Prevention of SQL Injection Attacks in Web Applications

Vaibhav Srivastava,
UG Students, Dept. of B-Tech, SRM Institute of Science and Technology, Kattankulathur, Chennai, Tamil Nadu, India, Email: vaibhavsrivastava1599@gmail.com

Abhinav Majumdar,
UG Students, Dept. of B-Tech, SRM Institute of Science and Technology, Kattankulathur, Chennai, Tamil Nadu, India, Email: abhi01082000@gmail.com

Jeyasekar A
Associate Professor, Dept. of B-Tech, SRM Institute of Science and Technology, Kattankulathur, Chennai, Tamil Nadu, India. Email: ajeyasekar@yahoo.com

Abstract

SQL injection attacks are the most basic type of cyber-attacks that execute arbitrary malicious code to retrieve confidential information from a SQL database. This paper aims to study and analyze three types of injection attacks and find out their vulnerabilities. Based on that, we propose a new measure to prevent the occurrence of SQL injection attacks. The proposed measure was experimented with and tested using a local webserver and found that it accurately detects and prevents SQL injection attacks. The proposed system could be implemented in the Web Application Firewall to detect and prevent malicious SQL traffic.

Keywords: *SQL Database, SQL Injection Attack, In-Band Injection, Blind Injection, Out-of-Band Injection, SQL Injection Prevention.*

1. INTRODUCTION

Structured Query Language so-called SQL, was first introduced in 1974. It was used to operate the relational database management system (RDBMS) but later became compatible with many other database systems. The language SQL comprises various queries used to organize data in the databases. In recent decades, Injection attacks have enormously increased, leading to the hype about cyber-crimes. An injection is an attack in which malicious code is injected into a script to change its working and get access to restricted data or rights. SQL being a primitive language has also become vulnerable to such attacks and are called SQL Injections. SQL Injection is the infiltration of malicious code into the input query of a SQL statement on any platform that dynamically operates with a live

Database, this injection results in the attacker getting information from a restricted database which might lead to an immoral use of data for any kind of profit. SQLI is one of the most dangerous attacks on any active database application. The first SQLI was recorded in 1998 and since then SQLI is been on top of all security threats. According to OWASP, it is in the top 10 cyber security threat list, from 2017 to 2019 two/thirds of all the attacks were SQL Injection.

The research objective of this paper is to develop an application that detects and prevents the SQL injection attack during the run time, isolating the attacker from accessing the

database and alerting the administrator. In this paper, we will be dealing with all existing SQL vulnerabilities and providing a prevention measure for each of them. The proposed prevention mechanism has five main components: 1) Sanitization, 2) Hashing of confidential data, 3) Sending Notification, 4) Banning and 5) WAF policies.

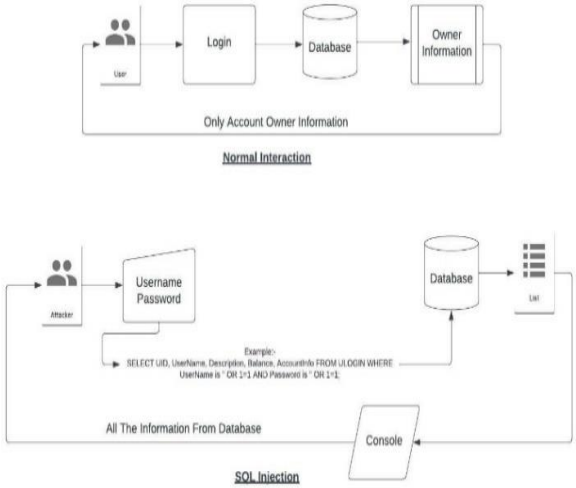
In sanitizing, the admin restricts the usage of special characters with a warning. In hashing the admin encrypts the confidential data. In Notification, alert notices are sent to the admin during breaching and in banning the attacker is restricted to access the website for a certain time. We have inspected the above five components and experimented with the proposed prevention mechanism in the local host using MySQL, PHP Apache web server, and JavaScript. It detects and prevents SQL injection attacks in runtime.

This paper has 6 sections where we are discussing the workflow of a SQL injection in section 2 and how SQL injection is performed, & in section 3 we are stating the prevention method which has been tested and analyzed. In section 4 we analyze all the prevention methods stated in section 3 and their drawbacks. Sections 5 and 6 are the conclusion and references respectively.

