International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Survey on Detection and Prevention of SQL Injection Attack

S. M. Chaware[1], Sujata S. Wakchaure[2]

[1, 2]Pune University, Maharashtra, India

Abstract: ***Many software systems include a web-based component that makes them available to the public via the internet and can expose them to a variety of web-based attacks. One of these attacks is SQL injection which can give attackers unauthorised access to the databases. This paper presents an approach for protecting web applications against SQL injection. Pattern matching is a system that can be utilized to distinguish or recognize any abnormality parcel from a consecutive activity. This paper also presents a recognition and avoidance strategy for protecting SQL Injection Attack (SQLIA) utilizing Aho-Corasick pattern matching calculation Furthermore, it focuses on different mechanisms that can detect several SQL Injection attacks.***

Keywords: SQL Injection attack, Pattern matching, Static pattern, Dynamic Pattern, Anomaly Score

1. Introduction

SQL injection vulnerabilities have been depicted as a standout amongst the most genuine threats for Web applications [4][1]. Web applications that are powerless against SQL injection may permit an attacker to addition complete access to their fundamental databases. Since these databases frequently contain sensitive consumers or client data, the ensuing security infringement can incorporate wholesale fraud, loss of secret data, and misrepresentation. At times, attackers can even utilize a SQL injection defenselessness to take control of and degenerate the system that has the Web application.

Web applications that are defenseless against SQL Injection Attacks (SQLIAs) are across the board. Truth be told, SQLIAs have effectively focused on prominent exploited people, for example, Travelocity, Ftd.com, and Surmise Inc. SQL injection alludes to a class of code-injection attacks in which information gave by the client is incorporated in a SQL query in such a path, to the point that piece of the client's input is dealt with as SQL code. By leveraging these vulnerabilities, an attacker can submit SQL summons straightforwardly to the database. These attacks are a genuine risk to any Web application that gets input from clients and consolidates it into SQL questions to a fundamental database. Most Web applications utilized on the Web or inside big business systems work thusly and could in this manner are helpless against SQL injection.

A standout amongst the most productive instruments to shield against web attacks utilizes Interruption Discovery System (IDS) and Network Intrusion Detection System (NIDS). An IDS utilizes abuse or abnormality location to protect against attack [3]. IDS that utilization oddity recognition system makes a gauge of typical use patterns. Abuse identification strategy utilizes particularly known patterns of unapproved conduct to foresee and locate resulting comparable sort of attacks. These sorts of patterns are called as signature [8][3]. NIDS are not help for the administration situated applications (web attack), in light of the fact that NIDS are working lower level layers [4].

2. Related Work

Beuhrer et. al. [6] has described a technique to prevent and to eliminate SQL injection attacks. The technique ibased on comparing, the parse tree of the SQL statement before inclusion of user input with the one that resulting after inclusion of input, at run time. This system implementation is intended to minimize the efforts the programmer needs to take; because, it automatically captures, both the actual query and the intended query and that too, with minimal changes necessarily to be done by the programmer.

Saltzer and Schroeder [7] propose a security system against the attacks similar to SQL Injection. They proposed a system using various stages. One of them was the fail-safe defaults, on which the positive tainting is dependent or follows, expresses that a conservative configuration must be focused around contentions why objects should to be open, as opposed to why they should not. In an expansive framework a few objects will be insufficiently considered, so a default of absence of permission is more secure.

An outline or usage botch in a component that gives unequivocal permission has a tendency to fizzle by declining permission, a safe circumstance, since it will be immediately recognized. Then again, a configuration or usage botch in a system that expressly rejects get to has a tendency to fizzle by permitting get to, a disappointment which may go unnoticed in ordinary utilization. This guideline applies both to the outward appearance of the assurance system and to its hidden execution.

Yusufovna [10] has presented an application of data mining approaches for IDS. Intrusion detection can termed as of detecting actions that attempt to threat the privacy, reliability and accessibility of the resources of a system. IDS model is presented as well as its limitation in determining security violations are presented in this paper.

Halfond and Orso [11] had presented a technology for detection and prevention of SQLIA. This technique made was based on the approach that intended to detect the malicious queries before their execution inside the database. To automatically build a model of the legal or correct

queries, the static part of the technique used the program analysis. This could be generated by the application itself. The technique used the runtime monitoring for inspection of dynamically generated queries and to check them against the static build model.

