A Study on SQL Injection Attacks

Mohammed Mustafa Khan

Abstract

Some web applications have inherent vulnerabilities that allow hackers to get unauthorized access to confidential and private data in an organization. SQL (structured query language) injection attacks remain to be a serious threat to web applications. It is one of the most used mechanisms by cyberattacks to steal or compromise sensitive data stored in relational databases. Some web application developers design systems with bugs or accidentally leave existing gaps by not screening the user's input for some unique symbols and characters that exist within the structural query statements or failing to validate the quality of information to find out if it is numerical or text that triggers the unprecedented outcome of its implementation. Attackers seem to have learned the existing web application's vulnerabilities correctly and seize to utilize these vulnerabilities to their advantage. SQL injection attacks comprise the insertion of a SQL command or query by using the input data from the client side that cascade to the application. It is conducted by injecting a malicious program in the SQL statement that modifies or alters the data by updating, inserting, or deleting. Additionally, superior SQL queries can initialize the administrative operations of the database, such as shutting down the database management system (DBMS), retrieving the content of a specific file existing in the DBMS, and issuing powerful commands to manipulate the operating system. Similarly, the SQL command can manipulate the cookies stored in a web application's database, allowing for actions such as retrieving, updating, or deleting cookie-related data, which could potentially be used for session management, user authentication, or tracking purposes.

Keywords: SQL injection, database, web application, SQL queries, vulnerabilities, detection, prevention.

1.0 Introduction

A sheer amount of companies' data is stored in particular database servers. Protecting these data from the eyes of intruders is a fundamental aspect that all organizations strive to achieve. Any data breach incidents will have detrimental effects on users' privacy as well as damage their reputation and may result in financial losses [12]. Databases have become essential components of modern organizations and, therefore, must be highly protected to ensure no data loss or leakage events.

To access and manipulate these databases, it utilizes the standard language known as SQL. Database administrators use SQL to carry out various operations on the data, such as storing and performing the CRUD (create, read, update, delete) functions. Databases store data in table format and have relations between each other. The relationship between data points is correctly stated, like the table's relationship with the field types, which can be defined in a schema that enables easy search between the relationships. The type of these databases is known as relational database (RDB). Examples of RDB include MySQL, Oracle, and Microsoft SQL server, among others [12]. Relational databases are more prone to SQL injection attacks than non-relation databases like MongoDB.

Most web applications utilize backend relational databases to store, transmit, and retrieve data. Web applications have become the prime target for attackers aiming to gain access to sensitive data for personal interest. Attackers employ various techniques and tactics to exploit to exploit vulnerabilities inherent in database programs. One of the widely used methods to exploit the databases is the SQL injections. This

research paper will walk you through the rationale attackers use SQL injections, the working principle of an SQL injection, an example of an SQL injection, and discuss the types of SQL injection attacks, as well as the detection and prevention of SQL injection attacks.

2.0 The Prominence of SQL Injection Attacks

SQL injection is used by attackers in an attempt to intelligently craft an SQL command into an input field rather than the predictable information. The rationale for this is to guarantee attackers a secure response from the database that will allow them to generate the construction of databases such as the schema contained in the databases. Once the SQL injection attack is completed successfully, it has the potential to damage or compromise information stored in respective databases. The popularity of SQL injection manifests a lot with the ASP and PHP applications [10]. These applications still utilize older functional interfaces. The ASP.NET and J2EE applications are unlikely or less prone to SQL injections since they utilize the features of existing programmatic interfaces. The devastating consequences of SQL injection attacks can be extreme. This extremity is limited by the skill and imagination of the attacker and, to some extent, defense-in-depth countermeasures such as short privilege links to the database server.