2. RELATED WORKS:

Fig 1 shows the user interaction with the database and how the attacker injects the malicious SQL query into the database. Generally, the SQLI has been mainly categorized into 1) Inband/Unsanitized SQLI, 2) Blind/Inferential SQLI and 3) Out-of-band SQLI. The In-Band and Blind are based on traditional SQL attacks. They generally use the same channel to perform an injection attack, i.e., the attacker injects the malicious code into the webserver and retrieves it directly from the database server. While the Out Of Band is a part of a modern injection attack and uses a second channel to retrieve the data. It does so by using a proxy server.

Fig 1. SQLIA Workflow



IN-BAND / UNSANITIZED

These attacks are generally, easily executed and exploited and mostly target a client through the same channel or median.

Consisting of two sub-components

- Error Based:

In this attack, the attacker passes a random query that leads to some specific errors. The attacker then uses that error statement and manipulates it in a certain way to retrieve the data from the E.g., an attacker can know about the number of columns in a specific table using the following order-by query.

```
1' ORDER BY 9--
```

If the number of columns is less than 9 then the attacker will simply get an error that no such column exists. This allows the attacker to know that either the table consists of 8 or fewer than 8 columns.

- Union Based:

In this SQLI, generally, the attacker uses the error statement and executes the further attack with the help of the union operator [4].

This attack uses the information derived from the error-based injection attack and aims to find more information about the table or the data in the table. Below mentioned are some of the union statement queries.

```
1' UNION SELECT 1,NULL ,NULL ,'a','a' ,NULL ,NULL ,NULL ,'a'--
```

This query gives out the probable data type of column so we can use the extended union statement as such and get the desired output. It is a hit-and-trial method and needs to be executed as many times as to get the right dataset.

```
1' UNION SELECT 1,NULL ,NULL ,table_name,'a',NULL ,NULL ,NULL ,'a' FROM
information_schema.columns--
```

This query uses the basic information we derived from error-based Injection and uses the varchar datatype to get information in output as a union query needs the same number of columns as in the parent query. The information schema is a SQL databases table where the database saves information about the database and tables and in this query, we are using this to get all the table names.

```
1' UNION SELECT 1,NULL ,NULL ,table_name,column_name,NULL ,NULL ,NULL ,data_type
FROM information_schema.columns
WHERE table_name='datasetf'--
```

This statement is a further extension of the previous statement. Using the union-based injection the attacker gets the column name and data type of a specific column he/she either aims to attack or suspects to have confidential data in.

```
1' UNION SELECT 1,CustomerID ,NULL ,StockCode,'a',NULL ,NULL ,NULL ,'a'
FROM datasetf--
```

This is the final phase where the information is extracted from the previous statement to get all the data from the table. In this example, the CustomerID and StockCode were given out as a result of injection after checking the correct details from the information schema.

2.1 BLIND / INFERENTIAL

This generally occurs when the backend database considers user input as a query and not data to be saved.

The error message shown by these websites initiates a chance of Blind SQLI. Some common responses by websites result in true and false responses, giving away some confidential information. This type of SQLI is used to gather information about databases and tables.

Consisting of two sub-components

- Boolean Based:

The attacker makes queries that ask the database true and false questions in this type. These questions either return database semantics or all the data of the database.

This type of SQL injection needs some prerequisites, using In-Band SQL injection we can find out the information about the database and table the column name and the data type, etc.

We can then use some SQL Boolean query to get the output may be as simple as,

```
1' OR 1=1--
```

As 1=1 is always true it will show all the fields as a result.

```
1' AND Ascii(substring(database(),1,1)) > 97
```

This gives results in 1 & 0 or true & false form if the first character of the database name is greater than 97 ASCII characters and we thus can hit and try between all the ASCII characters to come to a result.

```
1' AND SUBSTRING((SELECT "column name" FROM "table name"
WHERE "column name" = 'wxyz'), 1, 1) > 'a
```

In this, we are using a common SQL query substring and getting the first word of a specific string from a specific column from a table, from the details we got from In-Band Injection. Then through basic yes or no question, we can predict the specific letter of string if it is greater than ‘a’ or ‘A’ or less than ‘k’ or ‘K’. Thus, we can predict all the string letters, thus getting the forbidden information from the database.

- Time Based:

Time-based delays the response of the site and helps in calculating the true and false results. E.g., if the result to a query to a result is true delay by 10 seconds, and if no then no delay. The most common command is WAITFOR delay.

