Preventing SQL injection Attacks Using Cryptography Methods

Dinu P S¹, Deepa S Kumar², Dr. M. Abdul Rahman³
¹(PG scholar, Department of computer science, College of Engineering Munnar, Kerala, India)
²(Research scholar, Karpagam University, Coimbatore, India)
³(Pro Vice Chancellor, Kerala Technological University)

ABSTRACT
SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The attack takes advantage of poor input validation in code and website administration. It allows attackers to gain unauthorized access to the backend database to change the intended application generated SQL queries. Researchers have proposed various solutions to address SQL injection problems. However, many of them have limitations and often cannot address all kinds of injection problems. What’s more, new types of SQL injection attacks have arisen over the years. To better counter these attacks, identifying and understanding existing techniques are very important. In this research we present all SQL injection attack types and a technique and tools which can detect or prevent these attacks by the method of Hashing and Encryption.

Keywords - SQL injection attacks, Web application, Hashing and Encryption

1. INTRODUCTION
Internet has become one of the most powerful and interesting tools all across the world. Nowadays there is a plethora of web applications that cover a wide range of daily needs. A large number of electronic transactions, including e-commerce, e-banking, e-voting, e-learning and e-health among others, can be conducted online at any time and from anywhere. However, all these Internet applications are exposed to hacking attempts, security-related problems which is a major threat. SQL injection represents the most common indirect attack technique against web-powered databases and can effectively disassemble the secrecy, integrity and availability of web applications. SQL injection happens when an attacker inserts malicious SQL code into an SQL query by manipulating data input into an application. This kind of vulnerability is a serious threat to any web application that reads input from users and uses it to build and execute SQL queries to an underlying database. With SQL injection, the attacker can run arbitrary SQL queries, extracting sensitive customer and order information from e-commerce applications, or can bypass strong security mechanisms compromising the back-end databases and the data server file system. In the face of these threats, a surprisingly high number of systems on the Internet are entirely vulnerable to such attacks, leaving even experienced professional programmers unable to cover all possible SQL injection techniques.

SQL injection refers to a class of code-injection attacks in which data provided by the user is included in a SQL query in such a way that part of the user’s input is treated as SQL code. SQLIAs are a type of vulnerability that is ultimately caused by insufficient input validation—they occur when data provided by the user is not properly validated and is included directly in a SQL query [1]. By leveraging these vulnerabilities, an attacker can submit SQL commands directly to the database. This kind of vulnerability represents a serious threat to any web application that reads input from the users (e.g., through web forms or web APIs) and uses it to make SQL queries to an underlying database. Most web applications used on the Internet or within enterprises work this way and could therefore be vulnerable to SQL injection. Although the vulnerabilities that lead to SQLIAs are well understood, they persist because of a lack of effective techniques for detecting and preventing them. Programming practices such as defensive programming and sophisticated input validation techniques can prevent some vulnerability. However, attackers continue to seek new exploits that can avoid the checks programmers put in place. Moreover, defensive programming is labor-intensive, which makes it an impractical technique for protecting large legacy systems. General tools such as firewalls and current Intrusion Detection Systems (IDSs) are also typically ineffective against SQLIAs—SQLIAs are performed through ports used for regular web traffic (usually open in firewalls) and work at the application level (unlike most IDSs). Finally, most analysis-based techniques for vulnerability detection do not address the specific characteristics of SQLIAs and are thus ineffective in this context. The few analysis techniques specifically designed to target SQLIAs provide only partial solutions to the problem [2]. In particular,
dynamic techniques, such as penetration testing, introduce issues of completeness and often result in false negatives being produced, whereas techniques based on static analysis are either too imprecise or only focus on a particular aspect of the problem.

SQL Injection is one of the main issues in database mechanism such as Integration, Authentication, Availability. It affects the database without the knowledge of the authorization. It is a technique used to exploit the database Injection vulnerabilities (SQLIVS) and, therefore, 10% of total system through vulnerable web applications. All the other possible injections are quoted SQL changes to both the database schema and the contents.

2. WHAT IS SQL INJECTION?

SQL Injection is one of the many web attack mechanisms used by hackers to steal data from databases. It is perhaps one of the most common application layer attack methodology used today. It is the type of attack that takes advantage of improper implementations of your web applications that allows hacker to inject SQL commands into say a login form to allow them to gain the access to the data held within your database.

In essence, SQL Injection arises because the fields available for user input allow SQL statements to pass through and query the database directly. Web applications allow legitimate website visitors to submit and retrieve data to and from a database over the Internet using their preferred web browser. Databases store data needed for websites to deliver specific content to visitors and render information to clients, customers, suppliers, employees and a host of stakeholders. User credentials, financial and payment information, secret passwords, company statistics may all be resident within a database and accessible by legitimate users through off-the-shelf and custom web applications. SQL Injection attack is the hacking technique which attempts to pass SQL commands through a web application for execution by the backend database. If not implemented properly, web applications may result in SQL Injection attacks that allow hackers to view and collect information from the database.

2.1 SIMPLE EXAMPLE

Take a simple login page where a legitimate user would enter his username and password combination to enter a secure area to view his personal details or upload his comments and details in a forum. When the legitimate user submits his data, an SQL query is generated from these data and submitted to the database for verification. If valid, the user is allowed access the system. That is, the web application that controls the login page will communicate with the database through a series of planned commands so as to verify the username and password combination. On verification process, the legitimate user is granted appropriate access. Through SQL Injection, the hacker may input specifically created SQL commands with the intent of bypassing the login form barrier and seeing what lies behind it. This is only possible if the inputs are not properly implemented (i.e., made invulnerable) and sent directly with the SQL query to the database [3].

SQL Injection vulnerabilities provide the means for an attacker to communicate directly to the database.

