ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 3, Issue 1 (Jan. - Mar. 2015)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2015)

Positive Tainting and Syntax-Aware Evaluation as Effective Techniques to Prevent SQL Injection Attacks

[I]Seema Bhardwaj, [II]Abhilasha Kumari, [III]Pooja Jambhale, [IV]Pragati Kakade, [V]Avinash Kharade
[I,II,III,IV,V]Dept.of IT ISB&M School of Technology NANDE, Pune University, Maharashtra, India

Abstract
The tremendous growth and use of wide-range of Web applications today is the very reason that they are a potential target to the different forms of attacks. Attacking the databases through these Web applications is one of the major security threats. SQL injection attack (SQLIA) is used by various hackers to get access to the data present in the database by taking advantage of the loopholes which are present in the server side programs. It is a flaw in Web application and not a database or Web server problem. It is a method used to pass an SQL code through some user interactive applications over the Web. It is a serious threat to security as it is easy to generate and tough to design the most efficient method which counters SQL injection attack. In this paper, we propose a new and highly automated approach for protecting the existing Web applications against various SQL injection attacks. This approach has conceptual and practical advantages over the existing techniques. Conceptually, the idea is based on positive tainting and syntax-aware evaluation and practically this technique is efficient and easy to implement with minimal requirements. Further, we have also given an overview of some of the SQLIA. This paper also mentions the pattern matching algorithm called Aho-Corasick multiple keyword algorithm which will help to keep in check the various SQLIA already used by the hackers.

Keywords
SQL Injection, SQL Query, Positive tainting, Syntax-Aware Evaluation

I. Introduction

SQL Injection is one of the many web attack mechanisms used by hackers to steal data from organizations. It is perhaps one of the most common application for attacking. *SQL Injection* is a type of web application security vulnerability in which an attacker is able to insert a malicious SQL statement into an entry field for execution, exposing the back-end database.

Even though this vulnerability has had its presence for several years now, most of its popular techniques are based on safe coding practices, which are not applicable to the existing applications. Web applications basically interact with the databases, retrieve the data from it and then presents it to the user. To prevent as well as detect database from various SQL injection attacks two methods have been proposed which are static and dynamic pattern matching. Pattern matching checks a given sequences of tokens for the presence of the constituents of some pattern. Although, there is no single fixed way of attacking the database using SQL but there are various techniques used to intrude the system. Some of the already prevalent attacks are Tautology attack, Piggybacking Attack, Union Attack. Also there are other attacks like Blind Injection, Stored procedure, Timing Attacks which are also capable of harming the confidential data in a powerful way.

SQLIA attacks can be prevented using the following algorithms. Static Pattern Matching Algorithm and Aho-Corasick Multiple String Pattern Matching Algorithm are the popular and efficient ones.

II. Related Work

Several researches and different methods and approaches have been used and implemented in the last two decades for prevention of SQL Injection. SQLIA are considered to be of the topmost priority when it comes to solving problems related to Web security. When an attack on the database takes place it may lead to loss of confidential data. Not only loss but it may also malign the data in many ways. Survey of Web applications like e-commerce, net banking, online shopping and supply chain management sites, concludes that at least 92 percent of Web applications are vulnerable to some form of attack.

It is evident that confidential data have always been the target of hackers and hence different methods are applied by them to get access to such data and harm the system. Unfortunately there is no proper guarantee for preserving the underlying databases from current attacks. There are various detection and prevention techniques which can be divided into two categories.

First approach is to try detecting SQLIA by checking anomalous SQL Query structure using string and pattern matching methods and query processing. The second approach uses the dependency among data items which is less likely to change so that malicious database activities are identified. In both the approaches, many of the researchers proposed different schemes by integrating data mining and intrusion detection systems. This minimizes the false positive alerts, reducing human intervention and better detection of attacks. Moreover, different intrusion detection techniques are used either separately or otherwise.

Bertino et al proposed a framework based on anomaly detection technique and association rule mining to identify those queries that deviate from the normal database application behavior. Association rule mining technique is employed for mining frequent parameter list and the order to identify intrusions. Bandhakavi et al proposed a misuse detection technique to detect SQLIA by finding the intent of a query dynamically and then comparing the structure of that query with the normal ones based on the user input. Halfond et al developed a technique which makes use of a model-based approach to detect malicious and illegal queries before they are executed on the database.

