ISSN No. 0976-5697

Volume 2, No. 5, Sept-Oct 2011

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

C-Queued Technique against SQL injection attack

Sumit Dhariwal*
M.Tech Scholar,
Department of Information Technology,
SATI Vidisha M.P, India
sumitdhariwal22@rediffmail.com

Romil Rawat
M.Tech scholar,
Department Of Information Technology
SATI Vidisha M.P
rawat.romil@gmail.com

Nikhil Patearia
M.Tech Scholar,
Department of Computer Science & Engineering,
SATI Vidisha M.P, India
nikhil_sati29@rediffmail.com

Abstract: Web application is the great need of modernization, with the increase of web application grow, attacks have been manufactured. Among all attacks, SQL Injection is a most disastrous threat which destroys and even gains the complete accessibility of backend applications. Queries which are made dynamically after the user supplied input is highly susceptible to Injection .By providing the Single quotes, double quotes, double dashes, semicolon, tautology and other vulnerabilities inputs he could misconfigure or modify the contents of the underlying database of a web application. We proposed a concept to detect SQL injection attacks by Parsing the SQL Query into tokens (chunks of SQL queries). When attacker is making SQL injection he will use attacking tricks in his input. Our method consists of parsing of original query and a query with injection separately, the tokens are formed they all make a Circular-Link-List for which every token is an element of the circular link list. Two circular-link-lists resulting from both original query and a query with injection are obtained and their node-to-node is compared to detect whether there is injection or not. By checking the list-node cycle address and node-to-node comparison, the result would be made that, there is injection or not.

Keywords: Database security, SQL injection, Authentication Introduction.

I. INTRODUCTION

Today's modem web era, expects the organization to concentrate more on web application security. This is the major challenge faced by all the organization to protect their precious data against malicious access or corruptions. Generally the program developers show keen interest in developing the application with usability rather than incorporating security policy rules. Input validation issue is a security issue if an attacker finds that an application makes unfounded assumptions about the type, length, format, or range of input data. The attacker can then supply a malicious input that compromises an application [1]. When a network and host level entry points are fully secured; the public interfaces exposed by an application become the only source of attack.

The cross site scripting attacks, SQL Injections attacks and Buffer Overflow are the major threat in the web application security through this input validation security issues [11][1][2] Especially SQL Injection attacks breach the database mechanism such as Integration, Authentication, Availability ' and authorization [8][4]. The root cause of such prevalent SQL injection vulnerabilities is that web applications and intrusion detection systems use only limited set of attack patterns for evaluation [9], [10], [11], [12]. Sophisticated attacks that employ evasion techniques can easily circumvent most of the detection mechanism employed today [8][5]. In addition, even if the injected code is intercepted before execution, administrators are often presented with information that does not identify clearly the association between the commands that were attempted, the assets that were at risk, the threats that were imposed, and the countermeasures he/she has at disposal.

SQL injection is a bypassing technique through the unauthenticated user, he gets the access to most secure and the key of profile security, and the database could be changed, updated, or modified by intruder.

Our concept focuses on the entry point where an intruder could bypass towards the database. Here the original query and the query which is generated after the user input is compared. Here tokens are created of both the queries original query and dynamic query. And after tokenization circular linked-lists are created of both the queries, here node to node comparison is made and list node –cycle address is compared if it is found same ,there is no SQL-Injection otherwise there is Injection that is vulnerability is present.

II. RELATED WORK

With the reference [3][1], a SQL Injection attack occurs when input from a user includes SQL keywords so that the dynamically-generated SQL query changes the intended function of the SQL query in the application.

A. *Positive Tainting and Syntax Aware:-*

In this approach valid input strings are initially provided to the [13] system for detection of SQLIA. At runtime, it categorizes input strings and propagates the untrusted or other-than-trusted markings based on the initialization. After that, a 'syntax aware evaluation' is performed for evaluating the propagated strings. Thus, based on the evaluation, if untrusted strings are found, such queries are restricted from passing into the database server for processing. During initialization of the trusted strings, it performs identification and marking based on inputs. The strings are categorized as: (i) hard coded strings, (ii) strings implicitly created by Java and (iii) strings originated from external sources. In case of syntax-aware evaluation [6][7], it performs syntax evaluation at the database interaction point. Syntax defines the trust policies which are the functions defined by the web programmer. Functions perform pattern matching and if the result of matching gives positive outcome, the tool allows the query to be executed on the database server. Following issues are there in this method - (i) Initialization of trusted strings are developers dependent and (ii) Persistent storage of trusted strings may cause second order attack [1].