Halfond and Orso [12] had proposed a technique for countering SQL injection. The technique actually combined the conservative static analysis and runtime monitoring for detection and stoppage of illegal queries before they are executed on the database. The technique builds a conservative model of the legitimate queries that could be generated by the application in its static parts. The technique inspected the dynamically generated queries for compliance with statically build model in its dynamic part.

W. G. J. Halfond et. al. [13], proposed another, much automated methodology for ensuring existing Web applications against SQL infusion. This methodology has both calculated and commonsense favorable circumstances over most existing systems. From the calculated viewpoint, the methodology is focused around the original thought of positive spoiling and the idea of syntax-aware evaluation. From the reasonable outlook, the method is in the meantime exact and productive and has negligible arrangement necessities.

2.1 Types of SQLIA

2.1.1 Tautologies
Tautology-based attacks are among the simplest and best known types of SQLIAs. The general goal of a tautology based attack is to inject SQL tokens that cause the queries conditional statement to always evaluate to true[2]. This technique injects statements that are always true so that the queries always return results upon evaluation of WHERE condition [15].
Injected query: select name from user_details where username = ‘abc’ and password = or1 = 1.

2.1.2 Union Queries
SQL allows two queries to be joined and returned as one result set. For example, SELECT col1,col2,col3 FROM table1 UNION SELECT col4,col5,col6 FROM table2 will return one result set consisting of the results of both queries Using this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Injected query is concatenated with the original SQL query using the keyword UNION in order to get information related to other tables from the application [2].

Original query: select acc-number from user_details where u_id = 500
Injected query: select acc-number from user_details where u_id = ‘500’ union select pin from acc_details where u_id=’500’ [15]

2.1.3 Piggybacked
In this attack type, an attacker tries to inject additional queries along with the original query, which are said to ”piggy-back” onto the original query. As a result, the database receives multiple SQL queries for execution additional query is added to the original query. This can be done by using a query delimiter such as ”;”, which deletes the table specified [15]. Injected Query: select name from user_details where username = ‘abc’; droptable acc –

2.1.4 Timing attack
In this type of attack, the attacker guesses the information character by character, depending on the output form of true/false. In time based attacks, attacker introduces a delay by injecting an additional SLEEP(n) call into the query and then observing if the webpage was actually by n seconds [15].

2.1.5 Blind SQL injection attacks
Attackers typically test for SQL injection vulnerabilities by sending the input that would cause the server to generate an invalid SQL query. If the server then returns an error message to the client, the attacker will attempt to reverse-engineer portions of the original SQL query using information gained from these error messages [15].

2.2 Architecture for detection of SQL Injection Attack (SQLIA)

Amutha Prabakar and KarthiKeyan [1] give an algorithm for identifying and counteracting SQL Injection Attack utilizing Aho–Corasick Pattern matching algorithm. The existing plan has the accompanying two modules: 1) Static Stage, and 2) Dynamic Stage. The Static Pattern list keeps up a list of known Patterns of anomaly. In Static Stage, the client produced SQL Queries are checked by applying Static Pattern Matching Algorithm. In Dynamic Stage, if any type of new irregularity is happen then Alert will show and new Abnormality Pattern will be created. The new anomaly pattern will be redesigned to the Static Pattern List. The accompanying steps are performed amid Static and Dynamic stage; the changes are made in the Dynamic phase.

