Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT AND PROJECT COVER SHEET

Unit Code: COS300015 Unit Title: IT Security

Assignment number and title: 1 - Research report Due date: 25/06/2023

Lab group: none Tutor: none Lecturer: Mr. Kevin Loc

Family name: Dang Identity no: 103802759

Other names: Josh

To be completed if this is an INDIVIDUAL ASSIGNMENT

I declare that this assignment is my individual work. I have not worked collaboratively, nor have I copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for me by another person.

Signature: Luan

To be completed if this is a GROUP ASSIGNMENT

We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person.

ID Number	Name	Signature

Marker's comments:

Total Mark: ________________

Extension certification:

This assignment has been given an extension and is now due on ________________

Signature of Convener: ________________ Date: / 2023

6/25/2023

COS30015

IT Security – Assignment 1

Analysis of SQL Injection: Implementation, Detection, and Prevention Techniques

VI LUAN DANG

103802759

Analysis of SQL Injection: Implementation, Detection, and Prevention Techniques

Vi Luan Dang - 103802759

Faculty of Science, Engineering and Technology

Swinburne University of Technology

Ho Chi Minh, A35 Bach Dang, Tan Binh District

Abstract

Our everyday internet activities, such as online shopping and banking, rely heavily on database-driven web applications. These web applications and their underlying databases store our personal information, which we entrust to them (Halfond & Orso, 2007). However, the confidentiality and integrity of this sensitive information are not guaranteed, as malicious individuals attempt to compromise it regularly. One major threat to such information is SQL Injection (SQLI), which allows attackers to bypass authentication mechanisms and access, modify, or delete sensitive data stored in databases (Alghawazi, Alghazzawi & Alarifi, 2022). In this paper, we provide a comprehensive analysis of various SQL injection implementations ranging from classic SQL injection, blind SQL injection, and time-based SQL injection, as well as other advanced techniques for performing SQLI. We also examine various detection methods ranging from static detection and dynamic detection to advanced machine learning-based detection. Finally, we discuss various prevention strategies that can be used to mitigate the risk of SQLI attacks.

Keywords: SQL Injection (SQLI), classic SQL injection, blind SQL injection, time-based SQL injection, signature-based detection, anomaly-based detection, machine learning-based detection, prevention strategies.

1. Introduction

SQL Injection Attacks (SQLIAs) pose a serious threat to the security of database-driven applications and are included in the top ten vulnerabilities by the Open Web Application Security Project (OWASP) (T. O. Foundation, 2005). Major software companies, including Microsoft and SPI Dynamics, have also recognized SQLIAs as a critical vulnerability that developers must address to protect their systems (Aucsmith, 2004). Despite being a well-known technique and a prevalent issue, SQLI is challenging to prevent, as attackers continue to develop sophisticated ways to bypass the security measures in place.

Classical SQL Injections were relatively easy to prevent and detect, as there are many procedures and methodologies that have been implemented to address these risks (Singh, 2016) . One of the prime examples is the secure code writing techniques developed by Howard and his team (M. Howard & D. LeBlanc, 2003), which emphasize the importance of writing defensive code with proper validation of user inputs.

Defensive coding is encouraged but itself alone cannot guarantee the protection of a web application (Singh, 2016), as attackers can utilize additional attacking techniques in conjunction with SQLI to achieve malicious intent. Moreover, the abundance of SQLI methods means that attackers have a wide range of options when it comes to exploiting vulnerabilities in web applications (Alotaibi & Vassilakis, 2023).

There are numerous types of SQLIAs, and each has a different approach for attacking websites. Vulnerabilities exist everywhere in the web application, from the web form submission to the web's URL and authentication mechanism (Al-Shareeda, Manickam, & Sari, 2022). To address these challenges, this paper provides an extensive discussion of the modern SQL Injection attacks and the ways in which we can utilize to protect and defend against these types of attacks.

The paper is divided into five sections. Section 1 provides an introduction to SQLI and why it is important to address this threat. Section 2 describes how the paper is conducted and written. Section 3 provides detailed information on the most noteworthy type of SQLI attacks and various other techniques that can be used in conjunction with SQLI attacks. Section 4 discusses the detection and prevention techniques of attacks listed in Section 3. Finally, Section 5 provides a summary and conclusion.

2. Research method

This literature review is conducted in four main phases: (A) Planning the review; (B) Obtaining related research papers; (C) Analyzing selected papers; and (D) Discussing key points and analysis of the papers. Each phase corresponds to a specific section of the paper. Sections 2 and 3 cover the planning and selection of research documents, while Sections 4 and 5 involve the discussion of the selected documents. Figure 1 illustrates the research phases.

