Robust SQL Injection Prevention using Adaptive Method

Pavleen Segue
Student
Shah & Anchor Kutchhi
Engineering College, Mumbai
+91 9930507141
pavleen7@gmail.com

Sumeet L Ram
Student
Shah & Anchor Kutchhi
Engineering College, Mumbai
+91 8976661463
sumeetlaljeet@gmail.com

Priyanka Vallakati
Student
Shah & Anchor Kutchhi
Engineering College, Mumbai
+91 8286289225
priyv95@gmail.com

***Abstarct*:-** SQL injection targets one of the most important parts of any application and that is its database. It is a powerful method of getting access to a database and/or modifying the sensitive data present in the database. To prevent this and provide a high end security to the applications present on the internet, this paper proposes a method where a sample website is made to operate under normal conditions and the queries which are received here are transformed into their structural form according to the query scheme used which is extended query transformation in this case. This way a prototype is built and this is then subjected to actual usage at runtime where SQL injection might happen and when such a condition occurs this prototype is ready with a table of legitimate queries and whenever there is an input by the user, the query is brought down to its basic structural form irrespective of its complexity. Then this structural form is compared with the structural forms of the queries present in the secure_queries table. This scanning and fetching of queries is done with the help of hashing to make the method more efficient and faster and this method uses SHA2 hashing algorithm as it gives less collision rates.

Keywords:- *Security; database; web application; SQL injection; extended query transformation; hashing; SHA.*

*****

1. INTRODUCTION

Ever since the development of computer technology, websites went from merely being static pages of texts and images to dynamic web applications of custom built content. In December 2001, Open Web Application Security Project (OWASP) was founded as an international organization which aimed at web security discussions, enhancements and kept track of every possible hack that could be performed. Web applications are accessible to everyone and hence are vulnerable to various security threats [8] such as Cross-site scripting, Session hijacking, Buffer overflow, Denial of Service, SQL Injection etc. Nowadays most web applications are database driven. SQL injection is a common attack against database, which is an organized collection of data and supporting data structures and has been in the list of the top-10 security threats listed by OWASP [9] since last several years. SQL injection is a code injection technique where SQL query is injected by a malicious attacker as input, with the motive of exposing the back end database and retrieving sensitive information such as credit card details, customer information, social security numbers etc. It can cause damage to a great extent. Being an old problem which is an issue till date, there has been a substantial amount of work done to detect and prevent these SQL injection attacks by researchers. Despite the large amount of research that has been done in this area, most of the techniques proposed or implemented are computationally loaded or strenuous to deploy in production environments. This paper proposes an innovative technique to prevent SQL injection attacks by using an advanced extended [12] query transformation approach followed by an efficient hashing algorithm [5] i.e. the SHA 256/512 which ascertains an effective method to detect and prevent a variety of SQL injection attempts. The rest of the paper is organized in the following manner. Section 2 discusses the background and explains the SQL injection attack and its types. Section 3 outlines the related work in this area and compares the technique utilized in the paper. In Section 4, the key idea behind the proposed extended query transformation scheme and the hashing approach is explained. Conclusion and the references used are present in Section 5 and 6 respectively.

1.1 Objective

To make web applications more secure from the malicious attackers who try to access data from the backend using SQL Injections and retrieve sensitive information without any access to the source code.

1.2 Problem Statement

The method provides a solution to the different inconveniences caused to the data security of web applications.

Each SQL Injection attack is performed with a specific motive.

They are:

1) Add/Modify data – The main motive of this attack is to add or modify the information in a database.

2) Extract data – This attack uses techniques to catch valuable data values from the database.

3) Perform DoS – This attack shuts down the database of the website and denies service to any user.

4) Bypass authentication – This attack passes the rights and privileges of the application and database to the attacker.

5) Execute commands – This attack allows attackers to execute remote arbitrary commands on the database.

1.3 Methodology used

The method proposed uses an innovative extended query transformation technique to considerably reduce response time
and using the more favorable hashing technique i.e. SHA 256/512 to build a hash value for the inserted query. Here, the approach consists of preparing a model which has a list of the secure queries which are most likely to be used in a secured environment and then utilize the model at run-time to prevent SQL injection attempts. Any modification or enhancement to the application code requires rebuilding the normal-use model,
which is a major disadvantage of this approach. Further, these are mostly designed to protect a single web application and usually suitable for a specific language and database platform.

