Strategies for Safeguarding Against SQL Injection Attacks

Usman Azeem (20L-2064)
Computer Science
FAST-NUCES
Lahore, Pakistan

Muhammad Usama(20L-1233)
Computer Science
FAST-NUCES
Lahore, Pakistan

Zaid Khan (20L-2072)
Computer Science
FAST-NUCES
Lahore, Pakistan

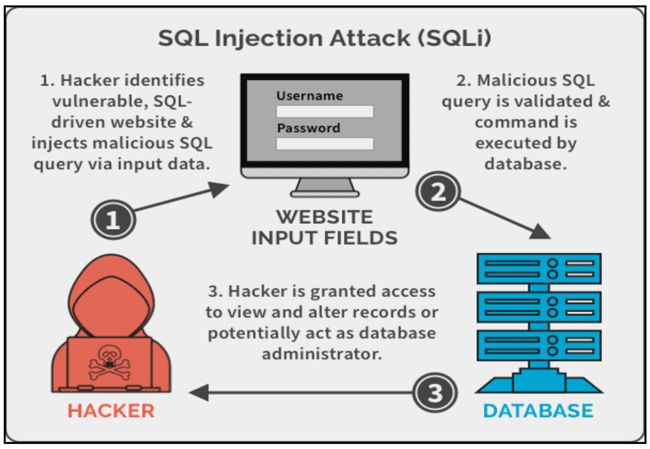
***Abstract*— The evolving digital era has led to the development of a huge number of websites. However, the security of user data and personal information poses a huge threat. The methods to attack a website like SQL injection are commonly used by attackers. The various types of SQL injection attacks are blind, In-Band, and Out-of-Band attacks. The biggest attack in history was the 7-Eleven breach in which attackers targeted several companies and stole around 130 million credit card numbers. There are a lot of techniques like input validation, firewalls, code-level defense, platform defense, design and techniques, and many more used to protect against SQL injection attacks. However, these practices limit the probability of attacks but do not guarantee complete prevention from attacks due to the dynamic nature of security.**

I. Introduction

Nowadays, with the advancement in technology day by day, online web services are used widely to give services to clients. This allows the development of interactive web applications for the use of thousands of clients. As the World Wide Web grows, the attacks also increase exponentially. Therefore, a lot of effective security measures need to be taken to protect user information and ensure the integrity of the system.

SQL injection is a cyber security attack in web applications to get unauthorized access to the database by attackers. Multiple websites allow users to input queries to get data from the database therefore they become vulnerable to SQL injection attacks. The Web applications allow users to input queries or conditions to fetch data. These queries are modified by the attacker in such a way that unlawful data is accessed by the attacker.

These attacks are very common as multiple websites lack input validation. These attacks can lead to an attacker gaining unauthorized data, manipulation of data, and complete control of the affected database. A company can get severe financial losses and damages in case of SQL injection vulnerabilities.


SQL Injection Attack (SQLi)
1. Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
Username
Password
2. Malicious SQL query is validated & command is executed by database.
1
2
WEBSITE INPUT FIELDS
3. Hacker is granted access to view and alter records or potentially act as database administrator.
3
HACKER
DATABASE

Attackers can easily modify, insert, update, and delete data leading to data corruption. SQL injection in some cases is used to insert malicious scripts in web pages, leading to cross-site scripting attacks. Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

A. Problem Statement

SQL injection attacks pose a huge security risk to users, causing severe damage to their professional, personal, and business-related information. This study will focus on SQLi attacks that users face and how to prevent these attacks. These attacks are dynamic therefore users continue to face threats and attacks for data breaches. We will be discussing in-depth analysis of types of SQL injection attacks, how they work, techniques to prevent these attacks, and their effectiveness in comparison to other techniques. Furthermore, we will discuss real-world SQLi attacks that happened and how those companies defended themselves against these attacks.

