Volume 4, Issue 4, April 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization

S.Anjugam
Department of Computer Applications
SRM University, Chennai,India

A.Murugan
Department of Computer Science and Engineering,
SRM University, Chennai, India

Abstract : Web applications are increasingly used in recent years to provide online services such as banking, shopping, social networking, etc. These applications operate with sensitive user information and hence there is a high need for assuring their confidentiality, integrity, and availability. This paper focused on how to detect and prevent SQL injection attacks on web applications using encryption and tokenization technique. The tokenization process is applied on the input query by detecting spaces, single quotes and double dashes etc. This process converts the input query into fruitful tokens on both client and server side and that are stored in a separate dynamic tables. Both tables are compared, if they are differnt, query is rejected and not forwarded to the database server.Otherwise, the query is proceed further to main database for retrieving result .It has better performance and provides increased security in comparison to the existing solutions. The goal of this paper is to provide improved security by developing a method which prevents illegal access to the database.

Keywords: SQL Injection,SQL Detection, Classification of SQLIA, Query Tokenization .

I . Introduction

The main intent to use SQL injection attack includes illegal access to a database, extracting information from the database, modifying the existing database, escalation of privileges of the user or to malfunction an application. Ultimately SQLIA involves unauthorized access to a database exploiting the vulnerable parameters of a web application.

A. SQL Injection

In order to locate the hotspots where SQLIA vulnerability occurs,we first discuss about the 3-tier logical view architecture of web applications which is shown in fig.1[3].

i). 3-tier Architecture of web application: 1) User interface tier: This layer forms the front end of the web application. It interacts with the other layers based on the inputs provided by the user. 2) Business logic tier: The user request and its processing are done here. It involves the server side programming logic. Forms the intermediate layer between the user interface tier and the database tier. 3) Database tier: It involves the database server. It is useful in storage and retrieval of data.

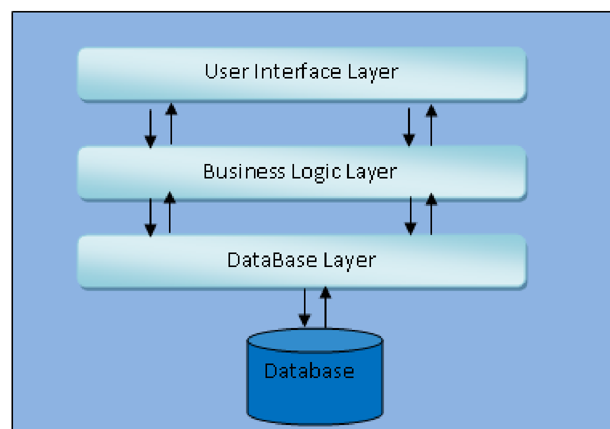

Fig. 1. Three Tier Architecture

ii).There are two types of SQLIA Detection : Static Approach : This approach is also known as pre-generating approach. Programmers follow some guidelines for SQLIA detection during web application development. An effective validity checking mechanism for the input variable data is also requires for the pre-generated method of detecting SQLIA.
Dynamic Approach : This approach is also known as post-generated approach. Post-generated technique are useful for analysis of dynamic or runtime SQL query, generated with user input data by a web application. Detection techniques under this post-generated category executes before posting a query to the database server [1,2].

© 2014, IJARCSSE All Rights Reserved

B.Classification of SQLIA

SQLIA can be classified into five categories:

1) Bypass Authentication - using tautology
2) Unauthorized Knowledge of Database - using illegal/incorrect queries
3) Unauthorized Remote Execution of Procedure
4) Injected Additional Query - Using Piggy-Backed Queries.
5) Injected Union Query

1) Bypass Authentication: Researchers have proved that query injection can't be applied without using space, single quotes or double dashes (--). In bypass authentication, intruder passes the query in such a way which is syntactically true and access the unauthorized data[6] . For example:

"SELECT salary FROM Employee WHERE ename=' or 1=1 - - 'and eid='';

This SQL statement will be passed because 1=1 is always true and - - which is used for comments, when used before any statement, the statement is ignored. So the result of this query allows intruder to access into Employee with its privileges in the database.