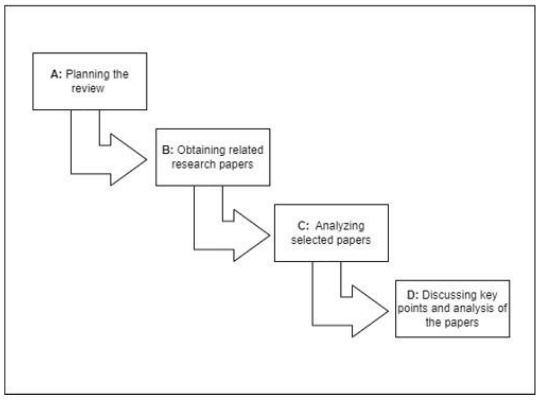
A: Planning the review
B: Obtaining related research papers
C: Analyzing selected papers
D: Discussing key points and analysis of the papers

Figure 1: research phases

2.1 Planning the review.

In this phase, I will summarize some of the most noteworthy question related to the understanding and discussion of SQLI, this question will also be the main topic of this paper. Some of the questions include:

+ What is SQLI and its target?
+ How can SQLI methods be categorized?
+ Types of SQLI and other methods that can be used in conjunction with SQLI.
+ What can be the prevention and detection techniques of the listed methods?

2.2 Obtaining the related research papers.

Related research documents are one of, if not the most important aspect of a literature review. Extensive and detailed strategy was made to make sure that selected documents are appropriate, high quality, and in scope with the chosen topic.

The library or sources what was used to retrieve the research papers were IEEE, Springer, Academia, PortSwigger and Science Direct. The retrieved papers are published between 2010 and 2023 with theexception of some books or techniques or principles. Some inclusion and exclusion criteria of the documents are as follows:

2.2.1 <u>inclusion criteria</u>

+ Papers related to SQL Injection attacks.
+ Papers from scientific, trusted databases such as ACM, IEEE, Springer, Academia, ResearchGate.
+ Papers about the detection and prevention of SQLI.

<u>2.2.2 Exclusion criteria</u>

+ Papers not covering the SQLI topic.
+ SQL automation papers published before 2010.
+ Papers with untrusted sources or without details about the author,

2.3 Analyzing selected papers.

After the preparation steps, the analyzing of selected papers was conducted to obtain the key points and knowledge from these papers in order to gain a solid and advance understanding of SQLI. The information provided in Section 3 and 4 will be the overall conclusion and understanding drawn from 18 research papers.

2.4 Discussing key points and analysis of the paper.

Finally, the similarities and differences between 20 selected research papers are summarized to provide a comprehensive view of this threat. The summarized understanding is then used as a guideline for writing this review.

3. Nature and different implementations of SQLI

Before we can dive in the implementation of SQLI, it is crucial to first have a basic understanding of the target of SQLI as well as the nature of this threat.

3.1 Database-driven Web application:

Our daily use of the internet, including e-commerce, content management, and online banking, relies heavily on web applications that are driven by databases. In this type of architecture, the database is essential to the application's functionality, as it stores all the data necessary for the application to operate (R., Shobana, & Phil, 2021). Consequently, the security of the database is crucial to the overall security of the application. Therefore, understanding SQL injection (SQLI) is critical to comprehending the nature, function, and design of database-driven web applications.

As shown in Figure 2, a database-driven web application comprises three tiers: the presentation tier, which incorporates the web browser; the logic tier, which includes the programming language; and the storage tier, which includes the database. The middle tier receives requests from the presentation tier and responds by querying and updating the back-end. When malicious code is entered into the front-end parameters, it is eventually retrieved by the back-end database for processing and execution (Al-Shareeda, Manickam, & Sari, 2022), which is how SQLI attacks occur (Umar et al., 2016). Defensive coding is one method for preventing SQLI attacks. However, implementing and enforcing such measures can be challenging, as attackers are constantly discovering new ways to circumvent them.

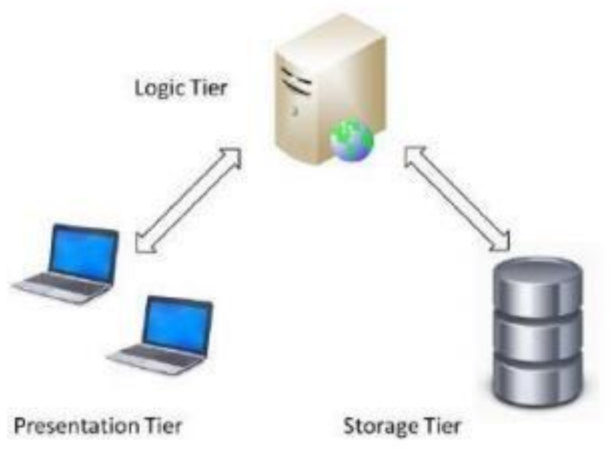

Figure 2: Three-tier web application

3.2 An example of SQLI into web application:

The SQL injection (SQLI) attack follows a basic process comprising three steps. Firstly, attackers perform reconnaissance or scanning of a targeted website to identify vulnerabilities related to SQLI manually or using tools. Once the vulnerabilities are identified, the attacker proceeds to exploit them by crafting a malicious SQL query to be executed by the database. Upon successful execution of the query, the attacker gains access to view and alter records or potentially escalate privileges to the database administrator (AL-Maliki & Jasim, 2022). It is crucial to note that the attack does not necessarily end there, as attackers may also implement backdoors or discover other vulnerabilities to further compromise the system once they gain access to the infrastructure. The figure below illustrates the basic process of an SQLI attack.

