A Study on Detection and Prevention of SQL Injection Attack

Rashmi Yeole[1], Shubhangi Ninawe[2], Payal Dhore[3], Prof. P. U. Tembhare[4]

[123]Student, Dept. of Computer Technology, Priyadarshini college of Engineering and Architecture Maharashtra, India
[4]Professor, Dept. of Computer Technology, Priyadarshini college of Engineering and Architecture Maharashtra, India

---***---

Abstract - *Various item systems consolidate an electronic section that makes them available to individuals as a rule by method for the web and can open them to a combination of online ambushes. One of these ambushes is SQL imbuement which can give attackers unapproved access to the databases. This paper displays an approach for securing web applications against SQL implantation. Design matching is a structure that can be utilized to perceive or see any anomaly convey a consecutive activity. This paper moreover shows an affirmation and avoidance strategy for guaranteeing SQL Injection Attack (SQLIA) utilizing Aho-Corasick design matching figuring furthermore, it focuses on different parts that can distinguish a couple SQL Injection ambushes.*

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

1. INTRODUCTION

SQL Injection Attacks have been depicted as a champion among the most affirmed perils for Web applications [4] [1]. Web applications that are weak against SQL mixture may permit an attacker to development complete access to their key databases. Since these databases once in a while contain sensitive buyers or client data, the accompanying security infringement can meld markdown blackmail, loss of puzzle data, and contortion. Every so often, aggressors can even utilize a SQL implantation absence of insurance to take control of and break down the system that has the Web application. Web applications that are helpless against SQL Injection Attacks (SQLIAs) are regardless of what you look like at it. To be perfectly honest, SQLIAs have feasibly based on detectable abused people, for example, Travelocity, Ftd.com, and Surmise Inc. SQL mixture suggests a class of code-implantation attacks in which information gave by the client is joined in a SQL request in such a path, to the point that bit of the client's data is overseen as SQL code. By using these vulnerabilities, an attacker can submit SQL summons obviously to the database. These attacks are a certifiable risk to any Web application that gets commitment from clients and hardens it into SQL request to a basic database. Most Web applications utilized on the Web or inside colossal business structures work thusly and could along these lines are defenseless against SQL imbuement. A champion among the most profitable instruments to shield against web strikes utilizes Interruption Discovery System (IDS) and Network Intrusion Detection System (NIDS). An IDS utilizes mistreat or variety from the standard range to guarantee against ambush [3]. IDS that utilization idiosyncrasy affirmation system makes a gage of typical use designs. Misuse recognizing evidence approach utilizes particularly known examples of unapproved prompt to suspect and discover occurring for all intents and purposes indistinguishable sort of strikes. These sorts of examples are called as signature [8][3]. NIDS are not help for the association organized applications (web ambush); in light of the way that NIDS are working lower level layers [4].

2. LITERATURE SURVEY

Beuhrer et. al. [6] has delineated a framework to thwart and to get rid of SQL imbuement ambushes. The technique depends on taking a gander at, the parse tree of the SQL verbalization before fuse of customer commitment with the one that resulting after thought of commitment, at run time. This structure execution is wanted to limit the attempts the designer needs to take; since, it subsequently gets, both the bona fide address and the proposed request and that also, with unimportant changes on a very basic level to be done by the product build. Saltzer and Schroeder [7] propose a security structure against the ambushes like SQL Injection. They proposed a structure using diverse stages. One of them was the shield defaults, on which the positive ruining is poor or takes after, imparts that a traditionalist course of action must be locked in around debate why articles should be open, rather than why they should not. In an expansive framework two or three articles will be inadequately considered, so a default of nonappearance of assent is more secure. A chart or utilize botch up in a section that gives unequivocal agree has a tendency to bomb by declining approval, a shielded condition, since it will be instantly seen. Then again, a setup or utilize botch in a structure that unequivocally rejects get to has a tendency to flop by permitting get to, a disappointment which may go unnoticed in customary utilization. This control applies both to the outward appearance of the affirmation structure and to its hid execution.