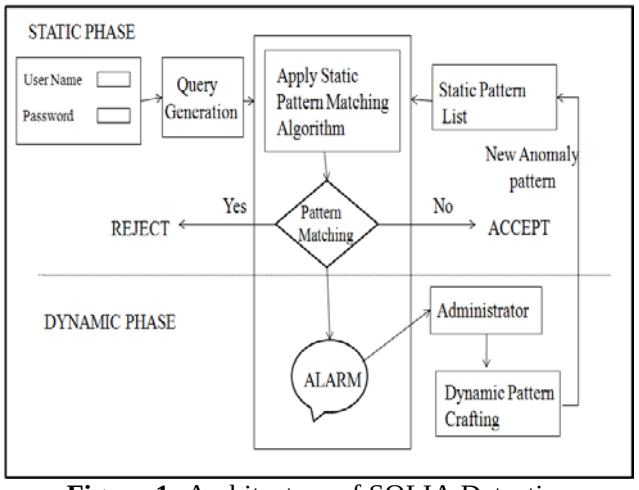

Figure 1: Architecture of SQLIA Detection

2.2.1. Static Phase
In this, a static pattern list is maintained. And we keep up a list of known anomaly patterns. The client generated SQL queries are checked by applying the Static Pattern Matching Algorithm.
Step 1. User’s Query is acquired and sent to the Static Pattern Matching Algorithm.

Step 2. Each Pattern is compared with the Anomaly Patterns, stored in the list, during the pattern matching.
Step 3. 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.

2.2.2. Dynamic Phase

In Dynamic Stage, if any type of new anomaly is occurred, then the alert is shown, and new anomaly will be created. This new anomaly pattern will be inserted in the Static Pattern List.
Step 1. Anomaly Score value is calculated for the user generated SQL Query,
Step 2. If the Anomaly Score value is more than the Threshold value, then an Alarm is given and Query will be passed to the Administrator.
Step 3. 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.

2.2.3. Anomaly Score value

In the static phase, each anomaly pattern from the static pattern List is compared with the user's query. The Anomaly Score value is generated for each query pattern static pattern list. If the query is match 100% with any of the pattern from the static pattern list, then that query is affected with SQL Injection Attack(SQLIA). 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.

3. Conclusion

In this paper, we presented a novel technique against SQLIAs, we surveyed a plan for recognition and counteractive action of SQL Injection Attack(SQLIA) utilizing Aho–Corasick pattern matching calculation. The surveyed plan is assessed by utilizing specimen of well known attack patterns. The technique is fully automated and detects SQLIAs using a model-based approach that combines static and dynamic analysis. This application can be used with various databases.

Reference

[1] M. A. Prabakar, M. KarthiKeyan, K. Marimuthu, "An Efficient Technique for Preventing SQL Injection Attack Using Pattern Matching Algorithm", IEEE Int. Conf. on Emerging Trends in Computing, Communication and Nanotechnology, 2013.
[2] William G.J. Halfond and Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 1, JANUARY/FEBRUARY 2008
[3] E. Bertino, A. Kamra, E. Terzi, and A. Vakali, "Intrusion detection in RBAC-administered databases", in the Proceedings of the 21st Annual Computer Security Applications Conference, 2005.
[4] E. Bertino, A. Kamra, and J. Early, "Profiling Database Application to Detect SQL Injection Attacks", In the Proceedings of 2007 IEEE International Performance, Computing, and Communications Conference, 2007.
[5] E. Fredkin, "TRIE Memory", Communications of the ACM, 1960.
[6] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", Computer Science and Engineering,The Ohio State University Columbus, 2005.
[7] J. H. Saltzer, M. D. Schroeder, "The Protection of Information in Computer Systems", In Proceedings of the IEEE, 2005.
[8] Kamra, E. Bertino, and G. Lebanon, "Mechanisms for Database Intrusion Detection and Response", in the Proceedings of the 2nd SIGMOD PhD Workshop on Innovative Database Research, 2008.
[9] S. Axelsson, "Intrusion detection systems: A survey and taxonomy", Technical Report, Chalmers University, 2000.
[10] S. F. Yusufovna, "Integrating Intrusion Detection System and Data Mining", IEEE Ubiquitous Multimedia Computing, 2008.
[11] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL Injection Attacks", College of Computing, Georgia Institute of Technology, 2005.
[12] W. G. J. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL Injection Attacks", College of Computing, Georgia Institute of Technology, 2005.
[13] W. G. J. Halfond, A. Orso, and P. Manolios, "Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks", Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, 2006.
[14] V. Aho and Margaret J. Corasick, "Efficient string matching: An aid to bibliographic search", Communications of the ACM, 1975.
[15] Mahima Srivastava, "Algorithm to Prevent Back End Database against SQL njection Attacks", 2014 International Conference on Computing for Sustainable Global Development (INDIACom).