Abstract:

SQL injection attacks, a class of injection flaw in which specially crafted input strings leads to illegal queries to databases, are one of the topmost threats to web applications. A number of research prototypes and commercial products that maintain the queries structure in web applications have been developed. But these techniques either fail to address the full scope of the problem or have limitations. Based on our observation that the injected string in a SQL injection attack is interpreted differently on different databases. A characteristic diagnostic feature of SQL injection attacks is that they change the intended structure of queries issued. Pattern matching is a technique that can be used to identify or detect any anomaly packet from a sequential action. Injection attack is a method that can inject any kind of malicious string or anomaly string on the original string. Most of the pattern based techniques are used static analysis and patterns are generated from the attacked statements. In this paper, we proposed a detection and prevention technique for preventing SQL Injection Attack (SQLIA) using Aho–Corasick pattern matching algorithm. In this paper, we proposed an overview of the architecture. In the initial stage evaluation, we consider some sample of standard attack patterns and it shows that the proposed algorithm is works well against the SQL Injection Attack.

Keywords—SQL Injection Attack; Pattern matching; Static Pattern; Dynamic Pattern
I. INTRODUCTION

Companies and organizations use web applications to provide better service to the end users. The Databases used in web applications often contain confidential and personal Information. SQL Injection attacks target databases that are accessible through a web front-end, and take advantage of flaws in the input validation logic of Web components such as CGI scripts. This communication is commonly done through a low-level API by dynamically constructing query strings with in a general purpose programming language. This low-level interaction (or) communication is dynamic (or) session based because it does not take into account the structure of the output language. The user input statements are treated as isolated lexical entries (or) string. Any attacker can embed a command in this string, which poses a serious threat to web application security. In the last few months application-level vulnerabilities have been exploited with serious consequences by the hackers have tricked e-commerce sites into shipping goods for no charge, usernames and passwords have been harvested and confidential information such as addresses and credit-card numbers has been leaked. The reason for this occurrence is that web applications and detection systems do not know the attacks thoroughly and use limited sets of attack patterns during evaluation. SQL Injection attacks can be easily prevented by applying more secure authentication schemes in login phase itself. Most of the attacks made on the web target the vulnerability of web applications. Vulnerabilities of web ability to obtain and charge the sensitive information, such as military systems, banks, and e-business, etc are exposed to a great security risk. SQL Injection Attack (SQLIA) is one of the very serious threats for web applications. Pattern matching is a technique that can be used to identify or detect any anomaly packet from a sequential action. Injection attack is a method that can inject any kind of malicious string or anomaly string on the original string.

For better understanding let us have look at the following example. We all know that most of the applications that we are accessing through internet will have a
login page to authenticate the user who is using the application. Figure 1 shows such a login page. Here when a user is submitting his username and password, an SQL query is generated in the back end to check whether the given credentials are valid or not. Suppose the given username is 1 and password is 111, the query will be:

\[
\text{Select * from login where user='1' and pass='111'}
\]

This is the normal case and if any rows are selected by the query, the user is allowed to log in. Now, we will see the attacker scenario. That is an attacker wants to log in without correct username and password. Instead of entering valid username if he uses injection string like “hacker’ OR ‘1’=‘1’—” as username and “something” as password, the query formed will be like this:

\[
\text{Select * from login where user='hacker' or '1'='1' -- and pass='something'}
\]

When this query is executed in the database, it will always return a true and the authentication will succeed.

II. RELATED WORK

Although web application can be classified as programs running on a web browser, web applications generally have a Tree-tier construction.

1) Presentation Tier: receives the user’s input data and shows the result of the processed data to the user. It can be thought of as the Graphic User Interface (GUI). Flash, HTML, Javascript, etc. are directly interact with the user.

2) CGI Tier: also known as the Server Script Process, is located in between the presentation tier and database tier. The data inputted by the user is processed and stored into the database. The database sends back the stored data to the CGI tier which is finally sent to the presentation tier for viewing. Therefore, the data processing within the web application is done at the CGI Tier. It can
be programmed in various server script languages such as JSP, PHP, ASP, etc.