3.0 The Working Principle of SQL Injection

SQL is a query language meant to manipulate data stored in a functional database. SQL queries are employed to carry out the CRUD functionalities and data retrieval. Different SQL essentials perform these tasks. For instance, the SELECT statement is used to query the database to recover data via user-offered strictures. The attacker needs to identify the loophole that exists in the user inputs in the web page or web application to successfully execute an SQL injection attack. The attacker proceeds to exploit by finding the unique identities of other users contained in the database [1]. After spoofing the users' identifiers, the affected users are impersonated by attackers. The impersonation of users is mainly aimed at database administrators since they are assumed to have database privileges to interact with data.

The web application or web page containing SQL injection flaws exploits the user's input by generating input content. The kind of content generated is known as "malicious payload," and it is the key aspect that is used to launch an attack [2]. Once the attacker sends the malicious payload to the database programs, the content is executed successfully, leading to an SQL injection attack. The SQL query enables an individual to select and retrieve data from the database. The SQL injection vulnerability may allow the intruder to gain full access to all the data in the database server. The SQL operates in a manner that permits an individual to alter data in the database server by either modifying, changing, or inserting new ones. This is the most interesting point that attacker intelligently utilizes to wreak havoc. For instance, an attacker can utilize SQL injection in a financial application to create some transactions null and void, modify balances, or transfer money from the user's account to another account.

4.0 Example of an SQL Injection

There are various SQL injection attacks, vulnerabilities, and procedures that can happen under different conditions. An attacker who aims to conduct an SQL injection capitalizes on a standard SQL query to perform unapproved data manipulation within the database. SQL injection can be performed in various ways. Some common SQL injections example entails:

- The retrieval of hidden data happens as a result of modifying an SQL query to recover further outcomes [13].
- Altering the application logic that makes the attacker subvert the query to jeopardize the application logic.
- Database examination makes it possible for the removal of information pertaining to the architecture and version of the database [13].

- UNION attacks involve the attacker recovering data from multiple database tables.

For the SQL injection example, we can inspect two database tables, including Users and Contacts. The users' tables may contain three fields. The field content of the user table can comprise of User ID, Username, and password. On the other hand, the contact table may have as many fields as shown. Note: this is used for demonstration purposes only to properly understand SQL injection examples. The data used here is mocked data.

User Table Fields		
UserID	Username	Password
7846	jondoe	P@ssw0rd

Contact Table Fields						
UserID	FirstName	Lastname	Address	Email	SecurityCode	CreditCardInformation
7846	john	doe	Boston	doe@yahoo	cat	P98642T

The user table has the login information.

Welcome to the Login Page	
UserName	jondoe
Password	P@ssw0rd

The database always stores passwords in a hashed format to increase password security. It is crucial to refrain from using cleartext to avoid passwords from being compromised. The user needs to input user credentials such as usernames and passwords to log in to the page. The information entered by the user is transmitted to the website's server, which constructs an SQL query that is later parsed to the database server. The structure of the query is shown.

Select ID from Users Where username = “johndoe” and password = “P@ssw0rd”

SQL works by assessing the query requests from each of the rows based on the false or true comparison [11]. The above example demonstrates that in every row where the username is john doe, and the password is P@ssw0rd, the user table is checked and responds with the ID value. The website's server recognizes the response that has been given to the database server. The website server will receive a 1, which means the information is true, and the page will be logged in.

In case we opt to get malicious with the query, it is possible to convince the server to believe that we have authentication, owing to the fact that the database server performs 1 or 0 checks. This can be attained by adding an OR to our password. In case we log in with y' or b=b as our password, it is possible to craft a new SQL query as shown:

Select ID from Users Where username = “johndoe” and password = “y” OR b=b

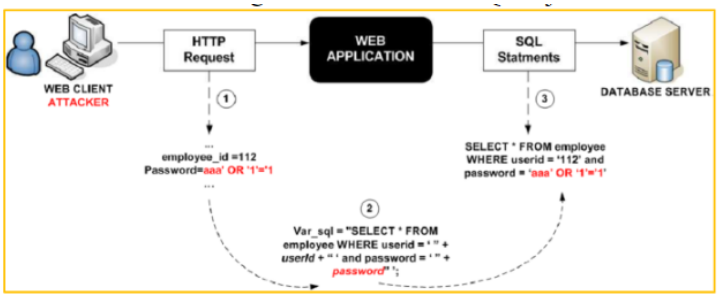
Eventually, the second option can be automatically validated by the database server even though y is not the password for johndoe. The database server inspects the alternative if y is not the password for johndoe and b is equal. The ID will be sent back to the application, and the user will be issued a successful authentication [11]. Additionally, the condition may not only be b=b condition. After equaling the two values, this command may work. It is possible to have c=c, 2=2, or even5628=5628.

If the webpage has the capacity to display data, it might be able to print other data on the screen as well. To collect the data, we can try combining two SQL requests simultaneously. Amalgamating two statements can be conducted by using the UNION SELECT command and adding to the initial command [11].

Select ID from Users Where username = "johndoe" and password = "y" OR b=b UNION SELECT lastName, securityCode From Contacts and creditCardInformation.

We can use many additional clauses to retrieve more data. The objective of an SQL injection attack is to gain access to the company's data.

Diagram showing an Example of SQL injection



5.0 Common Mechanisms for SQL Injection

User input injection. The most common type of attack involves a user crafting suitable user input and injecting an attack.

Injection via cookies. Cookies are files stored on the client side that contain user information and are automatically generated by web applications. Cookies restore the client information upon visiting the web applications. The intruder can get access to the cookie and embed a malicious code that will result in the insertion of an SQL injection attack [9].

Server variables injection. Server variables comprise network headers like HTTP and environment variables. Web applications utilize the aforementioned variables in different ways, such as discovering browsing patterns and recording metrics like usage statistics [9]. SQLi vulnerability may be created if the recorded variables are stored without sanitization.

Second-order injection. Also known as stored injections. They are the most complex and very challenging attacks to discover. The attacker appends malicious inputs into the database application to indirectly invoke an SQL injection attack when the input is utilized in the future. When the malicious input manages to reach the database, the attack starts manifesting [9].

6.0 Types of SQL Injection

There are various types of SQL injection (SQLi). The baseline of the SQLi is for the attacker to introduce random SQL into the database server of the web application. The simplest SQLi technique is through user input. Forms are the means by which web applications receive input from the users. The forms, which are frontend-based, transmit the user input to the backplane containing the database to process the request.

6.1 In-band SQLi

The attacker utilizes a similar communication channel to trigger their attacks and obtain their outcomes. In-band SQLi attacks are easy to perform and are one of the most popular types of SQLi attacks [3]. It is subdivided into two subcategories: error-based and union-based SQLi.

6.1.1 Error-based SQLi

This is an in-band SQL injection technique in which an attacker performs activities that result in error messages. The objective of the attacker is to invoke the database to display the error messages on the screen so that they can discover the architecture and construction of the database [3]. Programmers used these errors during the development phase of the database application. Omitting to obstruct these error messages will aid attackers in understanding the construction of the database application and weaponize their tactics to attack the database. Programmers need to deactivate error messaging notifications on the production site.

6.1.2 Union-based SQLi

This type of attack utilizes the UNION operator that merges two or more results of various SELECT statements to obtain a single outcome that is later sent back in an HTTP response [3]. The data obtained from this response helps the attacker to make concrete decisions. This type of attack is clearly demonstrated in the later section of the example of SQL injection.

6.2 Blind or Inferential SQLi

This kind of SQL injection attack does not depend on error messages. However, it relies on the logical response pertaining to the database query. For instance, an attacker can attach a conditional statement to a user input that is intended to be a string like username and password and oversee if the query returns a valid outcome or not [3]. The attacker can try various techniques like trial-and-error methodology in inferring data values by altering the situation and observing the behavioral response. This type of attack is complicated and hard for an attacker to exploit. However, it is one of the deadliest attacks when the attacker manages to execute it. Classified into two subcategories that are Boolean and time-based.