William et al proposed a system Web Application SQL Injection Preventer (WASP) to prevent SQL Injection Attacks by positive tainting. The basic approach consisted of identification of trusted data sources and marking data coming from these sources as trusted using positive tainting to track trusted data at runtime, and allowing only trusted data to become SQL keywords or operators in query strings. William et al also mentioned that syntax-aware evaluation is a method which considers the context in which the trusted and untrusted data is used.

Kamra et al proposed an enhanced model that can identify

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

intruders in databases where each user is not associated with some role. Halfond developed a technique that uses a model–based approach to detect illegal queries before they are executed on the database.

The Secure Web Applications Project (SWAP) is an interdisciplinary research initiative between the Laboratory for Communications Engineering and the Computer Laboratory of the University of Cambridge that intends to solve the problem of application-level Web security at a higher level. This project reduces application development time as well as protects against a large class of application-level attacks more effectively than the existing Web development methodologies.

Yusufovna proposed the application of data mining approaches for an intrusion detection system . M. Karthikeyan et al have proposed the use of patthern matching algorithm specifically, Aho-Corasick to match the patterns of incoming user queries which may be legal or illegal with the maintained list of patterns. Pekka Kilpelainen described the use and importance of Aho Corasick for set matching and string matching. In this paper, an IDS model is presented which is based on the techniques of positive tainting and syntax-aware evaluation for prevention of SQLIA which would also employ Aho-Corasick and Static Pattern matching algorithms.

III. Existing System

The existing system has the following two modules, first is Static Phase and second Dynamic Phase In the Static Pattern List, a list of known Anomaly Patterns is maintained. Each anomaly pattern from the Static Pattern List is checked with the user generated query. The Anomaly Score value of the query for each pattern in the Static Pattern List is calculated. If the Query matches 100% with any of the pattern from the Static Pattern List, then the Query is infected with attack. Otherwise, if the matching score is high it is called as an Anomaly Score value of a query. If the Anomaly Score value comes out to be more than the threshold value (assume 40%), then the query will be given to the Administrator for checking.

Static Phase

Step 1. The user generated SQL Query is sent to the Static Pattern Matching Algorithm.
Step 2. The Static Pattern Matching is then performed.
Step 3. The Anomaly patterns are maintained in Static Pattern List. During the pattern matching process, each pattern is compared with the stored Anomaly Patterns in the list.
Step 4. If the pattern matches with any one of the stored pattern in the list then the SQL Query is infected and it is an attack query.

Dynamic Phase

Step 1. If the pattern does not match, Anomaly Score value is calculated for the user generated SQL Query. If the Anomaly Score value is more than the threshold value, an alarm is generated and that query is passed to the Administrator.
Step 2. Step 3. If the Administrator receives any Alarm then the Query will be analyzed.

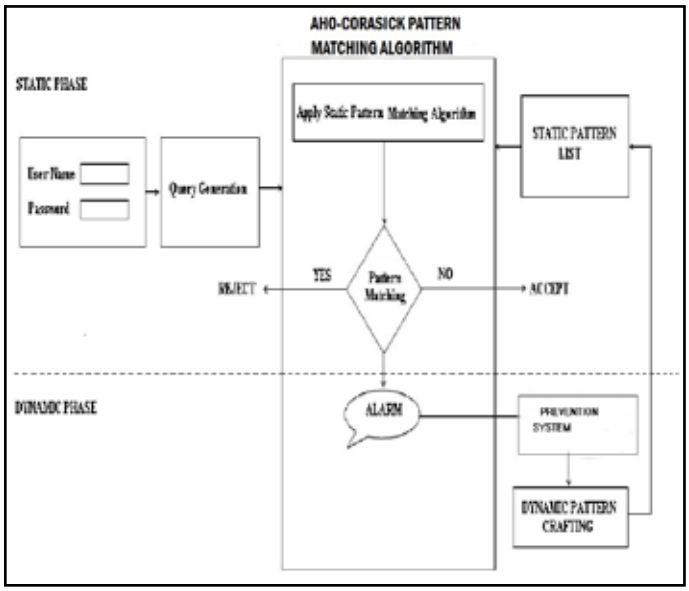

Fig.1: System Architecture

In the existing architecture, Static Pattern Matching Algorithm is the main part. Static Pattern Matching along with Aho-Corasick is used for reading every character in the SQL Query and for matching it.