2. BACKGROUND

2.1 SQL Injection

SQL injection is a code injection technique that exploits the security of the vulnerable web applications which occurs in the database of the applications. Any website can have a form which is designed to accept some data from the user, process it, and return some output. A hacker uses the opportunity of a web form to execute malicious SQL code with the intention of
breaking into, altering or damaging the database or in an attempt to retrieve confidential information from the system. For example:

When a user inputs a user name and password, the web application determines if these credentials are accurate or not. This allows the user to access or deny the portal. If the user name is "ABCDE" and the password is "12345", the web application sends an SQL query to the database to verify the credentials:

SELECT *

FROM users

WHERE name = ‘ABCDE’

AND password = ‘12345’

Suppose the attacker enters something like “pqrs” OR 1=1 --" instead of the username and anything else as password. In this case the SQL query will be as follows:

SELECT *

FROM users

WHERE name = ‘pqrs’ OR 1 = 1--‘

AND password = ‘xxxxx’

The above SQL statement will always return a true value because:

1) Name = ‘pqrs’ OR 1 = 1 - -‘will always return a true statement (1 OR 1 = 1).

2) The rest of the SQL Statement after the “--”sign is commented out i.e. that part of the query is not executed. Since the query returns a true value, the attacker was able to trick the application and hence it gained access of the database without the need to guess the credentials of any original user which would have been a tedious job.

2.2 Types of SQL Injection Attacks

The different types of SQL attacks are generally not performed in isolation; many of them are used together or successively depending on the motive of the attacker. Also there may be a number of variations of each attack type. Types
of SQL injection are:

1. *Union-Based Queries:*

UNION-based attacks allow the user to extract information from the database. Because the UNION operator can only be used if both queries have the exact same structure, the attacker
must craft a SELECT statement similar to the original query.
SELECT name, description

FROM products

WHERE category=1

UNION

SELECT 1 FROM information table

2. *Tautologies:*

It is based on insert code in one or more conditional query so that they always evaluate to true. SQL injection is based on 1=1 is always true in which an attacker give [‘‘ or 1=1 - -] for the user name input. The resulting query is: SELECT status, user_name

FROM table_user

WHERE user_name = ’’ or 1=1

3. *Logically Incorrect:*

It uses error messages in order to retrieve information. If the attacker inserts the following words into input field pin:“convert(int,(select top 1 sys_name from sysobjects where user_type=’u’)) ”. The resulting query is:

SELECT user_accounts

FROM users

WHERE login=’’AND pass=’’

AND pin=convert (int,(select top 1 sys_name from sysobjects where user_type='u'))

4. Piggy backed queries:

These are additional queries added to the original queries. The additional query is attached to the original query using the
query delimiter (;). In this the normal SQL statement is followed by additional query by using ";". Normal SQL statement + ";" + INSERT (or UPDATE, DELETE, DROP) <rest of injected query>. The query below is an example of piggy backed queries:

SELECT info

FROM userTable

WHERE login=`name ` AND pin=0;

drop table users

5. Stored Procedure:

An attacker can inject another stored procedure as a replacement for a normal stored procedure to perform privilege escalation, create denial of service, or execute remote commands. Here is a common form using a query delimiter (;) and the "SHUTDOWN" store procedure for this attack:
Normal SQL statement + "; SHUTDOWN; " < rest of injected query>. The query below is an example of attack using stored procedure:
SELECT name

FROM authors

WHERE username=`Jay` AND password=`abc`;

SHUTDOWN; --

6. Inference Attack:

In this the query gets executed only if the given condition is satisfied else it returns the other value. If the condition is not
met the error will be generated. The query below is an example of inference attack:

SELECT name, email

FROM members WHERE id=1

3. RELATED WORK

The base of this method has been originally proposed and implemented in, "Prevention of SQL Injection Attack Using Query Transformation and Hashing" [1] by D. Kar and P. Suvasini. The paper [1] uses a query transformation technique to transform queries into their structural form and then uses hashing to generate unique hash keys for each legal query generated. The hashing algorithm used is MD5 which is a broken algorithm as of now [7]. The stored queries are searched using the hashing algorithm as the algorithm would generate a unique hash key which would point to the transformed query. However, this approach cannot prevent second order SQL injection attempts since the parameter values are removed during the transformation process. There have been other methods proposed for the prevention of SQL injection such as:

1. "Secured Web Application Using Combination of Query Tokenization and Adaptive Method in Preventing SQL Injection Attacks" [3]. It uses query tokenization in which string length is checked. If it matches then false value is returned else the query gets a true value and implements an adaptive method for security. Query tokenization is when a query is parsed and the length of the character is checked using character list and then with association list to load the main page. It detects the spaces, single quotes and double dashes and all strings before each symbol represents a token. The token is arranged into the original injected query and the lengths are compared to check the attacks.