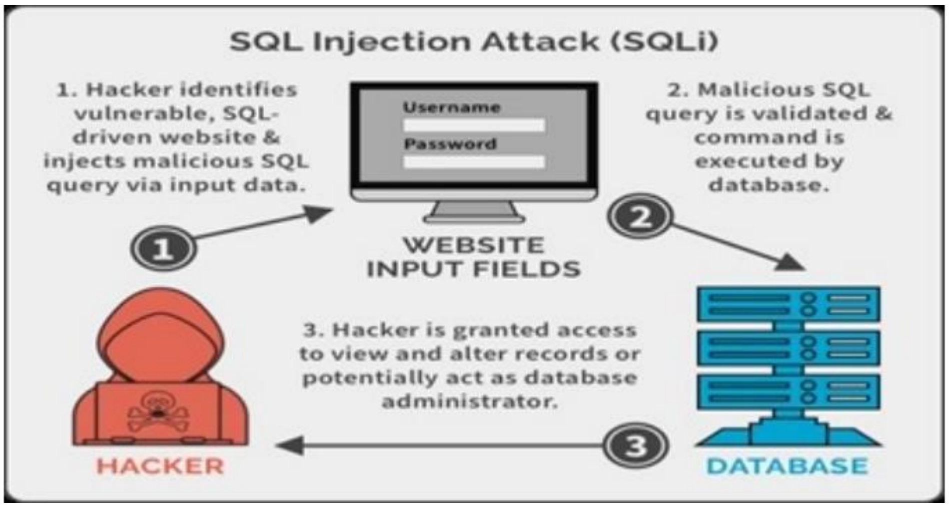
SQL Injection Attack (SQLi)
1. Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
Username
Password
2. Malicious SQL query is validated & command is executed by database.
1
2
WEBSITE INPUT FIELDS
3. Hacker is granted access to view and alter records or potentially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

Figure 3: SQL injection attack to web-based application.

The most basic utilization of SQL injection is to use a modified SQL query to return additional results. Consider a shopping application that displays products in different categories. When a user clicks on the `Gift` category (PortSwigger, 2021), their browser requests the following URL:

`https://insecure-website.com/products?category=Gifts`

This causes the application to make a SQL query to retrieve details of the relevant products from the database:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

If the application does not implement any defenses against SQL injection, the attacker can construct an attack like:

```
https://insecure-website.com/products?category=Gifts'--
```

Which will result in the SQL query:

```
SELECT * FROM products WHERE category = 'Gifts' --' And released = 
1
```

As the double-dash sequence '--' is a comment indicator in SQL, the database will interpret everything after the sequence as a comment, effectively eliminating the condition of the query. This is the most basic form of SQLI. Further discussion about other types of SQLI will be provided in the next part.

3.3 Types of SQL injection attacks

The SQL Injection attacks takes on many forms and variation. Some of them are specified as follows:

3.3.1 Tautology-based Attack

A tautology-based attack is a type of SQL injection attack that aims to insert SQL tokens into an application's input field (Halfond & Orso, 2007). The goal is to manipulate the query's conditional statement in such a way that it always evaluates to true, allowing the attacker to bypass authentication pages and extract data. Consider the login form below:

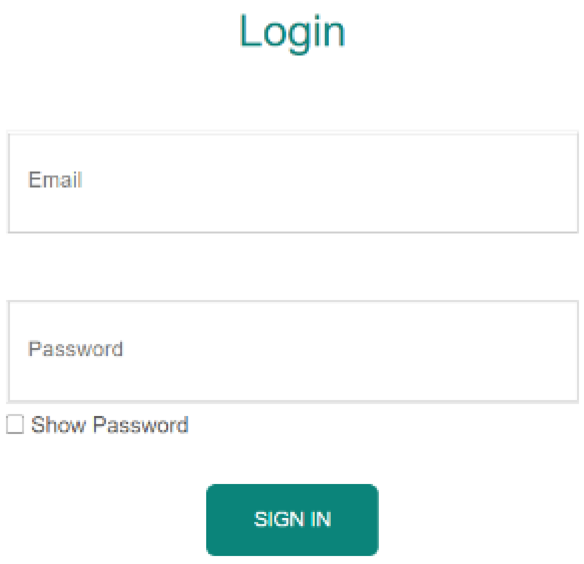

Figure 4: Login form

If a user submits the username "`luandang`" and the password "`Swinburne`", the application checks the credentials by performing the following SQL query:

```
SELECT * FROM users where username = 'luandang' AND password =
'Swinburne'
```

An attacker can insert a SQL comment sequence `'--'` to remove the password checking from the `'WHERE'` clause of the query by submitting the username `"administrator'--"` and the password as `"123"`. Without proper defense mechanism, the web application will submit the following SQL query to the database:

```
SELECT * FROM users where username = 'administrator'
```

The above query makes the application insecure as the checking mechanism for password has been commented out using SQL comment sequence, and the attacker will be granted with administrator privilege. The main usage of this attack is to extract unauthorized information and therefore escalate user privilege to achieve greater aim.