B. Project Goals

This research project aims to provide in-depth information about three types of SQL injection attacks, and how it allows unauthorized access to databases. Further, we will be highlighting the ways to prevent SQL injection attacks, including encoding inputs and positive pattern matching. The recent attacks like GhostShell, 7-Eleven, and GBGary breach will also be discussed. Finally, we will be providing a comparative analysis of SQL injection prevention techniques, highlighting their strengths and weaknesses.

C. Scope

This report aims to enlighten people about SQLI-cyber-attacks and guidance for the user on what actions to perform to prevent an SQLI attack. This information is valuable for a wide audience:

- Students
- Software Engineers
- Web Developers
- Security Analyst
- Government Officials with sensitive data
- Common People

The above list of people is affiliated with these attacks and will benefit from preventing SQLI cyber attacks in their workplace and surroundings. They will also get an insight into how to prevent these attacks.

II. Types of SQLi Attacks

In this section, we will be looking into the 3 types of SQLi attacks which are the most common apart from multiple types. Multiple attackers use a combination of SQLi attacks so that the probability to get access to data and perform malicious attacks is increased. The 3 types of SQLi attacks to be discussed are:

- In-Band SQLi
- Out-of-Band SQLi
- Inferential (Blind) SQLi

A. In-Band SQLi

In-band SQLi injection is the approach in which the attacker takes advantage of a single communication channel to send the malicious query and gather the results from the database. In simpler words, after sending the query, results are returned to the attacker through the web page. This type of SQLi attack has 2 subtypes.

1) Error-Based SQL Injection

This type of injection relies on error messages provided by the database. The internal structure of the database is exposed due to the response to these error messages. An attacker will inject a malicious query to gain sensitive information, database version, or how the query structure is to be used when getting results from the database. If the below-written query generates an error message it will automatically reveal the structure of the database.

SELECT * FROM User WHERE Username = 'Admin' AND Password = '123' OR 1=1;

2) Union Based SQL Injection

In union-based SQLi, an attacker combines the original query with his malicious query for execution. The things to be considered are that the number of columns and data types of columns must be the same. These things are important to consider as the query will give an error if these issues are not considered. The below query illustrates that it will return additional rows containing results from the injected malicious query by the attacker.

SELECT Username, Password FROM User WHERE Username = 'Admin' UNION SELECT 'Hacker', 'Malicious';

B. Out-of-Band SQLi

Out-of-band SQLi injection is used when the attacker is unable to use a single channel to launch an attack and gather results. This type of injection is least common as it is dependent on features used by database administrators. This type of injection requires DNS requests and HTTP requests to the server which is controlled by the attacker. Microsoft SQL Server "xp_cmdshell" allows DNS requests. These techniques are valuable when the server response is unstable (making an inferential time-based attack unreliable). The attacker parses DNS requests to his website to get the output. After the SQL query, the attacker will write a statement like this:

EXEC xp_cmdshell('nslookup malicious.attack.com') --';

C. Inferential SQLi

Inferential SQLi is also known as Blind SQLi. It occurs when an attacker is unable to directly see the result of SQL injection in the application but still, it is dangerous, unlike other SQLi attacks. The database structure is recreated by the attacker by sending payloads and observing the web applications' response to those payloads. This type of attack is often used in blind SQL injection. The attacker gets access to the information without getting input. There are two types of SQL injection attacks:

1) Time-Based SQL Injection:

The attacker will send a query that will use time delays to make the database wait before sending a response. To ascertain whether a query is True or False, the attacker examines how long the database takes to process it. The attacker does not rely on the results of the database. The example of a Time-Based SQL Injection is as follows:

SELECT * FROM users WHERE username = 'admin' AND IF(1=1, SLEEP(5), 0);

2) Boolean-Based SQL Injection:

This inferential technique relies on sending a query to the database and waiting for a response which can be True or False. The HTTP response's content will either change or stay the same based on the outcome. This type of injection requires a good understanding of the internal database structure and the application's response to queries being sent. In the case of large databases, this type of attack is slower as the attacker enumerates the database character by character. The example of a Boolean-Based SQL Injection is as follows:

SELECT * FROM User WHERE Username = '' OR '1'='1'

III. Motivations

In today's evolving digital era securing web applications against security threats is a critical aspect. SQL injection attacks are dangerous as they can compromise databases that contain confidential information of the user. Cyber-attacks are getting stronger day by day in terms of their severity and capability to exploit databases at the backend. Developers and other users need to get to know how these attacks occur and the severity experienced by companies in the past. Furthermore, this research will highlight the essential practices that need to be considered when developing a web application. Confidentiality of user information is the top priority of a company and safe coding techniques along with good design strategy must be used to ensure that user information is kept secure. This research paper will highlight all the essential strategies used to protect against SQL Injection attacks.

IV. Methodologies (or Literature Review)

Jeff Forristal, a cyber security researcher, initially reported the first SQLI attack in 1998. Following this, it was evident that some ways must exist to ensure that the systems are secure and prevent these attacks, as well as some approaches must exist to protect systems that are under attack.

In this chapter, we will look at a few approaches that can help systems protect themselves from these attacks and recover.

A. *Defensive Coding Techniques*

SQL injection vulnerabilities are frequently caused by poor input validation. Adopting safe coding practices is critical for mitigating these dangers.

Input encoding is a critical approach for preventing string injection attacks. Meta characters are commonly used by attackers to influence the SQL parser, causing it to perceive user input as SQL tokens. While restricting the use of these metacharacters is one approach, it may prevent legitimate users from delivering acceptable input containing such characters. A more effective method is to use functions that encode strings, guaranteeing that all meta characters are properly encoded. In this manner, the database treats them as regular characters.

Another important technique is positive pattern matching. Input validation mechanisms that can distinguish between acceptable and harmful input should be implemented by developers. In contrast to negative validation, positive validation focuses on detecting approved patterns or valid SQL inputs. While predicting every possible attack on a program may be difficult, developers may describe all allowed inputs. Positive validation is a more secure way of validating inputs since it concentrates on ensuring that inputs are correct rather than looking for forbidden patterns or SQL tokens.

B. *Detection and Prevention Techniques*

The various techniques used to detect and prevent SQL Injection are as follows:

1) *Black-box testing:*

Huang and colleagues introduced WAVES, a black-box testing technique for identifying SQL injection vulnerabilities in Web applications. WAVES utilizes a Web crawler to pinpoint potential injection points and applies predefined patterns and attack strategies to create attacks targeting these points. The system observes the application's responses to these attacks and utilizes machine learning to refine the attack tactics. While more effective than traditional penetration-testing techniques, it cannot guarantee completeness, a limitation inherent in all black-box and penetration-testing methods.

2) *Checking Code Statically:*

JDBC-Checker is a technique that statically validates the type validity of dynamically generated SQL queries. While it is not intended to detect general SQL injection attacks (SQLIAs), it can prevent attacks that exploit type incompatibilities in dynamically created query strings. JDBC-Checker addresses one of the primary causes of SQLIA vulnerabilities: insufficient input type verification. It falls short of detecting more general types of SQLI attacks, which commonly include syntactically and type-corrected queries. Wasserman and Su proposed combining static analysis with automated reasoning to ensure the absence of tautologies in application layer SQL queries.

3) *Combined Static and Dynamic Analysis:*

AMNESIA is a model-based approach that utilizes both static and dynamic analysis. AMNESIA employs static analysis in the static phase to construct models of allowed query types at each database access point. Queries are evaluated against these models before execution during the dynamic phase, identifying and preventing SQLIAs. While AMNESIA's static analysis is effective, the accuracy with which it constructs query models has an impact on its overall usefulness.

4) *Intrusion Detection System:*

Valeur and colleagues propose that SQLIAs be detected using an intrusion detection system (IDS). Their method entails training a machine learning methodology on a set of common application requests to construct models of common inquiries. The system then watches the program while it is running, flagging queries that deviate from the model as potential attacks. While their examination revealed success rates, learning-based techniques are dependent on the quality of the training set and cannot guarantee detection capabilities, with a bad training set increasing the likelihood of false positives.