2. "SQL Injection Attack Prevention Based on Decision Tree Classification" [4], is used to prevent SQLIA's. The decision tree is used to find out the class attack. The Http request is filtered using the decision tree. If the response is favorable i.e. positive then it give access to database else the service is denied.

3. "SQLrand: Preventing SQL Injection Attacks" [2] is a technique where there is a proxy between the client and server that de-randomizes the client requests. The parser recognizes the randomized queries and gives access to the database. But if it is an attack then the parser ceases to identify the randomized query and rejects it.

This paper uses an improvised version of the query transformation and that is the extended query transformation technique where all kinds of symbols and different special characters are converted [12]. The hashing technique used here is called Secure Hash Algorithm i.e. SHA (256/512) which is collision resistant and a robust [13] to hash the input having a fixed length key. Unlike MD5 which is no longer secure as a cryptographic hash function [7] as there are tons of rainbow tables against MD5 which are easy to find. The CMU Software Engineering Institute considers MD5 essentially "cryptographically broken and unsuitable for further use" [10]. When the user will enter its login id and password, the query will be transformed using an extended query transformation technique so as to normalize the query into its structural form. In this way, any SQL query, regardless of its complexity, is transformed into a series of words separated by spaces and the structural form is properly maintained. This input is then hashed using the hashing algorithm which is a fixed length key. The hashed key is matched with the stored queries in the backend database. The generated hash key points to the first row of the transformed query table in the database and the search is done. If the match is found then the user gets an access otherwise the access is denied. pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

4. PROPOSED WORK

4.1 Query Transformation

Query transformation is a set of techniques used to rewrite a query & optimize it for better. A three step process that transforms a query of relational SQL into its equivalent and more efficient level query of relational algebra is as follows:

4.1.1. Parse and translate

Check syntax and verify relations. Translate the query into its equivalent relational algebra expression.

4.1.2. Optimize

Generate an optimal evaluation plan for the query.

4.1.3 Evaluate

The query execution engine evaluates a plan. Executes the plan and returns the answers to the query.

The approach must assume that the attacker will try different methods to change the structure by inserting SQL keywords, special characters and alpha numeric values. As the
paper uses query transformation as the base, Table I compares the different approaches of collecting the queries. It should be noticed that this comparison is based on the articles and not empirically experienced.

Table 1. Comparing Query Transformation Techniques

Properties	Comparison of three techniques		
	Collecting all legitimate queries	Skeleton queries	Query Transformation
Storage	Very Large	Moderate	Less
Runtime Feasibility	Not feasible	Feasible	Feasible
Speed	Practically very less	Moderate	Fast

This paper has developed an extended version of the query transformation scheme [12] so that it converts an SQL query from a parameterized form into its structural form. In the extended scheme, all symbols and special characters are also converted into words i.e. all capital A-Z letters for all tokens and space character as token separator.

Table 2. Transformation Special Chars & Symbols

Symbol	Transformed Keyword	Symbol	Transformed Keyword
!= or <> Not Equals	NEQ	(	LPRN
& & Logical AND	AND	)	RPRN
|| Logical OR	OR	{	LCBR
~	TLDE	}	RCBR
!	EXCLM	[	LSQBR
@	ATR	]	RSQBR
#	HASH	\	BSLSH
$	DLLR	:	CLN
%	PRCNT	;	SMCLN
^	XOR	“	DQUT
&	BITAND	‘	SQUT
|	BITOR	<	LT
*	STAR	>	GT
-	MINUS	,	CMMA
+	PLUS	.	DOT
=	EQ	?	QSTN
		/	SLSH

Table 3. Query Transformation Scheme

Sr. No	Value	Transformation Scheme
1.	Newline characters (\r or \n) if any	Remove
2.	Inline Comments (/*. . . */) if any	Remove
3.	Anything within single/double quotes (a) Hexadecimal value HEX (b) Decimal value DEC (c) Integer value INT (d) IP address IPADDR (e) Single alphabet character CHR (f) General string (none of the above)	HEX DEC INT IPADDR CHR STR
4.	Anything outside single/double quotes (a) Hexadecimal value (b) Decimal value (c) Integer value (d) IP address	HEX DEC INT IPADDR