2) Unauthorized Knowledge of Database: In this type of attack, intruder injects a query which causes a syntax, or logical error in to the database. The result of incorrect query is shown in the form of error message generated by the database and in many database error messages, it contains some information regarding database and intruder can use these details[8]. This type of SQLIA is as follows:

SELECT salary FROM Employee WHERE ename=' sharan' and eid= convert (select host from host);

This query is logically and syntactically incorrect. The error message can display some information regarding database. Even some error messages display the table name also.

3) Unauthorized Remote Execution of Procedure: SQLIA of this type performs a task and executes the procedures for which they are not authorized. The intruder can access the system and perform remote execution of procedure by injecting queries. In below query, only SHUTDOWN operation is performed which shuts down the database .
For example:

SELECT salary FROM Employee WHERE ename=' sharan' ; SHUTDOWN; and eid = '';

4) Injected Additional Query: When an additional query is injected with main query and if main query generates Null value, even though the second query will take place and the additional query will harm the database. For example:

SELECT salary FROM Employee WHERE ename = 'Sharan' andeid = ' '; DROP table User;

First query generates Null because the space is not present between 'and' and eid, but the system executes the second query and if the given table present in database, it will be dropped.

5) Injected Union Query: In this type of attack, the intruder injects a query which contains set operators. In these queries, the main query generates Null value as a result but attached set operators data from database. For example:
SELECT salary FROM Employee WHERE ename='' and eid ='' UNION SELECT salary FROM Employee WHERE eid='100494';
In above query, the first part of query generated Null value but it allows the intruder to access the salary information of a user Having eid 100494. This paper emphasizes on various aspects of SQLIA. Section II describes the prevention techniques and operations in the previous work done in this field. Section III describes the proposed solution using AES encryption/Decryption and Tokenization. Section IV shows the Result and analysis and section V contains the conclusion and future research directions to prevent SQLIA.

II. Related work

In Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection, the authors proposed a solution to the problem of unauthorized access to the database by preventing it using an encryption algorithm based on randomization. This approach is based on SQLrand and randomization algorithm is used to convert input into a cipher text incorporating the concept of cryptographic salt. However, the main flaws in this approach are, Use of lookup table is not efficient way, cannot handle second order SQL injection attack and it also requires more space to store the look up table[3].

Gaurav Shrivastava and Kshitij Pathak proposed a model for SQL injection prevention using tokenization. In this paperl, they extract the where clause from the input query and put the remaining query (after where clause) in a temporary variable. They applied tokenization only on the remaining query which converts the tokens into hierarchical form such as left and right child. They performed validation of each token by comparing the value of left and right child to the root condition This model prevents all type of SQL injection attacks which are occurred only after the where clause [5].

Debabrata Kar and Suvasini Panigrahi proposed a lightweight approach to prevent SQL Injection attacks by a novel query transformation scheme and hashing. They used a novel query transformation scheme that transforms a query into its structural form instead of the parameterized form. In order to store the transformed queries, they proposed to apply a suitable hashing function to generate unique hash keys for each transformed query. This approach can also be easily implemented on any language or database platform with little modification. However, this approach cannot prevent second order SQL injection attempts since the parameter values (especially the string values) are removed during the

© 2014, IJARCSSE All Rights Reserved

transformation process. Another drawback is that for query transformation, they used query transformation scheme lookup table[4].

III. Proposed model

The proposed system focuses on how to detect and prevent SQL injection attacks on web applications using encryption and tokenization technique. The tokenization process is applied on the input query by detecting spaces, single quotes and double dashes etc. This process converts the input query into fruitful tokens and that are stored in a dynamic table at the client side.The table name, field name and data are encrypted using AES algorithm .The encrypted the original input query and the tokenized table are sending to the server side. At the server side , input query is decrypted and in turn converts into various token which are stored in to another dynamic table. Both dynamic tables are compared and if both are equal, it seems that there is no injection attacked in the given query ,hence the query is proceed further to main database for retrieving result .If they are different, query is rejected and not forwarded to the database server. The Customized error message notification is given to the client. Fig. 2 shows the proposed architecture the SQL Injection Attacks prevention.

