Yongye Tan

11/26/2022

Introduction to Network Security

SQL Injection Exacerbate Data Leakage

My research topic focuses on SQL injection exacerbating data leakage and how it connects to human over-reliance on technology use. With the emergence of technology, people increasingly store their private information on the Internet such as their credit balance on bank applications or profile information at various shopping sites. To access their privacy, the server must use a form of authentication to verify the identity of the user. This process usually involves a tuple of usernames and passwords to log in to their accounts. A text field is provided when a user types their information on the keyboard, then later reflects on the screen. After the user presses "submit," their information will be verified by the company server, which uses the database that stores the user's information to authenticate. During this process, a database manipulation language called Structured Query Language (SQL) is used to retrieve the information from the database. This place is where data leakage and alteration might occur without proper care in structuring the query based on the username and password. That being said, malicious hackers can insert a special query into the existing query to retrieve the data, and this attack is called SQL injection. In fact, SQL injection is the most common and easiest attack for over 50% of all web application attacks across the World Wide Web in the modern century (*Ninja Blog*) and ranks as one of the top ten most severe data leakage attack vulnerabilities in the Open Web Application Security Project (OWASP).

How does SQL exactly allow malicious hackers to insert the special query to access the database based solely on the username and password? To answer this question, we must first understand some simple SQL statements, executions, logic, and the difference between types of Database Management Systems, also known as DBMS. A query is a request for data. The simplest SQL query can be:

select * from database

The keyword “SELECT” is the action verb that serves the purpose of pulling data. The special “*” symbolizes “every entry.” The last part specifies the location of where the data is pulled from. The query can be added with another conditional statement using the keyword “where” to add an extra filter to the existing statement; for example,

“Select * from database where username = “Tom”

this query selects all information when there is a match that the username is equal to Tom. There are crucial action verbs like “INSERT,” “DELETE”, and “UPDATE”, which serve the purpose of inserting brand new data, erasing the existing data, and updating the existing data from the database. These four words are the fundamentals of building a functional database management system (*IBM*). A database management system (DMBS) is a mechanical system that manipulates and stores data for any organization such as schools, corporations, hospitals, or individual use. DBMS are categorized according to their type of data structures. Three basic types of DBMS data models spread on the Internet: Relational, Network, and Hierarchical (*HEAVY.AI*). In relational data models, data are organized as a group of separate tables with columns and rows. In a non-relational data model, data is organized as JSON data (*Morris*), where each data entry has a key pair with a value. In a network data model , more than one file can be connected to more than one file, similar to a spider-web structure, where one of the files can influence other files. In a hierarchical data model, data is organized as a tree-like structure, where data are stored as nodes and connected through memory addresses (*HEAVY.AI*). Each node has various branches connecting to other nodes. After explaining how SQL works and the data model SQL supports, we can discuss the SQL injection that any hacker can use to manipulate the database. There are two main types of SQL injection: Error Based and Blind Based (*Indusface*). The most common is error-based SQL injection. It involves inserting malicious queries into the text fields to retrieve error-based information about SQL syntax, and server version. In the article, “Types of SQL injection,” a single quote or double quote is added at the end of the query on the PasswordManager platform, which invalidates the logic. An example would be:

"select * from database “

When the backend logic tried to run this query, an alert window would pop up and tell the user there is an SQL error. The user can use the request in a txt file and sqlmap – sqlmap would try different payloads – to get information about which database is used: Apache 2.2.29, PHP 7.2.3, and MySQL >=5.0.12 (Prabowo). This exploitation technique is easiest since it simply involves a single symbol at any place of the query. Blind Based SQL injection categorizes into two types: boolean-based and timed-based SQL. Blind SQL is a special attack that asks the database question and understands what the database does by analyzing the response. First, Boolean is a simple logical statement, which is true or false, the same as saying whether something is correct or incorrect. Using the simplest example would be

select * from database where username = “Jimmy” and password = “apple3” or 1 ==1;

There are two logical statements in this query, which are “username… password…” and “or 1==1.” When the user inputs their username and password, the system would use the user’s input to verify with the stored user’s input that the user had signed up before. The first part of the logical statement might evaluate to false as either the username or password is not correct, however, the second part of the logical statement will evaluate to true due to the fact that 1 is indeed equal to 1 – quick reminder: false or true will always evaluate to true. That being said, no matter if the input is correct or incorrect, the system will always be able to access the database regardless. Secondly, time-based SQL happens when the attacker inserts an SQL query causing the database to delay for a certain amount of time to respond to the query. If slowing down the server to respond works successfully, that is an indication that SQL injection is possible. The following query is an example of a time-based attack to verify the database version (Beagle Security):

SELECT * FROM USER WHERE id=1-if(mid(version(),1,1) = '5', SLEEP(20), 0)

It keeps delaying the execution time of the query and continues it repeatedly when the program time is the same as the parameter time every 20 seconds. The time basis attack is inferential because even though it cannot retrieve or alter the user’s data, it can help hackers predict what version or type of database server it uses, as the SLEEP() function is only available in a certain database, which is MySQL 5, a relational database management system that uses SQL.