B. *Context Sensitive String Evaluation (CSSE):*

The basic idea behind this approach is to find out the root cause [14] of SQLIA. The root cause is the origin of the data (information about the data, termed as metadata) i.e., user-provided or developer-provided. Thus, any data provided by the user is marked as untrusted and data provided by the applications are termed as trusted. The untrusted metadata are used for syntactic analysis based on 'Context Sensitive String Evaluation (CSSE)'. Injection vulnerabilities may also occur due to programming flaws during developments.

C. *Program Query Language (PQL):*

A PQL is developed especially for web application programmers to [15] retrieve attack related queries. It also incorporates a static technique which finds the solutions to such attack related queries. The static analyser finds all potential matches conservatively using context-sensitive as well as flow-insensitive analysis. This static result guides the runtime or dynamic analysis. A PQL is a pre-defined grammar based language. It has query variables (arguments), statements (primitive, compound), sub queries (recursive event sequences or recursive object relations) reacting to match (print or abort etc.). A static checker and optimizer, translates by PQL into queries. The translation of the PQL into 'datalog' (another more expressive language) provides sufficient support to programs to resolve the attack related queries.

D. *Parse tree Evaluation Based on Grammar:*

The basic idea of this method is to block those queries generated [16] from user input, which defy the syntactic structure of the query, as defined by the developer. SQL queries generated at runtime are parsed based on a pre-defined grammar. Runtime SQL generated is parsed based on the grammar. Special literals '(|' and '|)' are used to mark the beginning and end of each input string. Each such string within markers-pairs is matched with the augmented grammar constructed for the purpose. If the query parses successfully, it meets the syntactic constraints and is declared as legitimate. Otherwise, it is declared as illegitimate and is blocked. A major issue of this method is that an attacker may manipulate the input string by entering the marking symbol Q |)Q . Thus the syntactical confinement of the string surrounding with Q(|Q and Q|)Q may be affected.

E. *Static Analysis:*

It combined with automatic reasoning has been proposed in [7]. This technique verifies that the SQL queries generated in the application usually do not contain a tautology and effective only in such cases.

F. *Dynamic Analysis:*

A free tool called Paros [8] automatically scans for SQL injection vulnerabilities with pre-defined attack codes. The Paros checks the contents of HTTP response messages to determine whether an SQL injection attack was successful or not.

G. *Combined Static and Dynamic Analysis:*

In [9], authors check SQL queries at runtime to see if they conform to a model of expected SQL queries. This approach uses a secret key to discover user inputs in the SQL queries. Thus, the security of the approach relies on attackers not being able to discover the key.

III. PROPOSED TECHNIQUE

Our technique consists of implementation of a method that detects vulnerable input. The special character and symbols which is the key-of-success of SQL-Injection attack, attack is thoroughly observed and marked these are double dashes, single quote, double quote, semicolon, bracket, and space. Tokens are so created cantains the prior steps of syntax analysis, and semantic analysis. The tokens of original query and SQL-injected query are created. All the created tokens are grouped to form the Circular-linked-list of original query and Circular-linked-list of SQL-injected query with a proper indexing.

The node-to-node comparison of circular linked-list and list-node cycle address of both the queries are compared ,if they matches ,that 's means there is no injection ,Otherwise there is SQL-Injection

Original query= select * from Utable where user_id=u_input;

Circular Linked list of original Query

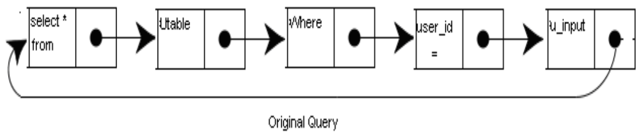

Figure 1: Original Query

Tokens(Node) of Original query

Node[0]= select* from .