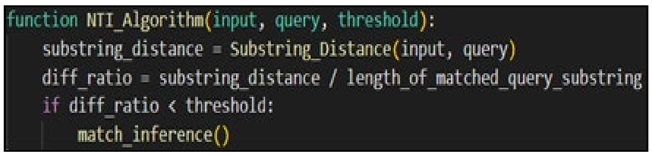
5) *Taint Based Approaches:*

Through static analysis, WebSSARI uses information flow analysis to identify input-validation issues. The method compares taint flows to the prerequisites for sensitive functions, identifying locations where the prerequisites are not met. It provides filters and sanitization procedures that can be incorporated into the program automatically to meet these criteria. The WebSSARI system considers input that has passed through a preset set of filters to be sanitized. During their investigation, the authors effectively found security flaws in a variety of existing apps. The assumption that sensitive functions have suitably describable preconditions within their typing system is a limitation of this method.

6) *Hybrid Taint Tracking:*

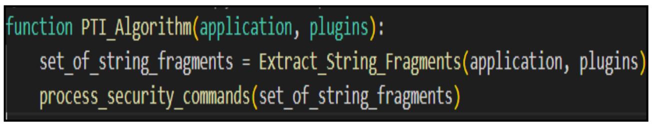
Joza is a well-known example of a hybrid taint tracking approach used to identify SQL assaults. Positive Taint Interface (PTI) Analysis and Negative Taint Interface (NTI) Analysis are the two main components of Joza.

By connecting application inputs and query strings, the Negative Taint Interface (NTI) Analysis explores taint markings. The NTI technique, depicted in pseudo-code, employs an approximation string-matching mechanism that allows for simple string changes such as whitespace removal and case conversions. Substring distance computes a difference ratio by dividing the string distance between an input and a query by the length of the matched query substring. A difference ratio smaller than a specific threshold indicates a match, and the algorithm's match inference is triggered.



```
function NTI_Algorithm(input, query, threshold):
    substring_distance = Substring_Distance(input, query)
    diff_ratio = substring_distance / length_of_matched_query_substring
    if diff_ratio < threshold:
        match_inference()
```

The Positive Taint Interface (PTI) method reconstructs security-critical commands from fragments of program strings. The PTI algorithm, which is effective in blocking OS command injection attacks, includes extracting a set of string fragments (abbreviated as F) by analyzing the program and any plugins to identify string literals.



```
function PTI_Algorithm(application, plugins):
    set_of_string_fragments = Extract_String_Fragments(application, plugins)
    process_security_commands(set_of_string_fragments)
```

V. Analysis

This chapter of our report will focus on the critical analysis of all the previous methodologies that are discussed in the previous chapter related to safeguarding against SQL-based injection attacks. Each approach has its positives and negatives and a deep understanding of the approaches is needed to make informed decisions related to the application of the approach.

A. Positive Pattern Matching

This is another defensive strategy that is discussed related to protection against SQL injection. It has some major positives as it minimizes the possibility of false positives by concentrating on validating well-known, acceptable input patterns and It also emphasizes predicted patterns while taking a proactive approach to input validation. Some limitations regarding this approach are that it requires a lot of understanding about the expected input patterns and it might not cover all potential attack patterns, as anticipation of every form of attack by developers is impossible. This approach provides a practical method for validating information. But for this strategy to be successful, developers must have a thorough grasp of the expected input patterns. For machine learning algorithms to dynamically learn and adapt to new acceptable input patterns, research could be conducted in this area.

B. Blackbox Testing (WAVES)

Huang and colleagues introduced WAVES, a black-box testing technique that uses a web crawler to find possible SQL injection vulnerabilities in online applications. Using pre-established attack patterns and strategies, it refines attack methods through machine learning in response to application responses. Although it is efficient, it does not provide the completeness guarantees associated with black-box testing techniques.