6.2.1 Boolean or Content-based SQLi

This type of attack pushes an SQL command to the database, and the application generates the result forcefully. It utilizes logical operators like OR, AND, or NOT to change the Boolean value of the conditional statement [3]. Different outcomes may be generated based on whether the query is true or false. Additionally, the content within the HTTP response is changed or may remain unchanged based on the returned outcomes. Eventually, the attacker can evaluate if the message created is a true or false outcome.

6.2.2 Time-based SQLi

It is a blind approach that involves the attacker sending an SQL query to the database, causing the database to wait for some seconds before responding. Time-based SQLi utilizes time-consuming functions like WAITFOR or SLEEP to delay the database [3]. Once the injection is initiated, the attacker observes the response time of the query and deduces data values by changing the time parameter.

6.3 Out-of-band SQLi

The attacker can conduct this attack if specific features are enabled on the database server used by the web application. Thus, this type of attack is executed when the attacker can not utilize a similar communication channel to trigger the attack and collect information or when the server is slagging for some activities to be conducted. This technique utilizes the server's capabilities to make DNS or HTTP requests to send data to an attacker [3].

7.0 Methods of Detecting SQLi Attacks

7.1 Web Application Firewall

The web application firewall (WAF) can be installed and configured on the server or network perimeter to detect SQL injection attacks by monitoring and filtering incoming HTTP requests, analyzing patterns in the traffic, and blocking malicious queries that attempt to manipulate database inputs [4]. It is crucial to use multi-layered approaches that involve a conglomeration of detection tools and avoid relying only on the WAF. Deploying the WAF with other intrusion detection tools, both network-based and host-based, can fortify the security of database applications.

7.2 Black-box Testing

It involves the use of penetration testers like web crawlers to discover all points of weakness of web applications that can be exploited by SQL injection attacks. The web crawler can be given user credentials

for authentication purposes [6]. They then simulate various types of SQL injection attacks to detect what type of SQL injection can exploit the company's database application and implement appropriate countermeasures.

8.0 How to Prevent SQLi Attacks

Protecting web applications or web pages against SQL injection attacks involves a combination of various methods and countermeasures. Arguably, there is no one solution that solves everything in SQL injection attacks [7]. Protecting database applications entails a series of methods and practices that create a formidable solution that will deter attackers from bypassing or spoofing sensitive data.

8.1 Sanitization

If the application accepts an unanticipated query entered by attackers, it is crucial to restrict the input functionality to secure data. Developers can implement input validation or sanitization to enable the application to take in only specific inputs into the form fields and omit those forms that do not correspond to expected values [8]. For instance, when web users want to create passwords, they are prompted to follow password policy best practices like the length of characters and include at least one special character.

8.2 Use Parameterized Queries

One of the popular techniques used to protect the code from SQL injection is parameterizing the SQL queries. Parameterized queries is a method that involves separating the SQL query from the user input values. The technique enforces the user inputs to be sent as parameters. Passing the user inputs as parameters eliminates any underlying executable code because the parameter is handled as data or literal value, and the type and length are also checked. Additionally, separating query logic from user input and automatically escaping special characters [5].

8.3 Filtering and Validation

Apart from detecting SQL injection attacks, WAF can also be used to prevent them. WAF screens out SQLi and blocks possible threats. WAF operates by matching the inputs to an application against a list of identified signatures to countermeasure malicious SQL statements [4]. It is important to update the list and carry out regular patching to stay abreast of the attackers.