IV. Proposed System

The SQLIA is a kind of attack which penetrates the user queries that have been forwarded towards the servers and databases. Such attacks are basically based on the belief of the people that the queries cant be compromised. Hence, in this paper, we have proposed a scheme for detecting as well as preventing the SQLIA. In this technique, we have used the Aho-Corasick multiple string pattern matching algorithm. The important feature of this technique is that, it does not generate false positive results. It first creates deterministic finite automata for all the predefined patterns and then by using automaton, it processes a text in a single pass. It consists of constructing a finite state pattern matching automata from the existing patterns and then uses the pattern matching automata to process the text string in a single pass.

A. Positive Tainting

Positive Tainting is based on the identification of the trusted data rather than untrusted data. Traditional Tainting is called negative tainting.

Positive tainting differs from negative tainting because it is based on the identification, marking and tracking of trusted, rather than untrusted, data. Positive tainting follows the general principle of fail-safe.

Negative tainting follows the identification of data which is not trusted and this is where positive tainting differs from negative tainting. This conceptual difference has significant effects. It helps address problems caused by the incompleteness in the identification of relevant data to be marked. Incompleteness can leave the Web Applications vulnerable to SQL injection attacks. With negative tainting, detection of attacks becomes very difficult. Thus, we have proposed the use of positive tainting in our approach. Identifying trusted data in Web Applications is often straight forward and always less prone to error.

Taint propagation is done during runtime. When the data is used and manipulated by users at runtime the taint markings associated with

data are identified,. Taint Propagation needs to be done accurately otherwise it would cause misuse of data. Our approach consists of: 1) Identifying taint markings 2) The effect of functions that operate on the tainted data precisely. The data is made up of characters. Hence to achieve accuracy, tainting at character level is carried in our approach. Here Strings are constantly broken into substrings for building SQL queries. Tokenization of the whole SQL Query is done i.e., the SQL query is broken down into tokens.

The way in which Web applications create SQL commands makes the identification of untrusted data problematic and the identification of all trusted data relatively straightforward.

Therefore, there are often many potential external untrusted sources of input to be considered for these applications, and enumerating all of them is considerably difficult and prone to error. For example, developers initially had assumed that only direct user input needed to be marked as tainted. Subsequent exploits demonstrated that attackers soon realized the possibility of exploiting local server variables and the database itself as the sources of injection. In general, it is difficult to guarantee that all potentially harmful data sources have been considered, and even a single unidentified source could leave the application vulnerable to attacks.

The situation is different for positive tainting because identifying trusted data in a Web application is often straightforward and always less error prone. To account for such cases, our technique provides developers with a mechanism to specify additional sources of external data that should be trusted. The data sources can be of various types, such as files, network connections and server variables. Our approach uses this information to mark data coming from these additional sources as trusted.

B. Syntax-Aware Evaluation

Positive tainting helps to create taint markings during execution but for achieving more security we must be able to use the taint markings to distinguish legitimate from malicious queries. The key feature of Syntax aware evaluation is that it considers the context in which trusted and untrusted data is present sothat it is made sure that all parts of query other than string or numerical or literals consists only of trusted. Conversely, if this property is not satisfied (e.g., if an SQL operator contains characters not marked as trusted), it can be that the operator has been injected by an attacker and block the query. Our technique performs syntax-aware evaluation of a query string immediately before the string is sent to the database to be executed.

Our approach must be able to use the taint markings to distinguish legitimate from malicious queries besides ensuring that taint markings are correctly created and maintained while executing. An approach that simply restricts the use of untrusted data in SQL commands is not a viable solution because it would mark any query that contains user input as an SQLIA, leading to many false positives.

To address this problem, a method is used which permits the use of tainted input as long as it has been processed by a sanitizing function. A sanitizing function is a filter that performs operations such as regular expression matching or replacement of sub-strings. The idea

of declassification is based on the assumption that sanitizing functions are able to eliminate or neutralize harmful parts of the input and make the data safe. However, in practice, there is no guarantee that the checks performed by a sanitizing function are adequate. Tainting approaches based on declassification could therefore generate false negatives if they mark as trusted supposedly-sanitized data that is in fact still harmful. Moreover, these approaches may

also generate false positives in cases where unsanitized, but perfectly legal input is used within a query.