3.3.2 Piggy-backed query attack

This attack involves adding extra query statements to the original query by utilizing a query delimiter (such as ';'). The original query is safe, but the additional query statements are considered to be injected queries (Shehu & Xhuvani, 2014).

Suppose we submit the username of "luandang" and the password as"Swinburne'; drop table user_table;", this will cause the application to send this SQL query to the database:

```
 SELECT * FROM users WHERE username = 'luandang' AND password =
'Swinburne'; drop table user_table;
```

Without proper protection, the attacker can join injected query to the safe query and manipulate the execution of the database, in this case dropping the table user_table. This would cause great harm to the functionality of the web application, or at worst deny the website of any functionality at all.

3.3.3 Union-based attack

The Union keyword in SQL allows the execution of one or more additional SELECT queries and append the results to the original query, for example:

```
SELECT a, b FROM table1 UNION SELECT c, d FROM table2;
```

This SQL query will return a single result set with two columns, containing values from columns a and b in table1 and columns c and d in table2.

FOR a `UNION` query to work, two key requirements must be met:

1. The individual queries must return the same number of columns.
2. The data types in each column must be compatible between the individual queries.

The detailed implementation and steps of which to perform this attack is, however, out of the scope of this paper, therefore, consider the login form of figure 4:

Suppose we submit the username "`luandang’ UNION SELECT * FROM users WHERE username=’admin’-- and password=’1234’”`. This will cause the application to send the following SQL Query:

```
SELECT * FROM users WHERE username=’luandang’ UNION SELECT * FROM
users WHERE username =’admin’
```

The above query concatenates the safe query with the malicious payload by using UNION keyword and therefore, make the statement vulnerable. The intention of this attack is to bypass authentication and extract valuable data from the database.

3.3.4 Stored Procedure attack

Store procedures are widely used as an additional database management method. They usually are PL/SQL codes compiled into a single execution plan and extensively used for performing commonly occurring tasks (Singh, 2016).
It is commonly thought by developers that stored procedure is the remedy for SQL Injection as it does not use the standard SQL. However, different vulnerability related to the scripting language still exist without careful consideration. Consider the below stored procedure:

```
CREATE PROCEDURE user_info
 @username varchar2
 @pass varchar2
 @customerid int
AS
 BEGIN
  EXEC ( ‵ SELECT customer_info fromcustomer_table WHERE
    username = ‵ ‵ ‵+ @username ’ ’ ’ and pass = ‵ ‵ ‵+ @pass ’ ’ ’
  GO
```

This stored procedure contains vulnerability as it constructs a dynamic SQL statement using input parameter @username and @pass without sanitizing them. Therefore, the attack can exploit this vulnerability by inputting malicious SQL code into the input parameter. For example, the attacker can inject ‘ ' OR 1=1; -- ‘ into the @username parameter, making the SQL statement constructed by the stored procedure to be manipulated as below:

```
SELECT customer_info from customer_table
WHERE username = '' or 1=1;
```

The added code after the injected parameter causes the database to ignore the rest of the original query, resulting in the execution of the malicious query with all rows returned.
This type of attack could bypass authentication checks and allow the attacker to access customer information without proper authorization.

The four SQLI techniques mentioned above are typically referred to as "traditional" or "classical" SQLI. These methods exploit vulnerabilities in systems that lack appropriate defenses, and the outcome of the malicious SQL payload can be easily observed (Shehu & Xhuvani, 2014). However, these techniques can be thwarted through the use of defensive coding and proper sanitization practices. Despite this progress, the development of SQLI has not come to a halt, as other "advanced" implementations have been proposed that are capable of bypassing defensive coding protections. The detailed description of these techniques will be provided below.

3.3.5 Blind SQL Injection Attack

Blind SQL injection arises when an application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

With blind SQL injection vulnerabilities, many techniques such as UNION attacks, are not effective because they rely on being able to see the results of the injected query within the application's responses (Shehu & Xhuvani, 2014) . It is still possible to exploit blind SQL injection to access unauthorized data, but different techniques must be used.