Yusufovna [10] has displayed a use of data burrowing approaches for IDS. Intrusion revelation can named as of perceiving exercises that attempt to chance the security, constancy and openness of the benefits of a system. IDS model is displayed and furthermore its confinement in choosing security encroachment are presented in this paper.

Halfond and Orso [11] had presented a development for disclosure and abhorrence of SQLIA. This method made relied on upon the approach that normal to recognize the noxious request before their execution inside the database. To thus manufacture a model of the true blue or right inquiries, the static part of the methodology used the program examination. This could be delivered by the application itself. The strategy used the runtime watching for examination of capably made request and to check them against the static shape show. Halfond and Orso [12] had proposed a technique for countering SQL implantation. The system truly joined the traditionalist static examination and runtime checking for disclosure and stoppage of unlawful request before they are executed on the database. The framework gathers a direct model of the valid inquiries that could be made by the application in its static parts. The framework evaluated the logically created request for consistence with statically build show in its dynamic part. W. G. J. Halfond et. al. [13], proposed another, very motorized method for ensuring existing Web applications against SQL implantation. This strategy has both processed and prudent positive conditions over most existing structures. From the found out point of view, the methodology is locked in around the initially thought to be certain demolishing and the likelihood of dialect structure significant evaluation. From the sensible perspective, the methodology is in the meantime right and helpful and has inconsequential strategy necessities.

3. RELATED WORK

3.1 Types of SQL Injection Attacks

In this fragment, we show and discuss the different sorts of SQL Injection Attacks. The unmistakable sorts of strikes are overall not performed in disengagement; a powerful bit of them are used together or progressively, dependent upon the specific targets of the attacker. Note furthermore that there are unlimited assortments of each ambush sort.

3.1.1 Tautologies

Tautology-based assaults are among the least difficult and best known sorts of SQLIAs. The general objective of a tautology based assault is to infuse SQL tokens that make the inquiries restrictive proclamation dependably assess to true [2]. This procedure infuses proclamations that are constantly genuine so that the inquiries dependably return comes endless supply of WHERE condition [15].

Injected query: select name from user_details where username = "abc" and watchword = or1 = 1.

3.1.2 Union Queries

SQL permits two inquiries to be joined and returned as one outcome set. For instance, SELECT col1,col2,col3 FROM table1 UNION SELECT col4,col5,col6 FROM table2 will return one outcome set comprising of the aftereffects of both inquiries Using this system, an aggressor can trap the application into returning information from a table not quite the same as the one that was planned by the designer. Infused question is connected with the first SQL inquiry utilizing the catchphrase UNION as a part of request to get data identified with different 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]

3.1.3 Piggybacked

In this attack, an intruder tries to infuse extra questions alongside the first inquiry, which are said to "piggy-back" onto the first question. Thus, the database gets numerous SQL questions for execution extra inquiry is added to the first inquiry. This should be possible by utilizing a question delimiter, for example, ";", which erases the table determined [15].

Injected Query: select name from user_details where username = 'abc'; droptable acc –

3.1.4 Timing attack

In this type of attack, the attacker surmises the data character by character, contingent upon the yield type of genuine/false. In time based assaults, assailant presents a postponement by infusing an extra SLEEP (n) call into the question and after that watching if the site page was really by n seconds [15].

3.1.5 Blind SQL injection attacks

Attacker ordinarily tests for SQL infusion vulnerabilities by sending the info that would bring about the server to produce an invalid SQL question. In the event that the server then returns a mistake message to the customer, the aggressor will endeavor to figure out segments of the first SQL inquiry utilizing data picked up from these blunder messages [15].