C. Static Code Checker (JDBC-Checker)

It is an important technique related to protection against SQL-based attacks. The primary objective of JDBC-Checker is to statically validate the type validity of SQL queries that are dynamically created. While it is not as good at identifying ordinary SQLIAs, it is effective at stopping attacks that take advantage of type mismatches. Wasserman and Su's combination of static analysis and automated reasoning overcome inadequate input type verification, but it is not immune to syntactically and type-correct inquiries.

D. Combined Static and Dynamic Analysis (AMNESIA)

AMNESIA is a model-based methodology that blends monitoring at runtime and static analysis. It builds models of acceptable queries using static analysis and dynamically validates them. Although it works well against SQLIAs, one possible drawback is that it depends on the precision of static analysis when building query models.

E. Intrusion Detection System (IDE)

IDS is a famous technique for protection against attacks or intrusions. In the methodology, the usage of an IDS with machine learning to find SQLIAs is suggested by Valeur and others. Common query models are constructed by the system, which then tracks model deviations while the application is running. Despite being successful, the effectiveness depends on the caliber of the training set; a subpar training set may result in false positives i.e. positive class would be predicted by the machine learning model that's incorrect.

F. Taint Based Approaches (WebSSARI)

In the analysis of Taint-based approaches, we will first discuss the WebSSARI which relies on information flow analysis to discover errors, and while this works well, however, its useability is limited because it assumes sufficient preconditions for sensitive functions. Future studies could look into ways to reevaluate this presumption and create more adaptable taint-based methods that can handle a larger variety of input validation conditions.

G. Hybrid Taint Tracking (Joza)

While analyzing the Taint Tracking we will now discuss Hybrid Taint Tracking (Joza). Joza's hybrid approach is comprehensive, combining PTI and NTI analyses. However, the need for careful tuning of threshold values suggests a potential challenge in achieving optimal performance across diverse applications. Research efforts could focus on developing adaptive threshold-setting mechanisms or automated tuning strategies for hybrid taint-tracking systems.

H. Comparative Analysis

This section depicts the summarization of the above analysis of different approaches related to safeguarding against SQL-based injection attacks. Each technique has its pros and cons. No single approach is 100% efficient in its implementation thus highlighting the necessity of choosing approaches carefully and contextually depending on the features of the intended web application.

The table summarizing all the approaches is on the next page.

Methodology	***Strength***	***Weakness***
Input Encoding	Mitigates SQL Injection by encoding meta-characters.	Implementation requires careful attention.
Positive Pattern Matching	Focuses on known legitimate input patterns.	May not cover all potential attack patterns.
Black-box Testing (WAVES)	Effective with machine learning refinement.	Lack of completeness guarantees.
Static Code Checker	Prevents type-mismatch attacks.	Limited against general SQL-based attacks.
Combined Static and Dynamic	Effective against SQL-based injection attacks.	Success is dependent on static analysis accuracy.
Intrusion Detection System	High success rate but dependent on the training set.	Potential for false positives with poor training.
Taint Based Approaches	Information flow analysis for error detection.	Assumption of adequate preconditions for functions.
Hybrid Taint Tracking (Joza)	Comprehensive approach with PTI and NTI analyses.	Threshold tuning is required for optimal performance.

The table above shows the summarization of critical analysis of these approaches which could help a user make a better decision regarding the choice.

I. Room For Further Research

In this section, we will further discuss any more room or areas for further research in addition to the techniques discussed above. These further areas of research will be related to these aspects:

1) Exploration of Dynamic Encoding Techniques:

Research could be done in the exploration of dynamic encoding techniques that can adapt to emerging threats and evolving meta-character usage patterns.

2) Algorithms related to Pattern Matching:

Research could be done regarding the examination of machine learning or Artificial Intelligence techniques for pattern recognition that can automatically pick up on and adjust to new, valid input patterns, eliminating the need for human definition.

3) Cross-Methodology Integration:

Research could be done in the exploration of methodologies or techniques that integrate multiple detection methodologies seamlessly, leveraging the strengths of each to create a more robust and comprehensive defense against SQL-based injection attacks.