Consider an application that uses tracking cookies to gather analytics about usage. Requests to the application include a cookie header like this:

```
Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4
```

When a request containing a TrackingId cookie is processed, the application determines whether this is a known user using a SQL query using the following SQL statement:

```
SELECT TrackingId FROM TrackedUsers WHERE TrackingId =
'u5YD3PapBcR4lN3e7Tj4'
```

This is vulnerable to SQL injection, but the result from the query is not returned to the user. However, the application does behave differently depending on whether the query returns any data. If it returns data (because a recognized TrackingId was submitted), then a message or some kind of indication might appear. This is, however, case-dependent and extensive reconnaissance and testing must be done prior to the implementation steps.

To know if this pattern exists or not, we can construct 2 payloads as follows in the HTTP request:

```
TrackingId = 'u5YD3PapBcR4lN3e7Tj4' AND '1' = '1'
TrackingId = 'u5YD3PapBcR4lN3e7Tj4' AND '1' = '2'
```

If the first payloads work AND the second payload does not work, then this pattern is vulnerable to blind SQL injection.

For example, suppose there is a table called `` `Users` `` with the columns `` `Username` `` and `` `Password` ``, and a user called `` `Administrator` ``. We can systematically determine the password for this user by sending a series of inputs to test the password one character at a time.

```
TrackingId = 'u5YD3PapBcR4lN3e7Tj4' AND SUBSTRING((SELECT Password FROM
Users WHERE Username = 'Administrator'), 1, 1) = 'm
TrackingId = 'u5YD3PapBcR4lN3e7Tj4' AND SUBSTRING((SELECT Password FROM
Users WHERE Username = 'Administrator'), 2, 1) = 't
```

With the above payload, If the first payload returns a message or some kind of indication then the first character of the password is 'm' and similarly, if the second payload returns an indication, then the second character is 't' (PortSwigger, 2021),. This can be done until the full password is retrieved.

3.3.6 Fast Flux SQL Injection Attack

In a fast flux SQL injection attack, the attacker injects SQL code into a vulnerable input field in a web application. The injected code creates a command and control (C&C) channel to the attacker's server (Lu et al., 2023). , allowing the attacker to execute arbitrary code on the database server and exfiltrate data from the database. The illustration below demonstrates how this attack may occur.

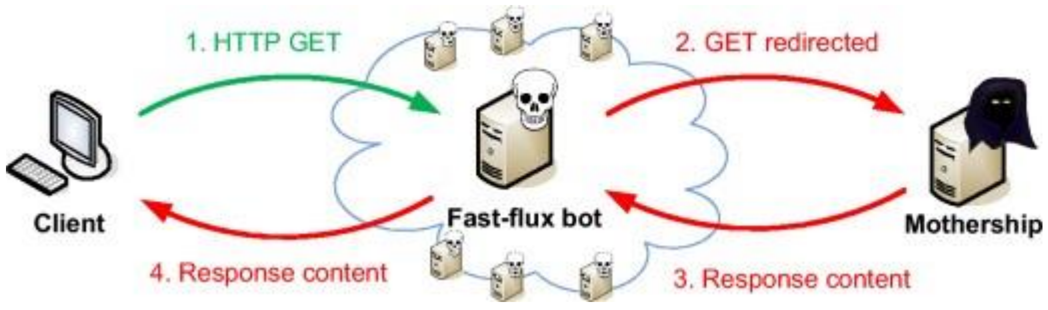

Figure 5: Fast Flux Attack

Using fast-flux network to rapidly change the IP address of the C&C , the attacker can evade detection of security measures making it exponentially challenging for security measures to identify and block the attackers’ infrastructure.

3.3.7 Compounded SQL Injection Attack

Compounded SQL injection is a type of SQL injection attack that involves chaining together multiple SQL injection vulnerabilities to achieve a more sophisticated and powerful attack. In a compounded SQL injection attack, the attacker may identify multiple vulnerabilities in the target that can be exploited to perform SQL Injection attack. The most noteworthy of this kind of attack are **SQLI + DDoS Attacks** and **SQLI + DNS Hijacking.**

SQLI + DDoS Attack: Distributed Denial of Service or DDoS is described as an attack that is used to strain the resources of a server so that the user is not able to access its functionalities. SQLI, however, is a technique to generate a payload to manipulated application’s execution (Singh, 2016). Therefore, SQLI can be used in order to pursuit DDoS Attack using advance SQL commands such as encode, compress, join etc.

Suppose an attacker found out a vulnerability in the web application that is prone to SQLI, a payload may be crafted into the vulnerable parameter to perform DDoS as follows:

```
‘ UNION ALL SELECT
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52
,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,
77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,
101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,1
19,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,13
7,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155
,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,
174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,1
92,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,21
0,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228
,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,
247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,2
65,266,267,268,269,270…,
```

This payload injects a large number of UNION statements (up to 437 in this example), each of which generates a separate request to the server. This can quickly overwhelm the server's resources and cause it to become unavailable to legitimate users. The attacker could use this SQL injection payload in combination with a botnet of compromised computers to launch a distributed denial of service (DDoS) attack against the target server. Hence, our DDoS attack using SQLI will be achieved.