3) Database Tier: stores and manages all of the processed user’s input data. All sensitive data of web applications are stored and managed within the database.

The database tier is responsible for the access of authentication users. A general framework for detecting malicious database transaction patterns using data mining was proposed by Bertino et al [1 2] to mine database logs to form user profiles that can model normal behaviors and identify anomalous transaction in database with role based access control mechanism. The system is able to identify intruder by detecting behaviors that different from the normal behavior. Kamra et al [3], proposed an enhanced model that can identify intruders in databases where there are no roles associated with each user. Bertino et al [4], proposed a framework based on anomaly detection technique and association rule mining to identify the query that deviates from the normal database application behavior. Bandhakavi et al [5] proposed a misuse detection technique to detect SQLIA by discovering the intent of a query dynamically and then comparing the structure of the identified query with normal queries based on the user input with the discovered intent.

The contribution of this paper is to propose a technique for detecting and preventing SQLIA using both static phase and dynamic phase. The proposed technique uses static Anomaly Detection using Aho–Corasick Pattern matching algorithm. The anomaly SQL Queries are detection in static phase. In the dynamic phase, if any of the query is identified as anomaly query then new pattern will be created from the SQL Query and it will be added to the Static Pattern List (SPL).

III. PROPOSED SCHEME

In this section, we introduce an efficient algorithm for detecting and preventing SQL Injection Attack using Aho–Corasick Pattern matching algorithm. The proposed architecture is given in figure 1 below. The proposed scheme has the following two modules, 1) Static Phase and 2) Dynamic Phase In the Static Pattern List, we maintain a list of known Anomaly Pattern. In Static Phase, the user generated SQL Queries are
checked by applying Static Pattern Matching Algorithm. In Dynamic Phase, if any form of new anomaly is occur then Alarm will indicate and new Anomaly Pattern will be generated. The new anomaly pattern will be updated to the Static Pattern List. The following steps are performed during Static and Dynamic Phase,

**Static Phase**

Step 1: User generated SQL Query is send to the proposed Static Pattern Matching Algorithm

Step 2: The Static Pattern Matching Algorithm is given in Pseudo Code is given below

Step 3: The Anomaly patterns are maintained in Static Pattern List, during the pattern matching process each pattern is compared with the stored Anomaly Pattern in the list

Step 4: If the pattern is exactly match with one of the stored pattern in the Anomaly Pattern List then the SQL Query is affected with SQL Injection Attack

**Dynamic Phase**

Step 1: Otherwise, Anomaly Score value is calculated for the user generated SQL Query. If the Anomaly Score value is more then the Threshold value, then a Alarm is given and Query will be pass to the Administrator.

Step 2: If the Administrator receives any Alarm then the Query will be analyze by manually. If the query is affected by any type of injection attack then a pattern will be generated and the pattern will be added to the Static Pattern list

**A. Anomaly Score value calculation**

In the static phase, each anomaly pattern from the Static Pattern List is checked with the user generated query. The proposed scheme will calculate the Anomaly Score value of the query for each pattern in the Static Pattern List. If the Query is match 100% with any of the pattern from the Static Pattern List, then Query is affected with SQL Injection Attack. Otherwise, the high matching score is called as an Anomaly Score value of a query. If the Anomaly Score value is more then the Threshold value (assume that 50%), then the query will be transfer to the Administrator.

**B. Pattern Matching Algorithm**

In the proposed architecture, Static Pattern Matching Algorithm is the main part and the pseudo code for the Algorithm is given below,
There are many approaches to recognizing patterns that involve using finite automata. The Aho–Corasick algorithm is one such classic algorithm. The idea is that a finite automaton is constructed using the set of keywords during the pre-computation phase of the algorithm and the matching involves the automaton scanning the SQL query statement reading every character in SQL query exactly once and taking constant time for each read of a character. Pseudo code of the Aho–Corasick multiple key word matching algorithm is given below.