9.0 Conclusion

In conclusion, SQL injection attacks pose a significant threat to web applications and relational databases by exploiting vulnerabilities in user input validation. Attackers can manipulate SQL queries to gain unauthorized access to sensitive information, alter data, or perform administrative functions within the database. Despite the widespread nature of these attacks, they can be mitigated through preventive measures such as input sanitization, parameterized queries, and the use of web application firewalls. Vigilant implementation of these security strategies can effectively safeguard databases from SQL injection threats, ensuring data and system confidentiality, integrity, and availability.

10.0 Reference:

1. L. Ma, D. Zhao, Y. Gao, and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, Sep. 2019, doi: https://doi.org/10.1109/iccnea.2019.00042.
2. Sangeeta, S. Nagasundari, and P. Honnavali, "SQL Injection Attack Detection using ResNet," *IEEE Xplore*, Jun. 2019. https://doi.org/10.1109/ICCCNT45670.2019.8944874

3. D. Sharma, S. Sarkar, M. Elluru, J. Caroline, E. Fiorenza, and Student, "SQL INJECTIONS AND PREVENTION ACTIONS," vol. 5, Sep. 2018, Available: https://www.jetir.org/papers/JETIR1810379.pdf
4. H. Yuan, L. Zheng, L. Dong, X. Peng, Y. Zhuang, and G. Deng, "Research and Implementation of WEB Application Firewall Based on Feature Matching," *Application of Intelligent Systems in Multi-modal Information Analytics*, pp. 1223–1231, Mar. 2019, doi: https://doi.org/10.1007/978-3-030-15740-1_154.
5. K. Ahmad and M. Karim, "A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, May 2021, doi: https://doi.org/10.14569/ijacsa.2021.0120636.
6. S. Mavromoustakos, A. Patel, K. Chaudhary, P. Chokshi, and S. Patel, "Causes and Prevention of SQL Injection Attacks in Web Applications," *Proceedings of the 4th International Conference on Information and Network Security - ICINS '16*, Dec. 2016, doi: https://doi.org/10.1145/3026724.3026742.
7. S. McDonald, "23.pdf on Egnyte," *Egnyte*, Jun. 2021. https://sansorg.egnyte.com/dl/Sk5M5j53c2
8. R. Jahanshahi, A. Doupé, and M. Egele, "You shall not pass: Mitigating SQL Injection Attacks on Legacy Web Applications," *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, Oct. 2020, doi: https://doi.org/10.1145/3320269.3384760.
9. M. S Aliero, A. A Ardo, I. Ghani, and M. Atiku, "Classification of Sql Injection Detection And Prevention Measure ," Jan. 2016. https://www.academia.edu/download/42244717/B06210617.pdf
10. J. Ionel and M. Pirnau, "SQL INJECTION ATTACKS AND VULNERABILITIES Ionel IACOB 11 Mironela PIRNAU 2," Jul. 2020. Available: https://web.rau.ro/websites/jisom/Vol.14%20No.1%20-%202020/JISOM%2014.1%202020_68-81.pdf
11. R. Banda, J. Phiri, M. Nyirenda, and M. M. Kabemba, "Technological Paradox of Hackers Begetting Hackers: A Case of Ethical and Unethical Hackers and their Subtle Tools," *ICT Journal*, vol. 3, no. 1, p. 40, Mar. 2019, doi: https://doi.org/10.33260/zictjournal.v3i1.74.
12. Taylor and G. Allen, "SQL For Dummies," *Google Books*, Dec. 2018. https://books.google.com/books?hl=en&lr=&id=mGV2DwAAQBAJ&oi=fnd&pg=PA3&dq=Sheer+amount+of+companies%E2%80%99+data+in+stored+in+particular+database+servers&ots=pkPCVVJR65&sig=sI8xRuEvxsIuZSkAclC-XwWV2bg
13. J. Mache, C. García Morán, N. Richardson, W. Rollins, and R. Weiss, "Hands-on SQL Injection in the classroom: Lessons Learned *," Jan. 2022. Accessed: 1BC. [Online]. Available: https://par.nsf.gov/servlets/purl/10417342