5.	System objects (a) System databases (b) System tables (c) System table column (d) System variable (e) System views (f) System stored procedure	SYSDB SYSTBL SYSCOL SYSVAR SYSVW SYSPROC
6.	User-defined objects (a) User databases (b) User tables (c) User table column (d) User-defined views (e) User-defined stored procedures (f) User-defined functions	USRDB USRTBL USRCOL USRVW USRPROC USRFUNC
7.	SQL keywords, functions, reserved words	To Uppercase
8.	Any token/word not transformed so far (a) Single alphabet (b) Alpha-numeric without space	CHR STR
9.	The entire query	To Uppercase
10.	Multiple Spaces	Single Space

To explain query transformation, consider an injected query generated according to a planned attack to bypass detection:

SELECT *

FROM products

WHERE prod_id = 24 OR 'ABC' = CoNcAt (char (0x28 + 25), char (80 - 0x0d))

This query contains a number of symbols and operators. The transformation scheme converts it completely into text form as:

SELECT STAR FROM USRTBL WHERE USRCOL EQ INT OR SQUT STR SQUT EQ CONCAT LPRN CHAR LPRN HEX PLUS INT RPRN CMMA CHAR LPRN INT MINUS HEX RPRN RPRN

Therefore any SQL query, regardless of its complexity, is hence transformed into a series of words separated by spaces. The structural form of query is correctly maintained by this query transformation approach.

4.2 Hashing

Process of finding an element within the list of elements in a particular order or randomly is called hashing. Hashing is the conversion of string of characters into a usually shorter fixed length value or key that depicts the original string. Hashing is used to index and retrieve objects in the database because objects can be found quickly using shorter hash keys than finding using the original values. The basic process is as follows:

1. The user queries the values of interest.
2. These values are then transformed into a hash key.
3. The database engine finds the index on the hashed column and returns the required row or a small subset of rows that match.

PHP offers a built-in function hash function for SHA. The first argument to the function is the algorithm name and the second argument is the string that will be hashed. The result it returns is the hashed string. There are several well-known hash functions used in cryptography. These include the Message Digest (MD) hash functions i.e. MD2, MD4 and MD5, used for hashing digital signatures into a shorter value called a message-digest. There is also the Secure Hash Algorithm (SHA), a standard algorithm that makes a larger message digest and is similar to MD4. Researchers have found several flaws in the SHA1 and MD5 algorithms. Hence, hash algorithms from the SHA 2 family like SHA256 or SHA512 can be used. As the name suggest, they produce hashes of length 256 and 512 bits. They are newer and considerably stronger than MD5. As the number of bits increase, the probability of a collision decreases. Note that all the hashing algorithms are not compared but as MD5 has been broken, the table gives a comparison between MD5 and a more secured hashing algorithm SHA 2.

Table 4. Comparing Hashing Techniques

Properties	**Key differences between MD5 and SHA2**	
	MD 5	SHA 2
Message Length	128 bits	256/512 bits
Speed	Faster, only 64 iterations	Slower than MD5, Required 80 iterations
Security	Less Secure than SHA	Higher Security than MD5
Attack	Attacks reported to some extents	No such attack reported yet

This paper proposes to use SHA-256 or SHA-512 [13] as there are number of limitations and security issues in other SHA algorithms and SHA- 256 and SHA-512 are the strongest among the SHA-2 hashing algorithms.

Figure 1 is the flowchart for the proposed method to detect and prevent SQL injection. It shows the Systematic flow of the whole procedure that would take place once the user enters the required input.