Node[1]=Utable.

Node[2]=Where.

Node[3]=user_id.

Node[4]=u_input.
Sql-Injected query= select * from Utable where user_id=u_input OR 1=1;
Circular Linked list of Sql-Injected query

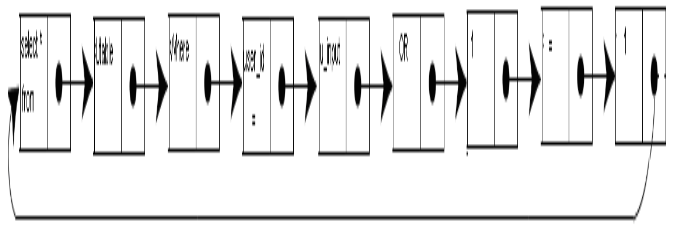

Figure 2: SQL-injected Query

Tokens(Node) of SQL-Injected query.
Node[0]= select* from .
Node[1]= Utable.
Node[2]= Where.
Node[3]= user_id.
Node[4]= u_input.
Node[5]= OR.
Node[6]= 1.
Node[7]= =.
Node[8]= 1.

The indexing of original query is from node[0] to node[4] and there is node-address cycle from

Node[0] $\Longrightarrow$ node[4] and

Node[4] $\Longrightarrow$ node [0]

And in the SQL-Injected query the indexing is from node[0] to node[8] and node address cycle from.

Node[0] $\Longrightarrow$ node[8] and

Node[8] $\Longrightarrow$ node[0]

Here if the link address cycle breaks and the node [index] is found to be mismatched the sql injection vulnerability is clearly predicted with the actual entry point of insertion and the fraud values with their signature it is the best way where the Injection is clearly found.

And by One-to-One node comparison of both queries it is clearly predicted that the node dta does not matched which 100% ensures the SQL-injection is there,

Here, we have used 2-Level security design.

a. For node to node comparison
b. List node cycle address

The both level ensures the successful prediction of SQL-Injection vulnerability.

IV. METHODOLOGY

The circular linked list of SQL query has been developed for evaluating the proposed technique .the system has been implemented on Microsoft sql-server as DBMS and java as front end language.here two stored procedures are created for comparing the tokens of original original query and user entered query,if both matches ,means no injection,otherwise there is sql-injection.every time user enters tokens are generated, when user logins into database.

A. *Testing:*

The proposed technique has been tested on a table having no. Of records.Dummy data table has been used,table consist of records of 100,200,300,400,500,600.

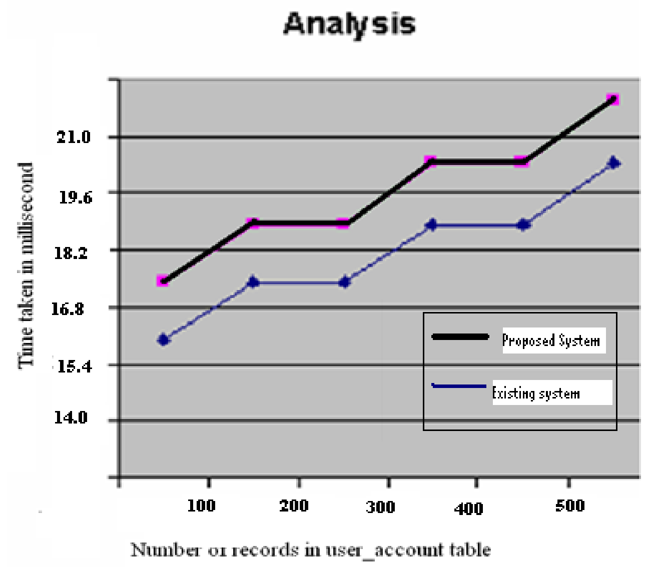

Figure 3: No. of records in user_ account table

Here, the graph looks linear that means there is very little difference of 1.4ms of extra time, it puts very little overhead on the existing system and seems negligible.