SQLI + DNS Hijacking: DNS Hijacking is a type of attack where an attacker intercepts and manipulates DNS queries and responses in order to redirect users to a malicious server. The attacker main goal is to embed the SQL Query in DNS request and capture it (Singh, 2016). Once DNS Hijacking has been achieved then the SQL Injection attack can be combined with DNS lookup to retrieve sensitive information.

```
do_dns_lookup ( ( SELECT password FROM users where username =
'administrator') + ’. insecure-website.com’ ) ;
```

The attacker uses a SELECT statement to retrieve the password hash they are interested in, which they then append to a domain name that they control (such as insecure-website.com) using DNS hijacking techniques. They then perform a DNS lookup by searching for a dummy hostname and monitor the name server for their domain using a packet sniffer, waiting for the DNS record that contains the password hash to be returned.

4. Detection and Prevention of SQLIAs

As SQLI techniques continue to evolve and become more robust, so too do the methods and methodologies used to detect and prevent these threats. This section will explore various papers that provide information on detection techniques and prevention strategies for SQLIAs.

4.1 Detection Methods of SQLIAs

Despite the limitations of traditional vulnerability detection technology in many application scenarios, its research provides a technical foundation and viable development path for future research in more advanced vulnerability detection technology. Common techniques for detecting SQL injection attacks include static analysis, dynamic analysis, a combination of both. The detection architecture is shown in Figure 6.

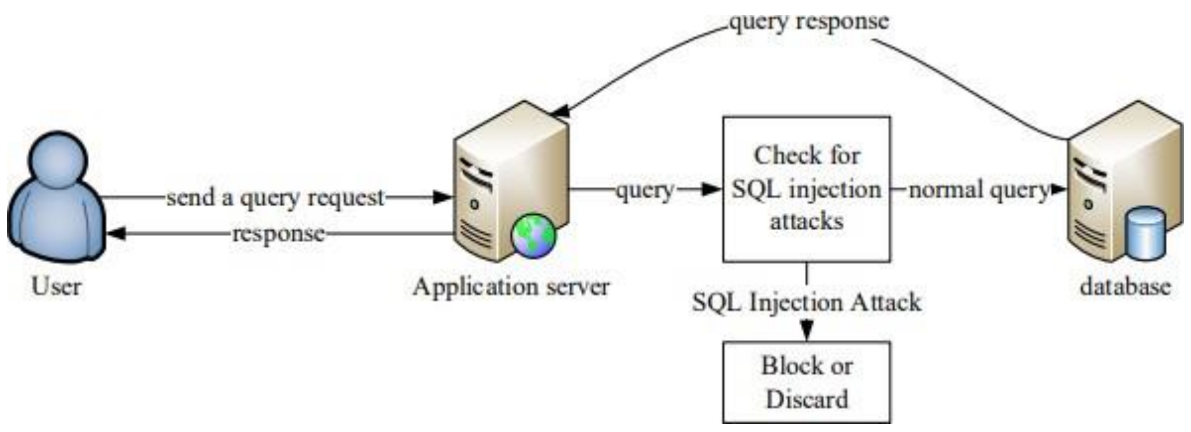

Figure 6: Traditional detection methods

Static analysis techniques involve analyzing incoming statements, commonly through source code analysis without executing the program to identify possible syntax error and type errors that could lead to SQL Injection (Lu, Fei, & Liu, 2023) . Signature detection is a common approach in traditional detection methods, this approach involvescreating a signature of known web attacks, and when that signature is detected in the incoming request, the suspicious traffic can be dropped through a fire wall or other security mechanism. This method, however, relies heavily on previously detected attack patterns. **Dynamic analysis**

analyzing its behavior while it's executing to detect any potential vulnerabilities or attacks (Lu, Fei, & Liu, 2023). This approach is ,however, resource-intensive and may not be suitable for real-time detection.

Combined static and dynamic analysis detection is a hybrid approach that combines static code inspection with real-time monitoring at runtime to detect and block SQL injection attacks (Yane & Chaudhari, 2013) . Static analysis can identify potential vulnerabilities in the source code, while dynamic analysis can confirm whether those vulnerabilitiescan be exploited in practice.

In the recent years, **AI detection methods** for SQLI have also emerged. These techniques involved statistical feature extraction methods to create features for machine learning classifiers or integrated learning models. The goal was to detect the presence of SQL injection attacks in real-time data streams. The detection of SQLI has been improved using a combination of machine learning classifier and rule engine (Azman et al., 2021). Other advanced detection method involves using a deep natural language processing-based tool for generating test cases for detecting SQLI.