The AC algorithm uses a refinement of a tries to store the set of Anomaly Keywords in a pattern matching example. The sample of Query generation is given in figure 2 and the process of pattern matching for the user generated query in given in Figure 3 and Figure 4.
Fig. 1. Proposed Architecture

```
SELECT * FROM user_account WHERE login = 'John' AND pass = 'xyz'
```

Fig. 2. SQL Query Generation with legal user name and password

```
SELECT * FROM user_account WHERE login = 'John' AND pass = 'xyz'
```

Fig. 3. SQLIA Pattern Matching Process

```
SELECT * FROM user_account WHERE login = 'John' AND pass = 'xyz'
```

Fig. 4. SQLIA Pattern Exactly Matching
IV. CONCLUSION

This paper describes the challenges that Internet applications that make use of a database system face in terms of security and protection the private data. SQL-Injection attacks is a legitimate threat that endangers the confidentiality of data and may cost an organization a great deal of money and even their reputation for different tools. The system is significantly safer from both online threats. there will be constant need to protect it. So we have proposed a scheme for detection and prevention of SQL Injection Attack using Aho–Corasick pattern matching algorithm. The proposed scheme is evaluated by using sample of well known attack patterns. Initial stage evaluation shows that the proposed scheme is produce not false positive and false negative. The pattern matching process takes minimum of O (n) time

REFERENCES

[1] Amit Kumar Pandey, “SECURING WEB APPLICATIONS FROM APPLICATION-LEVEL ATTACK”, master thesis, 2007
[2] C.J. Ezeife, J. Dong. A.K. Aggarwal, “SensorWebIDS: A Web Mining Intrusion Detection System”, International Journal of Web Information Systems, volume 4, pp. 97-120, 2007
[3] S. Axelsson, "Intrusion detection systems: A survey and taxonomy", Technical Report, Chalmers Univ., 2000
[4] Marhusin, M.F.; Cornforth, D.; Larkin, H., “An overview of recent advances in intrusion detection”, in proceeding of IEEE 8th International conference on computer and information technology CIT, 2008
[5] S. F. Yusufovna., “Integrating Intrusion Detection System and Data Mining”, International Symposium on Ubiquitous Multimedia Computing, 2008
[6] Low, W. L., Lee, S. Y., Teoh, P., “DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions”, in Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS), 2002
[7] F. Valeur, D. Mutz, and G. Vigna, “A learning-based approach to the detection of sql injection attacks”, in proceedings of the conference on detection of intrusions and Malware and vulnerability assessment (DIMVA), 2005
[8] Bertino, E., Kamra, A, Terzi, E., and Vakali, A, “Intrusion detection in RBAC-administered databases”, in the Proceedings of the 21st Annual Computer Security Applications Conference, 2005
[9] Kamra A, Bertino, E., and Lebanon, G., “Mechanisms for Database Intrusion Detection and Response”, in the Proceedings of the 2nd SIGMOD PhD Workshop on Innovative Database Research, 2008
[10] Kamra A, Terzi E., and Bertino, E., “Detecting anomalous access patterns in relational databases”, the VLDB Journal VoU7, No. 5, pp. 1063-1077, 2009
[11] Bertino, E., Kamra, A, and Early, J., “Profiling Database Application to Detect SQL Injection Attacks”, In the Proceedings of 2007 IEEE International Performance, Computing, and Communications Conference, 2007
[12] Bandhakavi, S., Bisht, P., Madhusudan, P., and Venkatakrishnan V., “CANDID: Preventing sql injection attacks using dynamic candidate evaluations”, in the Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007
[13] Halfond, W. G. and Orso, A., “AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks”, in Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005
[14] William G.J. Halfond, Alessandro Orso, and anagiotis Manolios, “WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transactions on Software Engineering, Vol. 34, No. 1, pp 65-81, 2008
[15] Buehrer, G., Weide, B. w., and Sivilotti, P. A, “Using Parse Tree Validation to Prevent SQL Injection Attacks”, in Proceedings of the 5th international Workshop on Software Engineering and Middleware, 2005
[16] Prabakar, M.A.; Karthikeyan, M.; Marimuthu, K., “An efficient technique for preventing SQL injection attack using pattern matching algorithm,” Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on , vol., no., pp.503,506, 25-26 March 2013 doi: 10.1109/ICE-CCN.2013.6528551