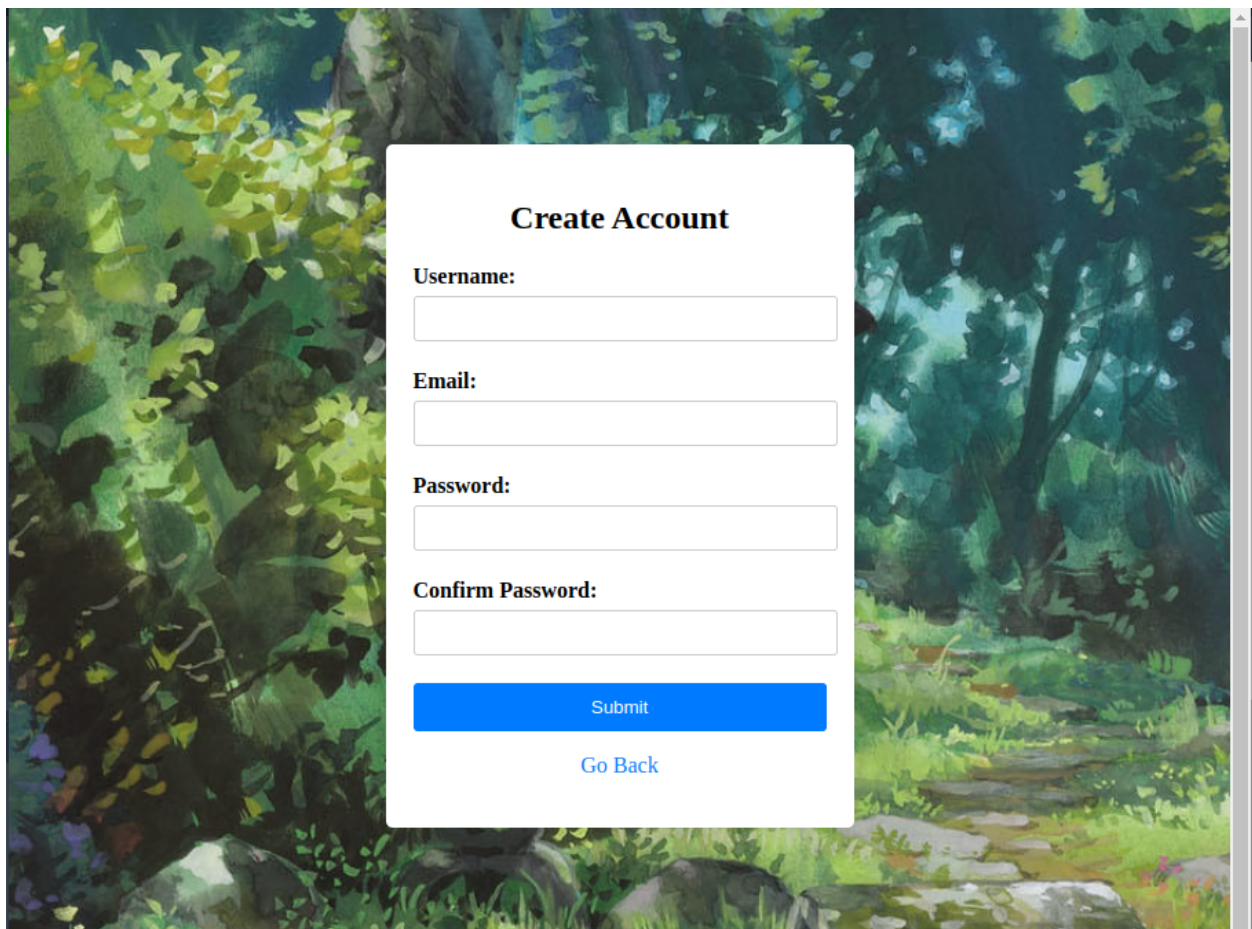
A screenshot of a web browser displaying a 'Create Account' form. The form is a white rectangular box centered on a background image of a lush green forest with sunlight filtering through the trees. The form contains the following elements: the title 'Create Account' in bold black text; a 'Username:' label followed by a text input field; an 'Email:' label followed by a text input field; a 'Password:' label followed by a text input field; a 'Confirm Password:' label followed by a text input field; a solid blue 'Submit' button; and a blue 'Go Back' link below the button.