4.2 Prevention Techniques of SQLIAs

Prevention of SQLIAs involves implementing measures to prevent attackers from injecting malicious SQL code into our environment. Web Application Firewalls (WAFs) and password-based cryptography are the most noteworthy techniques in this category.

WAFs use a variety of techniques to detect SQLI attacks, including signature-based detection, behavioral analysis, and machine learning (Alotaibi & Vassilakis, 2023) . Signature-based detection, as mentioned above, is an approach relies on patterns of previously attempted SQLI attacks. Behavioral analysis involves analyzing the behavior of the user or application to detect abnormal activity that may indicate an attack. Machine learning algorithms canbe trained to identify patterns of suspicious behavior that may be indicative of an SQLI attack.

Other strategies presented by Juanita Blue, Eoghan Furey, and Joan Condell is the prevention of SQLI using **Password-based Cryptography** (Blue et al., 2017). Figure 7 illustrated the implementation of this technique:

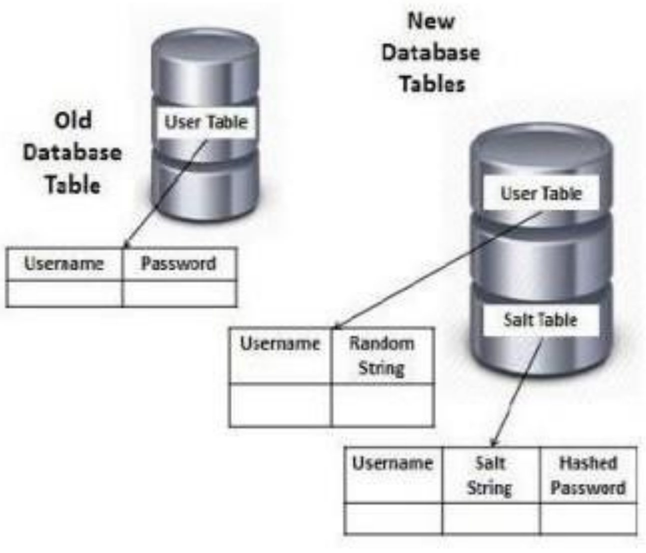

Figure 8: Password-Base Cryptography

This technique involves adding columns for storing the username, a salt string, and a protected password for each user account. The main user table includes columns for storing the username and a randomly generated string of character. To protect the plaintext passwords, they are first salted with a random string and then encrypted or hashed using a secure, one-way cryptographic hash function such as SHA-256 or SHA-512. The output of this hash function is then rehashed 1000 times to make it even more secure. Figure 9 demonstrates the algorithm used to change a password from plaintext to hashed.

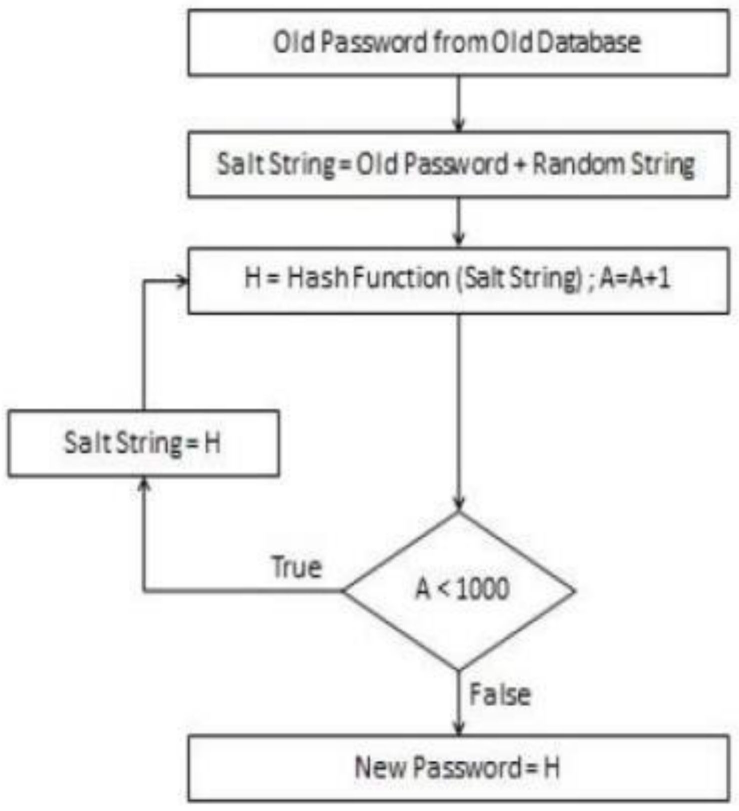
Old Password from Old Database
Salt String = Old Password + Random String
H = Hash Function (Salt String) ; A=A+1
Salt String = H
True
A < 1000
False
New Password = H

Figure 9: Password-hashed algorithm

5. Conclusion

In conclusion, SQL injection (SQLi) attacks pose a persistent threat to web applications and databases, and they can have severe consequences such as data manipulation, unauthorized data access, or complete system compromise. Therefore, it is crucial to implement effective measures to prevent and detect SQLi attacks.