Numerous companies and organizations face exploitation vulnerabilities under the attack of SQL injection each year (*"Latest SQL injection security news | The Daily Swig"*). One of the most recent security leaks about SQL injection happened in March 2022, it was Moodle, an education site for young students to post and review their assignments (*Haworth*). If a student logins in as a teacher, a SQL injection can be performed by creating a badge for the student hacking in. A badge is like a reward for students to show their achievements and accomplishments. Each time, a badge is created means a new SQL query is created. This creation leads to all the students' private information being accessed through the account of the hacker pretending to be a teacher. This is a serious potential threat to the school administration and students' privacy. If this private information is exposed to the public, third parties will potentially sell and advertise them, which will create chaos in society. Parents establish reliable relationships with their student's schools and trust that their lovely children's data will be secured somewhere in good hands. That being said, this is considered a way of challenging the education authorities and trust between institutions and parents. Another example of data leakage in a more notable company is HP, a computer hardware company, Device Manager exploit (*Bannister*). The founder of Cognitous Cyber Security, Nick Bloor, wanted to test the security of the HP Device Manager. He performed deserialization attacks against it during a network security assessment. They found the backdoor with superuser privilege that can allow them to access the Postgres database. The attack was performed with full brute force and simple logic. Then, he used HQL injection to exhaustively test with the database, then gain full remote control on the server. On a quick note here, SQL is based on a relational baseball model whereas HQL is a combination of OOP with relational database concepts. These are notorious examples of SQL injection attacks, which should not be done by anyone. Not only does it perturb the internal system organization, but it also violates the three basic foundational principles in cybersecurity: CIA.

CIA stands for Confidentiality, Integrity, and Availability (Chai). These three elements are important to secure the pillar of information security, ensuring data goes to the correct place and people who access data are the correct people. Confidentiality is the concealment of information and resources.

The purpose of this is to ensure that information is secure and safe. SQL injection violates this rule as it provides a backdoor to navigate to the server that they want access to. This can be prevented by granting specific read permission to whoever tries to access it – like how you do not want anyone to know how much money you have in the bank. Using boolean-based SQL, any hacker can easily bypass the authentication check. That being said, sensitive information from any unauthorized party should not be allowed or prohibited. Have you ever been afraid that your bank balance got altered one night without noticing? This is an instance of integrity, the trustworthiness of information or sources. We voluntarily give our privacy such as our address, and identification number to the bank and the bank uses them to create a bank account for us to store our deposits as a part of the balance. Our balance and personal data mustn't be altered by some third party or man-in-the-middle (*Imperva*), other than the true owner, which is the user. SQL language allows the backend to alter, delete, and update the user's data. Here is a simple example showing how integrity is violated:

1. INSERT INTO USER VALUES("balance", 2000.0);
2. '); UPDATE USER SET balance = 0.99 ; --
3. INSERT INTO user VALUES("balance", 2000.0); UPDATE USER SET balance = 0.99 ;-- ');

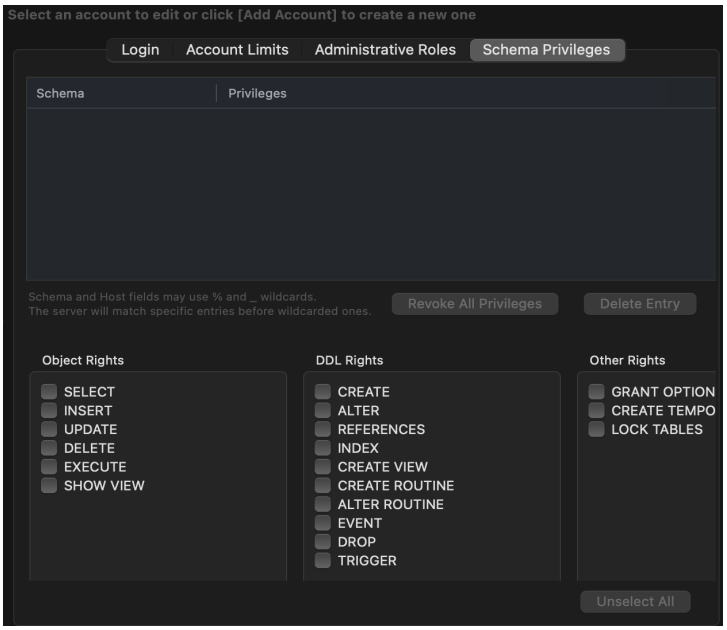
The first query creates a user with a balance with an initial deposit value of 2000, and the second query updates the user's balance to 99 cents. The third query combines both queries, which is a new form of SQL injection, inserting another SQL query after an existing query, causing the backend to execute two statements at a time. This combination allows the attack to be more robust as the hacker can try a myriad of other queries to test out whether they work. Any software developers should be very careful with the query design as a minor error can lead to even a loss of millions or billions of dollars. This further reinforced the understanding of why any company should need the set privilege to write permission to ensure the data is not overwritten by unknown parties or improper use. The last pillar of information security is availability, the ability to access information or resources. One of the simple examples that can best showcase this is: INSERT INTO user VALUES("balance", 2000.0); DELETE user; The backend creates a user with a balance then deletes the user on purpose. When the user tries to log in to their