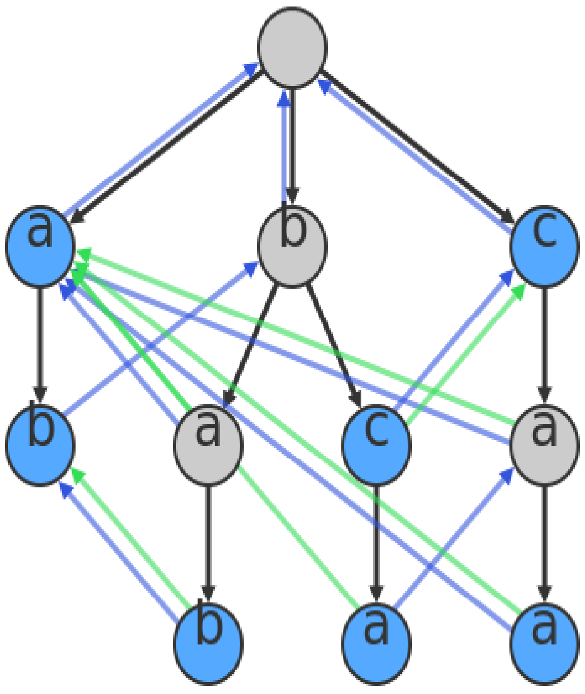
3.2 Aho–Corasick algorithm

In software engineering, the Aho–Corasick calculation is a string looking calculation created by Alfred V. Aho and Margaret J. Corasick. It is a sort of lexicon matching calculation that finds components of a limited arrangement of strings (the "word reference") inside information content. It coordinates all strings at the same time. The unpredictability of the calculation is straight in the length of the strings in addition to the length of the looked content in addition to the quantity of yield matches. Take note of that since all matches are found, there can be a quadratic number of matches if each substring matches (e.g. word reference = an, aa, aaa, aaaa and input string is aaaa).

Casually, the calculation develops a limited state machine that takes after a trie with extra connections between the different inside hubs. These additional interior connections permit quick moves between fizzled string matches (e.g. a look for feline in a trie that does not contain feline, but rather contains truck, and in this manner would come up short at the hub prefixed by ca), to different branches of the trie that share a typical prefix (e.g., in the past case, a branch for trait may be the best sidelong move). This permits the machine to move between string matches without the requirement for backtracking.

At the point when the string word reference is known ahead of time (e.g. a PC infection database), the development of the machine can be performed once disconnected and the assembled robot put away for later utilize. For this situation, its run time is straight in the length of the contribution in addition to the quantity of coordinated passages. The Aho–Corasick string matching calculation framed the premise of the first Unix order fgrep.

Example:
In this illustration, we will consider a lexicon comprising of the accompanying words: {a,ab,bab,bc,bca,c,caa}.
The diagram beneath is the Aho–Corasick information structure built from the predetermined lexicon, with every line in the table speaking to a hub in the trie, with the section way showing the (one of a kind) arrangement of characters from the root to the hub.
The information structure has one hub for each prefix of each string in the word reference. So if (bca) is in the word reference, then there will be hubs for (bca), (bc), (b), and (). In the event that a hub is in the word reference then it is a blue hub. Else it is a dim hub.



A visualization of the trie for the dictionary on the right. Suffix links are in blue; dictionary suffix links in green. Nodes corresponding to dictionary entries are highlighted in blue.
There is a dark coordinated "tyke" bend from every hub to a hub whose name is found by attaching one character. So there is a dark curve from (bc) to (bca).
There is a blue coordinated "postfix" bend from every hub to the hub that is the longest conceivable strict addition of it in the chart. For instance, for hub (caa), its strict additions are (aa) and (an) and (). The longest of these that exists in the diagram is (a). So there is a blue circular segment from (caa) to (a). The blue curves can be processed in straight time by more than once crossing the blue bends of a hub's parent until the navigating hub has a youngster matching the character of the objective hub.