A. AES Encyprion and Decryption

The attributes and data in the input query are encrypted using AES (Advanced Encryption Standard) algorithm which is fast, and requires little memory. Once the query is arrived at server side, which is decrypted by using the same key and in turn converts into various token which are stored in to another dynamic table. The performance comparison of cipher text over normal text shows that, cipher text is very difficult and time consuming to crack.

B. Query tokenization

Query tokenization technique converts the input query into various tokens. These tokens are generated by detecting single quote, double dashes and space in an input query. All string before a single quote, before double dashes and before a space constitutes a token. Tokenization process executes in following four essential steps and then forwarded to the server side.

Step 1 **:** Process the input query by replacing all the unnecessary characters which are used to make attacks on query.

Step 2 : Detect Single Quote, Double Quote, Double slashes and space in the input query.The fig.3. shows how tokens are formed by detecting spaces ,single quote and double dashes in input query for the below given input query.

" SELECT eid , ename FROM Employee WHERE salary > 2000 "

Step 3**:** Break the input query into various useful tokens .

Step 4: Store the tokens in a Dynamic table.

Step 5: Query Forwarding – After tokenization, the encrypted input query and dynamic token table are forwarded to server.

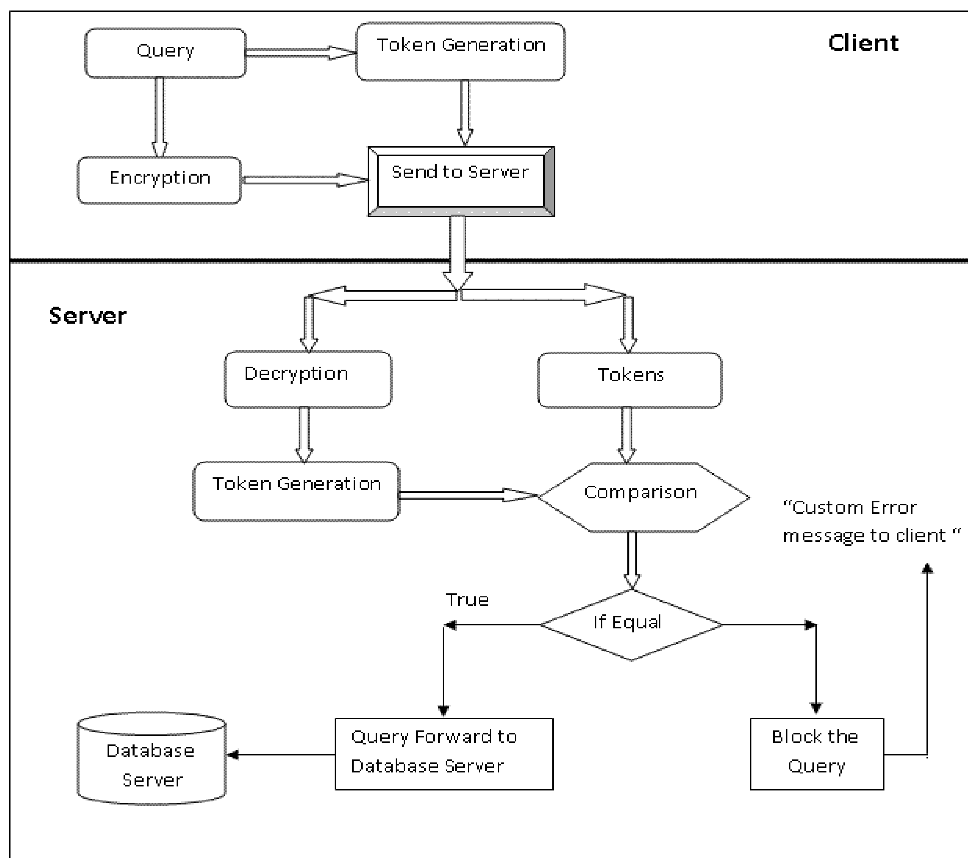

Fig. 2. Proposed Architecture

© 2014, IJARCSSE All Rights Reserved

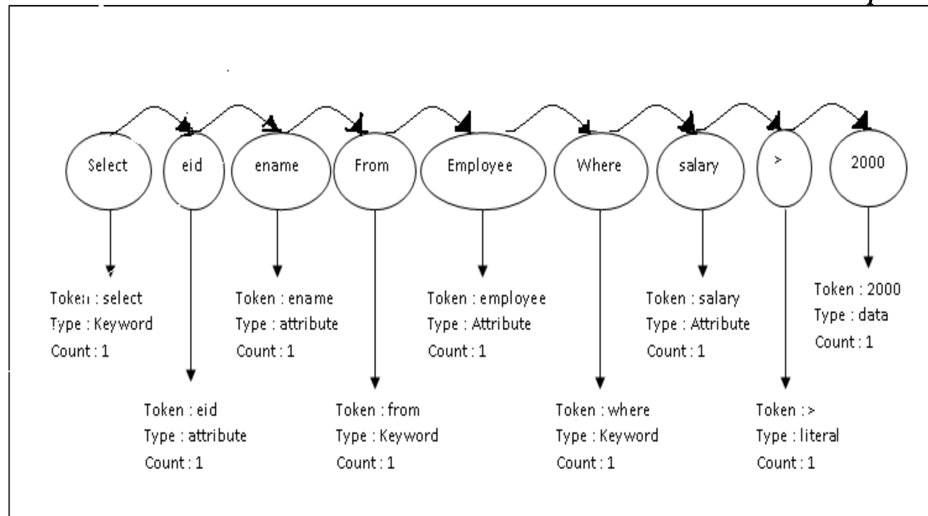

Fig 3. Token formation

C.Comparison of Dynamic tables

In this part, the length of both the dynamic tables such as D1 and D2 are compared. If lengths are same, then the each and every occurrences of both D1 and D2 are compared. After comparison, if both are same, there is no injection presented in the query and the query is steps forward further to main database for accessing the table. But if the length of D1 and D2 are different or else even any one token is different, then injection has attacked and query does not forwarded to the database. The attacked query has rejected and send customized error message back to the client.

IV. Result and analysis

The web application named Employee Information system is implemented in ASP.NET using C# and Microsoft SQL Server 2008 as backend database is chosen for experiment. This application is experimented with number of requests in three different iterations as shown in Table 1. The result of these iterations are shown as graphical representation in the fig 4.