Syntax-aware evaluation does not depend on any (potentially unsafe) assumptions about the effectiveness of sanitizing functions used by developers. It also allows for the use of untrusted input data in an SQL query as long as the use of such data does not cause an SQLIA. To evaluate the query string, the technique first uses an SQL parser to break the string into a sequence of tokens that correspond to SQL keywords, operators, and literals. The technique then iterates through the tokens and checks whether tokens (i.e., substrings) other than literals contain only trusted data. If all of the tokens pass this check, the query is considered safe and allowed to execute.

This approach can also handle those cases where developers use external query fragments to build SQL commands. In those cases, developers would specify which external data sources must be trusted, and our technique would mark and treat data coming from these sources accordingly.

This default approach, which considers only two kinds of data-trusted and untrusted, allows only trusted data to form SQL keywords and operators, is adequate for most Web applications. For example, it can handle applications where parts of a query are stored in external files or database records that were created by the developers. Our technique

also allows developers to relate custom trust markings to different data sources and provides custom trust policies which specify the legal ways to use data with certain trust markings. Trust policies are functions that take a sequence of SQL tokens as input and perform some type of check based on the trust markings related to the tokens.

Though, this technique tackles down the threat of SQLIA, the complete solution against SQLIA cannot be assured, as the attacks are always improvised with some new techniques. Thus, the future work is always welcome.

V. Conclusion

In this paper, we have proposed the efficient techniques for preventing and detecting different SQL injection attacks such as positive tainting and syntax aware evalution using Aho-Corasick Pattern matching calculations. Our approach makes it easy to remove all the false positives as well as makes trusted data readily identifiable in web application. It only allows marked strings to form a query using keywords and operations. Before the query is sent to the database our approach is being performed. It also used to increase the automation and implement security principles. Our system does not allow the use of untrusted data in queries. This approach also provides practical advantages over a lot of existing techniques whose applications require customized and complex runtime environments.

References

[1]. *Dr.M.AmuthaPrabakar,.KarthiKeyan, Prof.K.Marimuthu An Efficient Technique For Preventing SQL Injection Attack Using Pattern Matching Algorithm",2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN 2013).*

[2]. *Shelly Rohilla, Pradeep Kumar Mittal "Database Security by Preventing SQL Injection Attacks in Stored Procedures",International Journal of Advanced Research*

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

in Computer Science and Software Engineering,November 2013.

[3]. Khushbu Chauhan, Sumedha Dethe, Khushbu Shah and Aastha Goyal,"Hacking Prevention Using Positive Tainting", July 2012.

[4]. Kuldeep Kumar, Dr. Debasish Jena and Ravi Kumar, "A Novel Approach to detect SQL injection in Web applications", June 2013.

[5]. Arudchutha, T. Nishanthy and R.G. Ragel, "String Matching with Multicore CPUs: Performing Better with the Aho-Corasick Algorithm".

[6]. William G.J. Halfond, Alessandro Orso, and Panagiotis Manolios, "Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks".

[7]. Saima Hasib, Mahak Motwani, Amit Saxena, "Importance of Aho-Corasick String Matching Algorithm in Real World Applications"

[8]. Munqath H. Alattar S.P. Medhane," E_cient Solution for SQL Injection Attack Detection and Prevention", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-1, March 2013.

[9]. Sruthy Manmadhan1 and Manesh T, "A Method Of Detecting SQL Injection Attack To Secure Web Applications",International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.6, November 2012.

[10]. Dr. Rajashree Shettar, Agneev Ghosh, Amal Mohan, Amith Pramod, Chetan Raikar, "SQL Injection Attacks and Defensive Techniques".

[11]. William G.J. Halfond, Alessandro Orso, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transactions On Software Engineering", VOL. 34, NO. 1, Jan/Feb 2008.

[12]. Monali R, Borade and Neeta A. Deshpande, "Extensive Review Of SQLIA's Detection And Prevention Techniques", Vol.3, Issue10, October 2013.

[13]. William G.J. Halfond, Jeremy Viegas, and Alessandro Orso "A Classification of SQL Injection Attacks and Countermeasures" College of Computing, Georgia Institute of Technology, 2006.