Figure 1: User Account Creation Page

At this time a functional portion of the website has been set up. The func-

tionality includes a back-end database supported by MySQL and a front-end way for users to interact with the website/database. The functional section is currently confined to user account creation 1 and login system 2, however this is enough to begin testing SQL attacks. The back-end database for further user interactions has been created as well as a minimal version of the front-end 3. These two parts just need to be connected so that user searches actually send queries to the database. Security for the functional section is primarily done by parameterized queries and password hashing. Future work will also add input validation, as well as an unprotected version in order to demonstrate the severity of SQL injection attacks on an unprotected system.

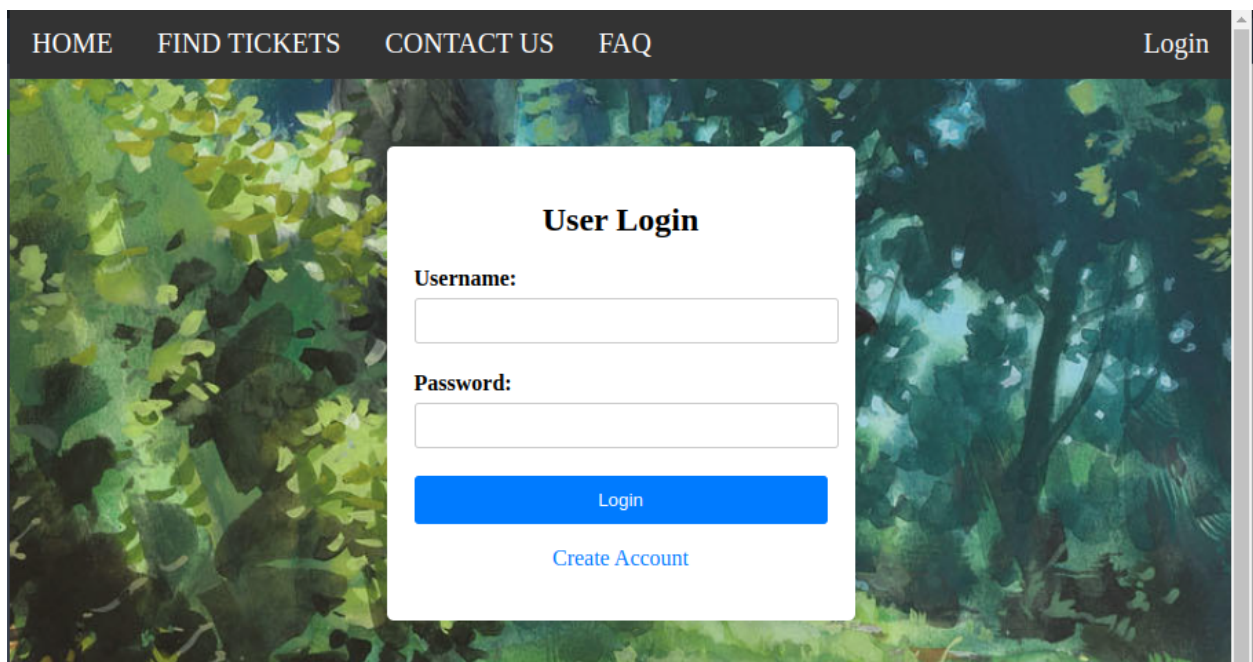


Figure 2: User Login Page

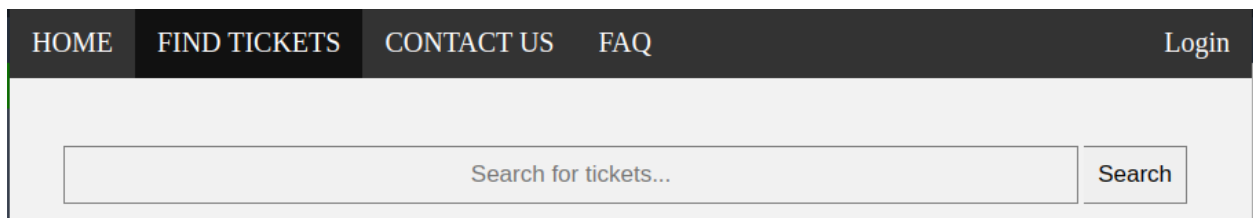


Figure 3: Search Bar for Ticket Database

4 Intermediary Findings

The current 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 alongside with creating a web service and an associated database that can demonstrate such techniques. As the development of the web service is still in progress, an emphasis on research has been crucial. We have examined various aspects of SQL injection, hoping to create a robust and resilient system that safeguards against security threats.

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

Validating input is a necessary step. Preventing erroneous data from accessing the database and maybe causing issues with deeper-level components is its main goal. Improving security necessitates thwarting potential attacks as soon as user requests are handled. Input validation is a helpful method for detecting unauthorized input before any data is further processed by the software.

Syntactic and semantic input validation are necessary when it comes to validation procedures. The primary objective of syntactic validation is to ensure that structured input fields have the correct syntax; phone numbers and social security numbers must be verified to be formatted appropriately. Conversely, semantic validation verifies that values are correct within the parameters of the specific business requirements. Verifying that a price falls inside the an-

anticipated range or that the start date occurs before the end date are two examples of this.[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]

4.4 Password Hashing in MySQL

Password hashing is an essential step for securely storing user credentials. Since MySQL version 8.0, passwords may be stored in a hashed format using the SHA-256 algorithm with a plugin. This plugin allows for enhanced security as passwords are stored in a nonreversible format, making it difficult for unauthorized users to decipher passwords even if the database is compromised.[9]

5 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 sample concert ticket purchasing website with 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.

6 Group Member Roles

6.1 Lasair Servilla

Areas of contribution:

- **Written portion of update 2:** For the written portion of update 2 I worked on the methodology section as well as my own contribution section.