2.2 DIFFERENT TYPES OF VULNERABILITIES

The SQL injection attacks can be done through various techniques. Some of them are specified below [4] [5]:

2.2.1 TAUTOLOGY ATTACK

The main objective of tautology-based attack is to inject code in conditional statements so that they are always evaluated as true. Using tautologies, the hacker wishes to either bypass user authentication or insert inject-able parameters or extract data from the database [6]. A typical SQL tautology has the form, where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. Bypassing authentication page and collecting data is the most common example of this kind of attack. In this type of attack, the attacker exploits an inject-able field contained in the “WHERE” clause of query. Attacker transforms this conditional query into a tautology and hence causes all the rows in the database table targeted by the query to be returned [7]. For example:-

```
SELECT * FROM user WHERE id='1' or '1'=1'-'AND password='1234';
```

or 1=1” the most commonly known tautology.

2.2.2 LOGICALLY INCORRECT QUERY ATTACKS

The main objective of the Illegal/Logically Incorrect Queries based SQL Attacks is to gather the information about the back end, database of the Web Application. When a query is rejected, an error message is returned from the database which includes useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application [8]. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical error by purpose. In this example attacker makes a type
mismatch error by injecting the following text into the input field:
1. Original URL: http://www.toolsmarket-al.com/veglat/?id_nav=223455
2. SQL Injection: http://www.toolsmarket-al/veglat/?id_nav=223455'
3. Error message showed: SELECT name FROM Employee WHERE id=223455'. From the message error we get the name of table and fields: name; Employee; Id By the gained information attacker can organize more perfect attacks. The Illegal/Logically Incorrect Queries based SQL attack is considered as the basis step for all the other attacking techniques.

2.2.3 UNION QUERY
In this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application.
Example: Following executed from the server:

```sql
SELECT name, phone FROM tbl_user WHERE userid=$id1
```
By injecting the following Id value into:

```sql
$id1= 1 UNION ALL SELECT credit Card Number, 1 FROM Credit CardTable
```
Then we will have the following query:

```sql
SELECT name, phone FROM tbl_user WHERE userid=1 UNION ALL SELECT creditCardNumber, 1 FROM Credit CardTable
```
This will join the result of the original query with all the credit card users to the attacker. The proposed implemented system contains the mechanisms, which will protect the web application from the above discussed SQL injection attacks.

3. PROPOSED APPROACH
Working of the proposed technology can be classified into two:
1. Hashing:
2. Encryption:
   In the proposed approach there is a need for one extra column in database, which contains the EX-OR of the Hash values of username and password at the time, when a user account is created for the first time and stores it in the User table. Whenever user wants to login to database his/her identity is checked using user name and password and its hash values. These hash values are calculated at runtime using store procedure when user wants to login into the database [9]. During the authentication of user, the SQL query with hash parameters is used. Hence, if a user tries the injection to the query, and our proposed methodology is working with SQL query, it will automatically detect the injections as the potentially harmful content and rejects the values. Hash function is a function which is used to generate digital data of different size to a fixed size. The values returned by the hash function for the given date are called the hash code. In hashing, once we create a hash code, it cannot be retained. Which provide the security for login process. So here we use hashing for preventing before login attacks.

3.1 HASHING ALGORITHM
Message-digit algorithm commonly known as MD5 algorithm is wildly used for generating the hash code [10]. It producing a 128 bit hash value, typically found in a text format as a 32 digit hexadecimal number. MD5 algorithm is used for the wide verity of the cryptographic applications.
MD5 algorithm consists of 5 steps:
Step 1. Appending Padding Bits.
The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:
4. The original message is always padded with one bit "1" first.
5. Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.
Step 2. Appending Length.
64 bits are appended to the end of the padded message to indicate the length of the original message in bytes. The rules of appending length are:
- The length of the original message in bytes is converted to its binary format of 64 bits. If overflow happens, only the low-order 64 bits are used.
- Break the 64-bit length into 2 words (32 bits each).
- The low-order word is appended first and followed by the high-order word.
Step 3. Initializing MD Buffer.
MD5 algorithm requires a 128-bit buffer with a specific initial value. The rules of initializing buffer are:
- The buffer is divided into 4 words (32 bits each), named as A, B, C, and D.
- Word A is initialized to: 0x67452301.
- Word B is initialized to: 0xEFCDAB89.
- Word C is initialized to: 0x98BADCFE.
- Word D is initialized to: 0x10325476.
This is the main step of MD 5 algorithm, which loops through the padded and appended message in blocks of 512 bits each. For each input block, 4 rounds of operations are performed with 16 operations in each round.

Step 5. Output.
The contents in buffer words A, B, C, D are returned in sequence with low-order byte first.

3.2 ENCRYPTION

Here we use the AES encryption method to encrypt the data stored in the database. Consider a scenario, where a user is authenticated by the secure login mechanism and login the system. Now, if this authenticated user make any intrusion into the system [11]. How can we defend it? Hence to prevent after-login attacks we have taken the help of data encryption. As we saw in the previous section that it is possible to collect the highly confidential information by using union operator we find out an alternative way to store all these confidential information’s. In our database, instead of directly storing all confidential information’s, we store them in encrypted format with a secure and confidential encryption-key. Now even if the dispatcher user can able to see the atm pin by using union operation, he cannot able to decrypt it without knowing the exact encryption method and encryption-key. So he cannot able to do any damage with that encrypted atm pin.

4. CONCLUSION

The proposed a new approach that is completely based on the hash method of using the SQL queries in the web-based environment, which is much secure and provide the prevention from the attackers SQL. But, our proposed strategy requires the alterations in the design of existing schema database and a new guideline for the database user before writing any new database. Through these guidelines, we found the effective outcomes in SQL injections Preventions.

REFERENCES