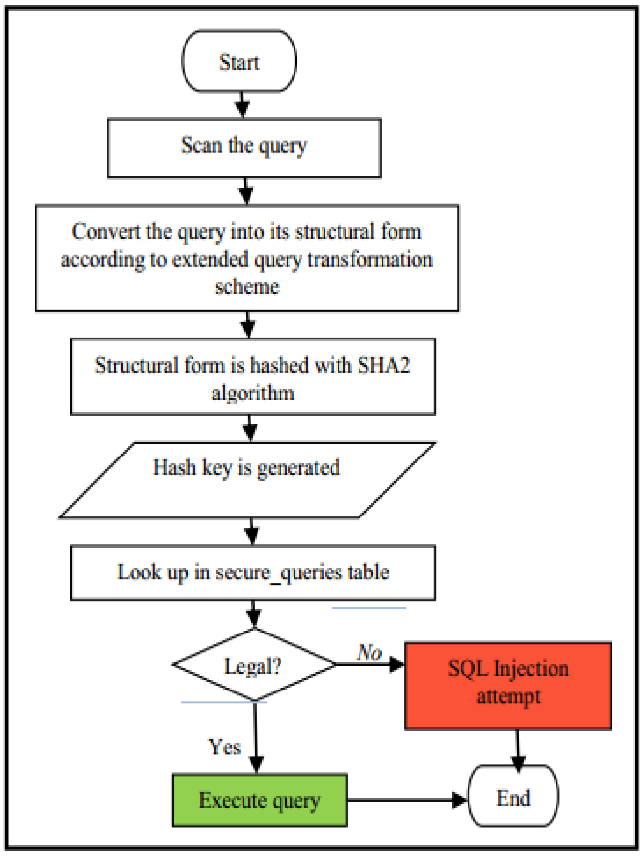
Start
Scan the query
Convert the query into its structural form according to extended query transformation scheme
Structural form is hashed with SHA2 algorithm
Hash key is generated
Look up in secure_queries table
Legal?
No
SQL Injection attempt
Yes
Execute query
End

Figure 1. Flowchart for Proposed Model

5. CONCLUSION

The proposed model is very useful in preventing different types of SQL injection attacks. The main contribution of this paper is the use of a new extended query transformation scheme. Here, a website operates under normal conditions so that the database contains a list of secure queries. The structural form of all the queries that is obtained here is done by using extended query transformation which is an improvised version of the query transformation technique. Then this prototype is subjected to SQL injection at runtime and it finds out if the structure of the input query at runtime matches with the structures of the different queries that are present with the prototype and this fetching of queries is done by using hashing algorithm SHA 256/512 which is more secure than other hashing algorithms such as MD5, MD4.

6. ACKNOWLEDGMENTS

The technical paper has been one of the significant academic challenges and without the support, patience and guidance of the people involved, this task would not have been completed.
Firstly we would like to express profound gratitude towards our Principal, Prof. Vinit Kotak for extending his support in all our endeavors and Ms. Swati Deshpande, head of the Department of I.T for her guidance and mentorship throughout. It has been a privilege to have Prof. Smita Bansod as the project guide who has been extremely patient and supportive since the beginning and guided us in the right direction.
Last but not the least, we would like to express our gratitude to Pragyanam'17 for providing this platform. We acknowledge our sincere gratitude to all those who helped us to make this literature review a success.

REFERENCES

[1] D. Kar and P. Suvasini, "Prevention of SQL Injection Attack Using Query Transformation and Hashing," in Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013, 2013, pp. 1317–1323.
[2] SQLrand: Preventing SQL Injection Attacks – Stephen. W. Boyd and Angelos D. Keromytis, Department of Computer Science Columbia University {swb48,angelos}@cs.columbia.edu
[3] "Secured Web Application Using Combination of Query Tokenization and Adaptive Method in Preventing SQL Injection Attacks" by Noor Ashitah Faculty of Comp &Mathematical Science
[4] "SQL Injection Attack Prevention Based on Decision Tree Classification" - B.Hanmanthu B.Raghu Ram, Kakatiya Institute of Technology & Science, Warangal, Telangana, India.
[5] A Comparative Analysis of SHA and MD5 Algorithm Piyush Gupta, Sandeep Kumar Department of Computer Science and Engineering Jagannath University, Jaipur
[6] Vulnerability of software integrity and code signing applications to chosen prefix collisions for MD5 by Marc Stevens and Arjen Lenstra.
[7] Article on "MD5 considered harmful today" by Alexander Sotirov, Marc Stevens and Jacob Appelbaum.
[8] Article based on "6 threats to web applications and how to avoid it" at http://www.commonplaces.com/blog/6-threats-to-web-applicationsecurity-&-how-to-avoid-it/
[9] OWASP Top 10 Threats at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013
[10] Article based on "MD5 Collision Demo" published on Feb'06 by Peter Selinger.
[11] Information on "SQL Injection vulnerability" at https://www.netsparker.com/web-vulnerability-scanner/vulnerabilitysecurity-checks-index/sql-injection/
[12] SQLiDDS: SQL Injection using Query Transformation and Document Similarity by Debabrata Kar and Suvasini Panigrahi.
[13] SHA-512/256 Shay Gueron 1, 2, Simon Johnson 3 , Jesse Walker4 1 Department of Mathematics, University of Haifa, Israel