Table 1. Results obtained by Experiment

TOTAL DATABASE REQUESTS SENT BY CLIENT	INJECTED QUERIES	VALID QUERIES
500	80	420
700	100	600
1000	175	825

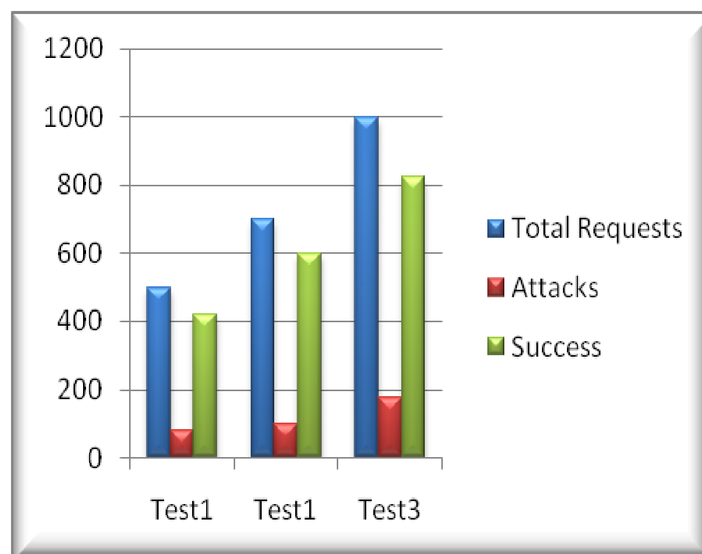

Fig 4. Graphical representation of results

Table 2 shows the comparison of proposed system with existing system on the basis of different SQL injection type like Bypass authentication, Unautherized Knowledge of Database, injected additional query and Second order SQL Injection.

Table 2. Comparison with Existing System

SQL INJECTION TYPES	SQLIA PREVENTION TECHNIQUE	
	EXISTING MODEL'S OUTCOME	PROPODES MODEL'S EXPECTED OUTCOMES
Bypass authentication	Prevented	Prevented
Unautherized Knowledge of Database	Prevented	Prevented
Injected additional query	Prevented	Prevented
Second order SQL Injection	Not Prevented	Prevented

V. Conclusion and future work

This paper has presented a lightweight method to prevent SQL injection attacks by applying query tokenization technique to convert SQL queries into number of useful tokens and then encrypting the table name, fields, literals and data on the query using AES Encryption algorithm .This approach avoids memory requirements to store the legitimate query in repository and facilitates fast and efficient accessing mechanism with database. Our experimental results show that this approach can effectively prevent all types of SQL injection attempts. This approach does not require major changes to application code and has negligible effect on performance even at higher load conditions due to its low processing overhead. It can also be easily applied to any other language & database platform without major changes. Further explore on the query transformation scheme is needed to make use of new encryption algorithm for preventing SQL injection attacks.

References

1. Asha. N, M. Varun Kumar,Vaidhyanathan.G of *Anomaly Based Character Distribution Models in the,"Preventing SQL Injection Attacks"*, International Journal of Computer Applications (0975 – 8887) Volume 52– No.13, August 2012 .
2. A S Yeole, B BMeshram, *"Analysis of Different Technique for Detection of SQL Injection",* International Conference and Workshop on Emerging Trends in Technology (ICWET 2011) – TCET, Mumbai, India, ICWET'11, February 25–26, 2011, Mumbai, Maharashtra, India. 2011 ACM.
3. Avireddy. S, Perumal.V, Gowraj.N,Kannan R.S, Thinakaran.P, Ganapthi .S, Gunasekaran J.R, Prabhu.S, Random4: *An Application Specific Randomized Encryption Algorithm to prevent SQL injection*, ieee transactions on communications, vol. 60, no. 5, may 2012.
4. Debabrata Kar, Suvasini Panigrahi, *Prevention of SQL Injection Attack Using Query Transformation and Hashing*, IEEE International Advance Computing Conference (IACC),2013.
5. Gaurav Shrivastava, Kshitij Pathak, *SQL Injection Prevention using Tokenization: Technique and Prevention Mechanism*, IJARCSSE, Volume 3, Issue 6, June 2013.
6. Ke Wei, M. Muthuprasanna, Suraj Kothari, *"Preventing SQL Injection attacks in Stored Procedures"*.Proceedings of the 2006 Australian Software engineering Conference (ASWEC'06).
7. Witt Yi Win, Hnin Hnin Hnin, *A Simple and Efficient Framework for Detection of SQL Injection Attack* , IJCCER, Volume 1, Issue 2, July 2013.
8. Kai-Xiang Zhang, Chia-Jun Lin, Shih-Jen Chen, Yanling Hwang, Hao-Lun Huang, and Fu-Hau Hsu, "TransSQL: *A Translation and Validation-based Solution for SQL-Injection Attacks*", First International Conference on Robot, Vision and Signal Processing, IEEE, 2011
9. Justin Clarke, *SQL Injection Attacks*, Syngress Defence, 2nd Edition, 2012

© 2014, IJARCSSE All Rights Reserved