This injection is used when the webpage isn't showing any visible details and no sign of true or false,

```
1' AND IF(Ascii(substring(database(),1,1)) > 97,sleep(2),0)
```

In this, we are using the SQL substring query to check the database name letter by letter and if it is true then the SQL database will sleep for 2 seconds resulting in a delayed result or web page loading otherwise if false no change in loading time.

```
1' AND IF(SUBSTRING((SELECT Description FROM datasetf WHERE CustomerID = '18055' LIMIT 1), 1, 1) >
'D,sleep(5),0)
```

Or we can check a desired column data letter by letter through hit-and-trial and get a definite result of the content in the column using a similar sleep query.

2.3 OUT-OF-BAND:

In this, the attacker uses outer or external channels to perform the injection attack and collect important data. The whole injection attack is based on three things. First, injecting a malicious query to infiltrate the database data. Second, passing the data through an external channel to the listening server. These channels are mostly of two types DNS (Domain Name System) or HTTP (Hypertext Transfer Protocol). At last, accessing the data via a listening server.

The working of the whole Out-Of-Band SQLI process is based on this one formula:

OOB= Fx. (SQL Command+ SvDN)

Where SQL command is the query used to extract data, SvDN is the subdomain provided by the server and Fx is the function used for passing requests.

1. Using DNS channel

Initially to perform this attack a proxy/listening server is needed. The particular server provides a subdomain to create a DNS lookup. After that, the server fetches all the important information and allows the attacker to retrieve/exfiltrate the data.

2. Using HTTP channel

This attack is similar to the DNS attack with the only difference, that instead of using a file name an HTTP link is provided to initiate an out-band request. After that using the domain and SQL query, the attacks take place and allow the attacker to retrieve the data.

3. PROPOSED PREVENTION MECHANISM

The proposed prevention mechanism comprises five important processes: Sanitizing the input fields by not allowing any special characters in the info field, and hashing confidential data in the database so that the attacker cannot decode the confidential data. Notifying the administrator if an attacker tries to breach the database, isolating the IP address of the attackers so that they will not be allowed in accessing the database, eventually the policies are formulated and set at the Web Application Firewall to avoid the malicious traffic generated by the SQL attacker.

3.1 Sanitizing the input fields:

This method is used when the client's input fields are not properly sanitized or restricted on most websites. E.g., the most common attacking query in an SQLI is '1=1'. This is a Boolean expression that always means true leads to showcase the entire database. This easily allows the attacker to manipulate data. The better way to deal with this is not allowing to use of any special character in the info field. This will restrict the attacker from using any

kind of fishy query which may result in an injection attack.

ALGORITHM:

1. Fetch the input character or string from the input box.

2. Create a database for storing specific information.

3. Check if the string contains any type of special characters.

4. If it contains, then passes a denial message stating invalid input.

5. Even parse the special character if it's typed in the input box.

6. And if not then, simply state-input corrects.

7. Finally store the improvised data in the respective database.

3.2 Hashing confidential data:

Hashing is a form of encryption method that changes the basic structure of any data and stores it in an encoded way. This ensures the confidentiality of data. The encryption is done using various hashing algorithms which can be varied accordingly. Therefore, it becomes difficult for the attacker to decrypt the data and get the important information out of it.

ALGORITHM:

1. Fetch the input strings from the input boxes.

2. Create a database for storing specific information.

3. Ask the user about the confidential data via the message box.

4. Hash the specific approved data using an appropriate hashing function.

5. Check if the data has been hashed or not.

6. Store the hashed data in the respective database.

3.3 Sending Notifications to admin:

If somehow an attacker tries or manages to breach a database the admin should get an alert notification indicating the incident. By this, the admin will get to know about the breaching and will be able to secure the database. The notification can be sent in any format i.e., it can be in the form of mail, an SMS or a voice call, or even all three of them together.

ALGORITHM:

1. Create a database and store admin information.

2. Fetch the inputs from the input boxes.

3. Check whether the entered info matches the ones in the database.

4. If yes then allow the user.

5. Else, then sends the notification to the admin about breaching.

6. If the admin confirms his or her identity then allow it.

7. Else simply deny access to the database.

3.4 Banning IP address:

If an attacker tries to pretend as an admin and is forcefully willing to access the database then this method forbids doing so. In this firstly the verification of the inserted information takes place, if it doesn't match then it gives the user some specific chance to log in. If the user fails to do so, then this method collects the remote IP address of the user and bans it for a specific period. This forbids the user to enter the site until the ban is not removed.