- **MySQL setup:** I determined what MySQL service we were going to use for this project and ran through the initial set up to get it working locally on my computer. After I guided my team and informed them on what they would need to do in order to set it up for themselves.
- **GitHub:** Created the GitHub repository for the project and shared it with team members.
- **Website initial setup:** I created the base flask site we were going to use for the SQL injection attacks. This includes creating the file 'main.py' with a empty home page and setting up the file structure so that flask is able to correctly find any HTML files and image files needed for the site.
- **Website connection to database:** With the basic site set up I added functionality to connect to the MySQL database. As a part of this I created CSV files that would remain static and could be used to create the tables within the database each time the user returned to the landing page of our site. I set it up in this way so that any damage done

to the MySQL database during our attacks could be easily reversed. The tables set up at this time were the users and events.

- **Website user account creation:** The next contribution I made to the site was adding in user account creation functionality. This included the front-end page that the user sees when creating an account as well as the back-end that connects to and edits the database accordingly. At this point it only had parameterize user input and password hashing for safety measures as adding other measures fell to another team member.
- **Website user login:** As a part of the user system I also added the login system. This again includes both the front-end page that the user sees and the back-end work that connects to and checks the database. This also includes just parameterized user input as a starting point.
- **Troubleshooting team questions:** Given I am the only team member with extensive experience in this area I tried to answer questions that came up during other team members contributions. Including but not limited to explaining the structure of the project setup, to helping with issues connecting to the MySQL database.
- **Setting up team goals:** I played a strong role in setting up team goals and determining what the final site would look like. As mentioned I have a history working on this type of project, so it is due to this that I did so. It was agreed on as a group that I would act as a kind of leader in the technical aspects of this project.

I would like to add that a lot of team members spent a good portion of their time up to this point working on catching up their knowledge in order to contribute to the physical project. I volunteered to do a lot of the work in the beginning with the understanding that their quantifiable contributions would come largely after this update is due.

6.2 Emely Seheon

Areas of contribution:

- **Researched MySQL** I conducted a deep dive into MySQL, exploring its functionalities and intricacies to gain a comprehensive understanding. This involved not only grasping its core mechanics but also de-

vising optimal strategies for database structuring and query execution, ensuring efficiency and scalability in our projects.