Dictionary {a, ab, bab, bc, bca, c, caa}

Path	In Dictionary	Suffix Link	Dict Suffix Link
()	–		
(a)	+	()	
(ab)	+	(b)	
(b)	–	()	
(ba)	–	(a)	(a)
(bab)	+	(ab)	(ab)
(bc)	+	(c)	(c)
(bca)	+	(ca)	(a)
(c)	+	()	
(ca)	–	(a)	(a)
(caa)	+	(a)	(a)

There is a green "lexicon addition" curve from every hub to the following hub in the word reference that can be come to by taking after blue bends. For instance, there is a green circular segment from (bca) to (an) on the grounds that (an) is the main hub in the word reference (i.e. a blue hub) that is achieved when taking after the blue circular segments to (ca) and afterward on to (a). The green circular segments can be registered in direct time by over and over navigating blue bends until a filled in hub is found, and memorizing this data. At every progression, the present hub is stretched out by discovering its kid, and if that doesn't exist, discovering its postfix's youngster, and if that doesn't work, discovering its addition's postfix's tyke, et cetera, at long last consummation in the root hub if nothing's observed some time recently.
At the point when the calculation achieves a hub, it yields all the word reference passages that end at the present character position in the info content. This is finished by printing each hub came to by taking after the lexicon addition joins, beginning from that hub, and proceeding until it achieves a hub with no word reference postfix connect. What's more, the hub itself is printed, in the event that it is a word reference passage.
Execution on information string abccab yields the accompanying strides:

Analysis of input string abccab

Node	Remaining String	Output:End Position	Transition	Output
()	abccab		start at root	
(a)	bccab	a:1	() to child (a)	Current node
(ab)	ccab	ab:2	(a) to child (ab)	Current node
(bc)	cab	bc:3, c:3	(ab) to suffix (b) to child (bc)	Current Node, Dict suffix node
(c)	ab	c:4	(bc) to suffix (c) to suffix () to child (c)	Current node
(ca)	b	a:5	(c) to child (ca)	Dict suffix node
(ab)		ab:6	(ca) to suffix (a) to child (ab)	Current node

3.3 Proposed System
In web security issues, SQLIA has the top generally need. Fundamentally, we can organize the area and balancing activity strategies into two general classes. In the first place approach is endeavoring to recognize SQLIA through

checking Anomalous SQL Query structure using string matching, design matching and address taking care of. In the second approach uses data conditions among data things which are more unwilling to change for recognizing vindictive database works out. In both the classes, substantial bits of the experts proposed various arrangements with consolidating data mining and interference area frameworks. Hal warm et al [21] developed a methodology that uses a model–based approach to manage distinguish unlawful questions before they are executed on the database. William et al [20] proposed a structure WASP to check SQL Injection Attacks by a procedure called positive dirtying. Srivastava et al [22] offered a weighted gathering burrowing approach for recognizing data base assaults. The dedication of this paper is to propose a procedure for perceiving and envisioning SQLIA using both static stage and element stage. The peculiarity SQL Queries are disclosure in static stage. In the dynamic stage, if any of the request is perceived as anomaly question then new example will be produced using the SQL Query and it will be added to the Static Pattern List (SPL).

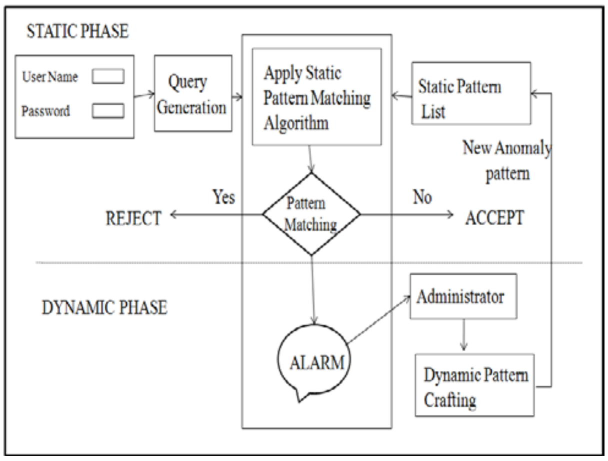


Figure 1: Architecture of SQLIA Detection

CONCLUSIONS

In this paper, we showed a novel system against SQLIAs; we concentrated a plan for affirmation and killing action of SQL Injection Attack (SQLIA) utilizing Aho–Corasick design matching calculation. The investigated plan is assessed by utilizing case of most likely comprehended strike designs. The technique is totally automated and recognizes SQLIAs using a model-based approach that solidifies static and component examination. This application can be used with various databases.

REFERENCES

[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).