This paper has provided a comprehensive overview of SQLi attacks, including their types, implementation techniques, and potential impact on web applications. Additionally, we have discussed various prevention and detection techniques, such as input validation, web application firewalls (WAFs), and password-based cryptography. IT practitioners must take a proactive role in implementing appropriate and effective security measures to prevent and detect SQLi attacks. By doing so, they can ensure that sensitive data and infrastructure remain protected from cyber-attacks.

References

Halfond, W.G.J., & Orso, A. (2007). Detection and Prevention of SQL Injection Attacks. In M. Christodorescu, S. Jha, D. Maughan, D. Song, & C. Wang (Eds.), Malware Detection (pp. 83-104). Springer. https://doi.org/10.1007/978-0-387-44599-1_5.

Alghawazi, M.; Alghazzawi, D.; Alarifi, S. (2022). Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. Journal of Cybersecurity and Privacy, 2(4), 764-777. https://doi.org/10.3390/jcp2040039.

T. O. Foundation. (2005). Top ten most critical web application vulnerabilities. Retrieved from http://www.owasp.org/documentation/topten.html.

Aucsmith, D. (2004, September). Creating and maintaining software that resists malicious attack. Distinguished Lecture Series. Retrieved from http://www.gtisc.gatech.edu/aucsmith_bio.html.

Shehu, B. & Xhuvani, A. 2014, 'A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques.

Al-Shareeda, M. A., Manickam, S., & Sari, S. A. (2022). A Survey of SQL Injection Attacks, Their Methods, and Prevention Techniques. Paper presented at the 2022 International Conference on Data Science and Intelligent Computing (ICDSIC), Karbala, Iraq (pp. 31-35). doi: 10.1109/ICDSIC56987.2022.10075706.

R., Shobana, Suriakala, Dr., & Phil, M. (2021). A Thorough Study On Sql Injection Attack-Detection And Prevention Techniques And Research Issues.

Singh, J. P. (2016). Analysis of SQL Injection Detection Techniques. Theoretical and Applied Informatics, 28. doi: 10.20904/281-2037.

Halfond, W. G. J., & Orso, A. (2007). Detection and Prevention of SQL Injection Attacks. In M. Christodorescu, S. Jha, D. Maughan, D. Song, & C. Wang (Eds.), Malware Detection (pp. 87-114). Springer, Boston, MA. doi: 10.1007/978-0-387-44599-1_5.

Lu, D., Fei, J., & Liu, L. (2023). A Semantic Learning-Based SQL Injection Attack Detection Technology. Electronics, 12(6), 1344. doi: 10.3390/electronics120613

Appelt, D., Panichella, A., & Briand, L. (2017). Automatically Repairing Web Application FirewallsBased on Successful SQL Injection Attacks. Paper presented at the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France (pp. 339-350). doi: 10.1109/ISSRE.2017.28.

AL-Maliki, M., & Jasim, M. (2022). Review of SQL injection attacks: Detection, to enhance thesecurity of the website from client-side attacks. International Journal of Nonlinear Analysis and Applications, 13(1), 3773-3782. doi: 10.22075/ijnaa.2022.6152.

AL-Maliki, M., & Jasim, M. (2022). Review of SQL injection attacks: Detection, to enhance the security of the website from client-side attacks. International Journal of Nonlinear Analysis and Applications, 13(1), 3773-3782. doi: 10.22075/ijnaa.2022.6152.

Alotaibi, F. M., & Vassilakis, V. G. (2023). Toward an SDN-Based Web Application Firewall: Defending against SQL Injection Attacks. Future Internet, 15(5), 170. doi: 10.3390/fi15050170.

Howard, M., & LeBlanc, D. (2003). Writing Secure Code. Pearson Education.

Umar, K., Bakar Md Sultan, A., Zulzalil, H., Admodisastro, N., & Taufik, M. (2016). SQL Injection Attack Roadmap and Fusion. Indian Journal of Science and Technology, 9(28), 1-8.

PortSwigger. "SQL Injection." PortSwigger, 2021, https://portswigger.net/web-security/sql-injection.

Yane, P. Y., & Chaudhari, M. S. (2013). SQLIA: Detection and Prevention Techniques: A Survey. IOSR Journal of Computer Engineering (IOSR-JCE), 2, 56-60.

Azman, Muhammad & Marhusin, M.F. & Sulaiman, Rossilawati. (2021). Machine Learning-Based Technique to Detect SQL Injection Attack. Journal of Computer Science. 17. 296-303. 10.3844/jcssp.2021.296.303.

Blue, Juanita & Furey, Eoghan & Condell, Joan. (2017). A novel approach for secure identity authentication in legacy database systems. 1-6. 10.1109/ISSC.2017.7983624.