- **Expanded on Expected Results** Elevated the project's anticipated outcomes by refining and detailing our expected results. With more detailed knowledge of the scope of our project, I created a more concise set of expected results.
- **Gathered Information on SQL Injection and Similar Attacks on Web Servers** Undertook a rigorous exploration of security vulnerabilities, particularly focusing on SQL injection and other potential threats to web servers. This involved extensive research to comprehend the nuances of SQL injection attacks, as well as similar exploits like the notorious Heartbleed vulnerability. By acquiring this knowledge, I fortified our defenses and equipped the team with crucial insights to preempt and mitigate such risks effectively.
- **Learned and Understood Web Design Practices** Learned the fundamental aspects of web development practices, which includes the relationship between frontend and backend technologies. This encompassed a broad spectrum of tools and languages, including HTML, JavaScript, Python, PostgreSQL, Flask, and more. By assimilating these fundamental concepts, I fortified our capacity to create robust, dynamic web solutions that seamlessly integrate frontend user experiences with backend functionality.

6.3 Victoria Lien

Areas of contribution:

- Contributed to the "Intermediary Findings" section of update 2 as well as my own subsection in "Contributions".
- The majority of this update's time was spent on research, particularly in the areas of SQL injection vulnerabilities, SQL databases, and prevention strategies against SQL injection attacks. To write the "Intermediary Findings" section, I researched common SQL vulnerabilities, including approaches of detecting SQL injection vulnerabilities, MySQL (as this was the database we chose to use in the project), prevention strategies against attacks, such as input validation and parameterized

queries. I also did a bit of research on password hashing in MySQL due to not wanting to store plaintext passwords in the database and to add an additional layer of security.

- I spent time researching HTTP requests in Python as this was an area of the code I was originally tasked with; my experience with HTTP requests was previously limited to ASP.NET and I did not have experience with Python-related requests.

6.4 Bryce Palmer

6.5 Kelsey Knowlson

My contribution was that I spent a fair amount of time researching how to create a secure user interface that will not be vulnerable to SQL injection. I also looked at the possible ramifications of not having these security measures and what that would look like in code. Additionally, since I have no experience with Javascript or CSS, I needed to do a number of online courses and learning to be able to help build the user interface and the website. I researched about MySQL, the SQL database we picked, and I looked into how the code would function with the server and host. I did some error checking with group mates to make sure that we could all access the MySQL database when they logged in since there was a slight issue initially with connecting the site to MySQL due to password issues.

Lastly, I researched the ways that one could break or get unauthorized access to the SQL database with SQL injection. I looked into the code that could be used on input lines, which entailed many hours of learning SQL and its functions since that is necessary to know if I want to use injections on the site. Overall, at this stage of the project I spent researching, learning, and formulating a plan to contribute for my section which is to set up a safe and unsafe mode where I would fully protect the site on the safe mode and on the unsafe mode, I would leave it open to attack. Since I was solely in charge of adding security, I had a large amount of work, but it wasn't just back end since I needed to have a way for the user to switch between safe and unsafe mode, which means changing and updating the user interface and connecting those changes with the back-end security code that I was setting up.

I also just want to comment that even though much of my code was not fully actualized at this stage, I still was working very hard and contributing to the final project output. I hope that I accurately conveyed in my previous comment.

6.6 Ester Aguilera

Areas of contribution:

- **Researched and learned basics of SQL injection vulnerability** through the use of research papers and cybersecurity web articles.
- **Learned basics of web development** through free online tutorials, concentrating primarily on the functionality of website user interfaces by learning HTML, CSS, and JavaScript, with a secondary focus on server-side components.
- **Coordinated team meetings, documented agreed-upon goals and tasks, and initiated discussions** on task delegation.
- **Wrote motivation portion and my contributions** for Update 2.

6.7 Jingru Dou

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).
- [9] M. Lord. “Protecting mysql passwords with the sha256_password plugin.” (2015), [Online]. Available: https://dev.mysql.com/blog-archive/protecting-mysql-passwords-with-the-sha256_password-plugin/ (visited on 03/31/2024).