4) Enhancement of Real-time Static Analysis:

Research could be done in the enhancement of static analysis techniques to provide real-time feedback during application development, enabling developers to address SQL injection vulnerabilities early in the development lifecycle.

5) Dynamic Adaptability:

Researchers could focus on the research methodologies that would focus on the dynamic adaption of mechanisms related to the detection of SQL-based injection attacks which are changing with time. This is a very crucial area of research as with every passing day a new attacker with a new attack strategy emerges in the world of cyber security.

Thus, research should focus on improving current techniques and investigating novel approaches to deal with the always-changing threats presented by SQL Injection Attacks. To create robust and efficient SQLIA detection and prevention systems, it will be essential to integrate different methodologies collaboratively and prioritize automation and adaptability.

VI. Conclusion

In conclusion, this study suggests that protecting against SQL infusion attacks requires a variety of methods. Defensive coding practices, such as input encoding and pattern matching, provide robust protection against malicious data. Identification and counteraction methods, such as black-box testing, static code checking, and intrusion detection systems, are crucial in identifying and preventing potential attacks. The model-based approach of AMNESIA, which combines static and dynamic examination, is effective but depends on the accuracy of its static investigation. Corrupt-based strategies like WebSSARI and crossbreeding like Joza offer potential for error identification and prevention. Further research is needed to address the evolving threat of SQL infusion attacks which involves research in areas like dynamic adaptation, exploration of algorithms related to pattern matching, exploration of dynamic encoding techniques, cross-methodology integration, and enhancement of real-time static analysis. Readers are advised to carefully read the approaches in detail and choose the appropriate one according to their needs, resources, and the types of attacks that they could potentially face.

References

[1] S. M. S. Sajjadi and B. Tajalli Pour, “Study of SQL Injection Attacks and Countermeasures,” International Journal of Computer and Communication Engineering, pp.539–542, 2013, doi: https://doi.org/10.7763/ijcce.2013.v2.244.

[2] G. Shrivastava and K. Pathak, “SQL Injection Attacks: Technique and Prevention Mechanism,” International Journal of Computer Applications, vol. 69, no. 7, pp. 35–39, May 2013, doi: https://doi.org/10.5120/11857-7626.

[3] A. Maraj, E. Rogova, G. Jakupi, and X. Grajqevci, “Testing techniques and analysis of SQL injection attacks,” IEEE Xplore, Oct. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8169902/ (accessed May 03, 2020).

[4] A. K.Kolhe and P. Adhikari, “Injection, Detection, Prevention of SQL Injection Attacks,” International Journal of Computer Applications, vol. 87, no. 7, pp. 40–43, Feb. 2014, doi: https://doi.org/10.5120/15224-3739.

[5] J. O. Atoum and A. J. Qaralleh, “A Hybrid Technique for SQL Injection Attacks Detection and Prevention,” International Journal of Database Management Systems, vol. 6, no. 1, pp. 21–28, Feb. 2014, doi: https://doi.org/10.5121/ijdms.2014.6102.

[6] M. Medhat, C. Albert, and M. M. El Hadi, “Efficient Solution for Detection and Prevention of SQL Injection Attacks: Wave System Technique,” Compunet (The Egyptian Information Journal), vol. 16, no. 18–19, pp. 13–23, Mar. 2017, doi: https://doi.org/10.12816/0050477.

[7] T. K. George and P. Jacob, “A Proposed Architecture for Query Anomaly Detection and Prevention against SQL Injection Attacks,” International Journal of Computer Applications, vol. 137, no. 7, pp. 11–14, Mar. 2016, doi: https://doi.org/10.5120/ijca2016908808.

[8] M. Martin, B. Livshits, and M. S. Lam “Finding Application Errors and Security Flaws Using PQL: A Program Query Language”, ACM Notices, Volume 40, Issue:10 pages, 2005.

[9] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: A static analysis tool for SQL/JDBC applications," 2004, pp. 697- 698