V. CONCLUSION

To apply SQL-Injection attack on web application, the attacker must have to use the defined special symbols .for analysing and removing SQL-Injection vulnerability we have a unique and most secure techniques. In this technique circular-linked-lists of original query and SQL-Injected query, which are constructed by the tokens of both query, followed by SQL grammar, javaCC, syntax analysis and semantic analysis, by comparing the node-to-node of circular linked-list and list node cycle address checking, SQL-Injection can be detected and eliminated. Here 2-level security is used for 100% ensuring the correctness and validness of SQL queries. It is also best way to find the Injection signature of the attacked SQL queries.

VI. REFERENCES

[1]. NTAGWABIRA Lambert and Kang Song Lin. Use of Query Tokenization to detect and prevent SQL Injection Attacks (2010).

[2]. R. Ezumalai and G. Aghila. Combinatorial Approach for Preventing SQL Injection Attacks. IACC, 2009.

[3]. MeiJunjin. An approach for SQL Injection vulnerability detection. IEEE, 2009.

[4]. Ke Wei, M. Muthuprasanna and Suraj Kothari.Preventing SQL Injection Attacks in Stored Procedures. IEEE, 2006.

[5]. Nuno Antunes and Marco Vieira. Detecting SQL Injection vulnerabilities in web services. IEEE, 2009.

[6]. G. Wassermann and Z. Su, "An Analysis Framework for Security in Web Applications," Proc. FSE Workshop on Specification and Verification of Component-Based Systems, pp. 70–78, 2008.
[7]. Paros, Parosproxy.org. http://www.parosproxy.org/.
[8]. Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," Proc. Annual Symposium on Principles of Programming Languages (POPL), pp. 372– 382, 2006.
[9]. A. S. Ofer Maor, "Sql injection signatures evasion," White Paper, Imperva Inc., 2005. [Online]. Available: http://www.imperva.com/application defense center/white papers/ sql injection signatures evasion.html
[10]. D. Litchfield, "Data-mining with sql injection and inference," Technique Report, an NGSSoftware Insight Security Research (NISR) Publication, 2005. [Online]. Available: http://www.ngssoftware.com/research/papers/sqlinference.pdf
[11]. S. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in American Conference on Neutron Scattering, College Park, Maryland, USA, 6-10 June 2004, pp. 202–302.
[12]. B. W. W. G. T. Buehrer and P. A. G. Sivilotti, "Using parse tree validation to prevent sql injection attacks," in International Workshop on Software Engineeringand Middleware, Lisbon, Portugal, September 2005.
[13]. William G.J. Halfond, Alessandro Orso and Panagiotis Manolios. Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks. SIGSOFT'06/FSE-14, November 5-11, 2006, Portland, Oregon, USA.
[14]. Tadeusz Pietraszek and Dhris Vanden Berghe. Defending against Injection Attacks through Context-Sensitive String Evaluation. Proceedings of Recent Advances in Intrusion Detection (RAID2005).
[15]. Finding Application Errors and Security Flaws Using PQL: a Program Query Language. OPSLA'05, October 16-20, 2005, San Diego, California, USA.
[16]. Z.Su and G. Wassermann. The Essence of Command Injection Attacks in Web Application. In the 33rd Annual Symposium on Principles of Programming languages, pages 372-382, Jan. 2009.

Short Biodata of Authors:

Nikhil Patearia presently pursuing M.Tech in the department of Computer Science & Engineering at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of B.E. secured in Computer Science & Engineering at Truba Institute of Engineering & Information Technology Bhopal, in 2009. Research Interest includes Network Security, Data Mining and Artificial Intelligence.
Mobile: +91-8989443679, E-mail: nikhil_sati29@rediffmail.com

Mr.Romil Rawat presently pursuing M.Tech in the department of Information technology & Science at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of Research Interest includes Network Security, Data Mining and Artificial Intelligence.
Mobile: +91-9907708093, E-mail: rawat.romil@gmail.com

Mr.Sumit Dhariwal presently pursuing M.Tech in the department of Information technology & Science at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of Research Interest includes Network Security, Data Mining and Artificial Intelligence.
Mobile: +91-9752070470, E-mail: sumitdhariwal22@rediffmail.com