account, not only can they not see their balance, the database does not have any evidence that the user exists due to the previous deletion. In other words, the user lost the ability to access things that belong to them. This pillar is the most dangerous threat among the three in my opinion. If the hacker performs the time-based SQL attack, which is delaying the server's response by whatever seconds they want, then they can put several or even all the servers to sleep. That being said, if none of the servers are available to the public, citizens will complain about the service, which leads to negative customer retention, and produce inconvenience and chaos for a lot of microservices and other companies who use the server. After explaining the critical section of the CIA, we understand how deadlier and more serious SQL injection is. But how exactly or what defense technique can we use to prevent such attacks from happening repeatedly? We will talk about that in the next paragraph with more details about the Parameterized query and prepared statement.

Preventing SQL injection vulnerabilities is not as easy as we thought. It requires a high level of technique and programming skills to prevent edge cases. Assurances are built off of the carefulness and skills of software developers as they are the ones who build the application we use in public. The First step, every Software Engineer, Quality Assurance, System Administrator, and other IT staff should receive proper suitable security training to fully understand SQL injection and techniques (*Acunetix*). In simple words, this step is to bring and increase awareness of security leakage and build strategies against such attacks. The second step is to prepare a query. Every query should be well-designed and have a purpose for what to do. This step requires the design and implementation aspect, as a query most likely is going to use more than one table and multiple fields of a table. Keep in mind that no user input should be trusted, and any user input could potentially risk data leakage through SQL. Step 3 is the most important step, which is to create prepared statements, which is a parametrized and reusable SQL query that forces the software engineer to write the SQL query and the user input separately. These three critical steps are crucial for every secure application, hence, every developer should follow them.

How do we build a successful and versatile prepared statement and how many steps does it require to build it? Six careful steps are required from the software engineer and whatever the DBMS is

being used before building it (*"How to prevent SQL Injection vulnerabilities: How Prepared Statements Work"*). The first step is called parsing, which involves the breaking down of SQL queries. Each query constitutes too many words, so misspellings and any grammatical or syntax errors must be evaluated for the sanity and validity check. The second step is called semantics check. The DBMS will check if the column or table exists for the query and whether the user has the read or write privilege to execute this query. This requires pre-set-up and configuration before the user can run the query. In MySQL workbench, "administrative roles" and "Schema privileges" are the two places to perform the semantic check (*"MySQL Workbench"*), from which the administrator can have the options to set permission for insert, execute, show view, and so on. A picture of the MySQL workbench showing privileges is shown below:



The third step is called binding, where the query is converted into binary code, compiled, and then sent to the server for performing optimization. Step four is optimizing the query, which requires lots of logic and algorithms to make it fast and efficient. Using the keyword "explain" would tell us how the SQL database executes a query, which gives the developer a deep understanding of how they should structure their code, for example, **"explain select * from DB"**. The fourth step is a time-saving step, which is called cache, saving the most efficient algorithm in the cache, and is used the next time when the same query is

executed, in other words, the prior four steps are skipped immediately. The last step is execution. Here is a walkthrough of how preparing a statement works in Java (*"How to prevent SQL Injection vulnerabilities: How Prepared Statements Work*"):

1. Creating a query: String query = "Insert into Users ("user_id", "first name," "email") Values (?,?,?)";
    - Creating a query with "?" indicates a placeholder for each value
2. PreparedStatemenBlind t ps = connection.prepareStatement(query);
    - Declaring special data types for preparing for the query,
3. Inserting values into each placeholder and to ensure no SQL injection,
    - ps.setString(1, the user.getUID()), ps.setString(2, user.getFname()), ps.setString(3, user.getEmail())
    - The number shows the position of the placeholders where they are located at