ALGORITHM:

1. Create a database with columns such as IP, ban time, and count.

2. Fetch the user inputs and check if it exists or not.

3. If so, then allow access to the user.

4. Else, give a limited trial to the user and store it in the count column.

5. Also fetch the user's Ip.

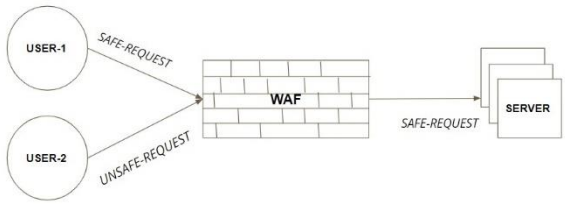
6. As the count increases the limit place a ban time to the respective Ip.

7. Store the Ip and count limits in their respective columns.

3.5 Using WAF:

WAF stands for web application firewall. The main usage of WAF is it acts as a shield between a web app and the internet. It generally observes and examines all types of data that passes through the server. If any problem occurs it blocks the malicious traffic. The WAF functions by specific rules called policies. It determines what the threats are and filters them out. Along with SQL injection, it also prevents cross-site forgery and cross-site scripting (XSS).

Fig 2. Filtering safe requests using a firewall.



4. ANALYSIS OF PROPOSED MECHANISM

The main concept of the sanitization algorithm is to forbid the user from preventing him to use any sort of malicious queries that may lead to basic injection attacks. It not only denies any sort of usage of a special character but also automatically removes it if typed. The algorithm only rectifies the typing of any special characters. If the attacker tries attacking with a Boolean query, then the code will simply not work and the attack will eventually occur. In the hashing algorithm, the PHP hash function is used which is based on the concept of bcrypt. The uniqueness of bcrypt is that it always creates a 12 key character that the user wants to hash. Even if the attacker tries to use the same hashing algorithm and decode the password, he/she won't be able to do so because of the one-way property of hashing algorithm The major drawback of using hashing algorithm in the proposed mechanism is that if in any case, the user forgets his/her password then the whole info about the user needs to be deleted and a new one should be introduced.

5. CONCLUSION:

In this paper, we have experimented with & executed different types of SQL Injection Attacks. To perform this initially we have gone through every aspect of these attacks and studied their workflow. We also tested & implemented their work on our localhost and evaluated their drawbacks and vulnerabilities. On its basis, we have studied and implemented some of the existing and also created some new algorithms for preventive measures. These algorithms are general measures of blocking any SQLI. Our future work will majorly focus on finding out new ways to counter all types of attacks within a single application. The existing work will help in pushing the remaining forward.

References

[1] XuePing-Chen "SQL injection attack and guard technical research", Procedia Engineering 15 (2011), 4131 – 4135

[2] Vivek Sharma, Subodh Mishra," SQL Injection Prevention by Blocking Internet Protocol Address after Analyzing Error Message of Database Server", International Journal of Computer Science and Information Technologies, Vol. 7 (2), 2016, 566-569

[3] Raghuraman. K, R.Venkatesan, Rubidha Devi.D," STUDY ON SQL INJECTION TECHNIQUES", IJPTFI, 12.11.2016, ISSN: 0975-766X

[4] Ashish Kumar, Sumitra Binu, "Proposed Method for SQL Injection Detection and its Prevention", International Journal of Engineering & Technology, 7 (2.6) (2018), 213-216

[5] Mohd Amin Mohd Yunus ETAL, "Review of SQL Injection: Problems and Prevention", INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION, 2549-9610, VOL 2 (2018), NO 3 – 2

[6] "SQL documentation" from https://dev.mysql.com/doc/ downloaded on 05/04/2022

[7] "Biggest Threat to Application Security: SQL Injection Attacks" from https://www.appknox.com/blog/sql-injection-attacks

[8] Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan "A SURVEY ON SQL INJECTION: VULNERABILITIES, ATTACKS, AND PREVENTION TECHNIQUES", IEEE 15th International Symposium on Computer Electronics, 2011, No 468-471

[9] Haifeng Gu ETAL, "DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data", IEEE TRANSACTIONS ON RELIABILITY, VOL. 69, NO 1. MARCH 2020, No 188-202