The Prepared Statement fixed all the issues for SQL injection. Let's analyze it with the types of SQL injection we had talked about. Any error-based injection would fail when we construct a prepared statement with a query. Not only will it fail, but it also throws a proper exception to the backend if the developer includes a try-catch clause. For boolean-based SQL, when we construct the query with two desired user inputs, the prepared statement will allocate slots for only two inputs for positions 1 and 2. When the hacker tried to trick the database with the third logical statement "or," it would immediately fail the logic of the prepared query as it does not accept more than two queries. In short, Prepared Statements are the best security to modularize each input and allow the whole query to be legible, ensuring the DBMB performs a quick semantic check and returns the correct output to the users' hand.

Knowing how to protect our data security is essential. We have gone through from understanding SQL injection, the types, and the real-life examples, to techniques and strategies to prevent such attacks. Securing our date in a safe place, and having a way not to let anyone else touch or view our privacy establishes the trust relationship between the client and the platform. Everyone lives in the world of the Internet, therefore information is easily accessible. Data sharing is ubiquitous as well as data accessing

and CIA plays a critical role in the world of Information Security. Constructing an invisible “firewall” such as Prepared Statement helps society at large to be a safer zone and mitigate data leakage as much as possible. The best we can do is to eliminate each data leakage and constantly look for ways to improve our defense methods. My final thought to the end of this research paper is: Are there more ways other than Prepared Statement that we can build to prevent SQL injection?

Works Cited

Acunetix. “What is SQL Injection (SQLi) and How to Prevent Attacks.” *Acunetix*, https://www.acunetix.com/websitesecurity/sql-injection/. Accessed 26 November 2022.

Bannister, Adam. “HP Device Manager exploit gave attackers full control over thin client servers.” *PortSwigger*, 12 October 2020, https://portswigger.net/daily-swig/hp-device-manager-exploit-gave-attackers-full-control-over-thin-client-servers. Accessed 26 November 2022.

Beagle Security. “Time based Blind SQL Injection (SQLi).” *Beagle Security*, 5 June 2018, https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html. Accessed 26 November 2022.

Haworth, Jessica. “SQL injection vulnerability in e-learning platform Moodle could enable database takeover.” *PortSwigger*, 10 March 2022, https://portswigger.net/daily-swig/sql-injection-vulnerability-in-e-learning-platform-moodle-could-enable-database-takeover. Accessed 26 November 2022.

HEAVY.AI. “What is DBMS? Definition and FAQs.” *HEAVY.AI*, https://www.heavy.ai/technical-glossary/dbms. Accessed 26 November 2022.

“How to prevent SQL Injection vulnerabilities: How Prepared Statements Work.” *Security Journey*, https://www.securityjourney.com/post/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work. Accessed 26 November 2022.

IBM. “What is a database management system?” *IBM*, https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system. Accessed 26 November 2022.

Imperva. “What is MITM (Man in the Middle) Attack.” *Imperva*, https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/. Accessed 26 November 2022.

Indusface. “Types of SQL Injection.” *Indusface*, https://www.indusface.com/blog/types-of-sql-injection/. Accessed 26 November 2022.

“Latest SQL injection security news | The Daily Swig.” *PortSwigger*, https://portswigger.net/daily-swig/sql-injection. Accessed 26 November 2022.

Morris, Jeff. “Why NoSQL JSON Databases Are So Useful.” *Couchbase*, 14 June 2021, https://www.couchbase.com/blog/json-database/. Accessed 26 November 2022.

“MySQL Workbench.” *MySQL*, https://www.mysql.com/products/workbench/. Accessed 26 November 2022.

Ninja Blog. “Full-Stack Server Protection.” *BitNinja - Full-Stack Server Protection*, 18 August 2021, https://bitninja.com/blog/the-most-common-types-of-cyberattacks-4-sql-injection-attacks/. Accessed 26 November 2022.

Techopedia.com. “What is a Network Database? - Definition from Techopedia.” *Techopedia*, https://www.techopedia.com/definition/20971/network-database. Accessed 26 November 2022.

Chai, Wesley. “What is the CIA Triad? Definition, Explanation, Examples.” *TechTarget*, https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA. Accessed 26 November 2022.

Prabowo, Kresno Murti. “2019.” *Lebah Baru*, 13 December 2019, https://kresnomurtiprabowo91.blogspot.com/2019/. Accessed 26 November 2022.