Original Article

A REVIEW ON SQL INJECTION, DETECTION AND PREVENTIONS TECHNIQUES

Ashish Mishra[1], Nikhil Mehra[2], Shivendu Dubey[3]

[1]Department of CSE Gyan Ganga Institute of Technology and Sciences, Jabalpur (M.P.), India.
ashishmishra@ggits.org
[2]Research Scholar, Department of CSE Gyan Ganga Institute of Technology and Sciences, Jabalpur (M.P.), India.
mehranikhil839@gmail.com
[3]Department of CSE Gyan Ganga Institute of Technology and Sciences, Jabalpur (M.P.), India.
shivendudubey@ggits.org
Corresponding author-ashishmishra@ggits.org

DOI: 10.47750/pnr.2023.14.S01.148

Abstract

There is several web servers in the world, and numerous website use these web servers to access the World Wide Web. These websites are open to attacks, typically of the input validation variety. These assaults make it simple to breach websites and allow anonymous user to release sensitive information. The open market is in a very risky condition right now. The analysis done as stated and on top that computerized environment leads us do research work on SQLIAs, and top of that this being more oppressed and dangerous invasion techniques, that uses runtime validation for detection of such attacks and track their event. At the same time as we shall also focus on the type of evaluation method which will be generalized and at the same time can be adapted in any existing system.

Keywords- SQL Injection attack,(SQLIA), Web Application, Vulnerability Identification, Vulnerability Mitigation.

I. INTRODUCTION

According to a OWASPD-Open Web Application Security Project. SQL Injection accounts 14-15% of all web application attacks [1]. due to flaws like phishing and social engineering attacks, and service denial Nowadays, attacks are common., we must be aware of the existing attacks. Phishing and email spamming are the most basic Social Engineering attacks.

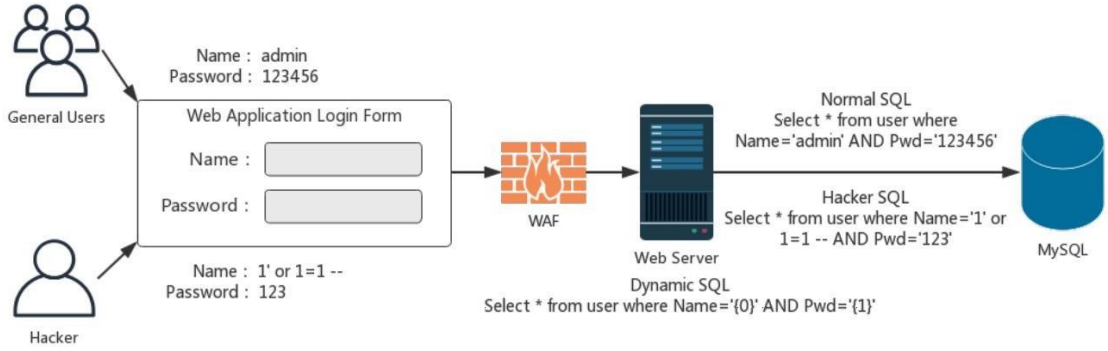
The majority of recent Tab Nabbing is a phishing attack can fool even the most tech-savvy online user [2]. In a web security survey that we conducted with about 100 students, we discovered that nearly 80% were unable to trace phishing attacks. Almost everyone conducts confidential transactions every now and then. OWASP in relation to the web [1]..

Digital transformation has enforced companies to change their business models and adapt to the new IT strategies. All the service business models strongly leverage IT and IT services, where web applications play a crucial role in day to day operations.

A study is being conducted on confidential transactions. Attacks using the Structured Query Language Injection (SQLIA) very common flaw in web security that has posed a serious threat to the internet in recent years, according to the Open Web Application Security Project (OWASP) top ten list published in 2017[1]. SQLIA is a type of code injection attack that exploits a lack of user input. validations.

SQLIA allows an attacker to interface using the legitimate Structured Query Language (SQL) queries are those that an application uses makes to its repository, where the attacker includes a malicious code which is known an injecting a malicious content. Once the SQLIA is successful the attacker can perform operations on the database such as view, insert, delete etc., which might require high level of authorization generally. These unauthorized modifications/ deletions will cause persistent changes to the application's content or behavior. Figure 1. shows the Sql Injection Attack Process.

Figure 1: Sql Injection Attack Process



II. LITERATURE REVIEW:

Explanation: If we carefully dissect the coding architecture of a SQLIA, we discover that the programmer infuses the client a kind of string that can modify the structure of the first inquiry, and each infusion is carried out either in the centre of the question or near the end; implies that the programmer as a rule annex the question. So, if we save all of the data about the framework of the legitimate inquiry then compare it to the powerfully created question, we can confirm that the powerfully created question is a SQLIA or a significant question.

Work Related to: In this section, we list the strategies that are strongly associated with our own and discuss their key points. and SQL queries, but cannot distinguish between different types of SQL infusions assaults.

In light of an AI strategy, [3] Atefeh Tajpour, Suthaimi, Maslin Masrom. Suggested using an interruption identification framework (IDS). IDS is prepared by utilizing a large number of common application inquiries, creating the models of common questions, after that screening at runtime, the application to recognize the inquiries that do not correspond to the model. The general IDS performance is determined by the nature of the preparation set; a poor preparation set would result in numerous false positives and negatives.

SQL rand gives you a structure that enables designers to construct SQL queries using randomized catchphrases rather than the standard SQL watchwords. SQL is captured by an intermediary between the database and the web application. The security of this key depends on aggressors not being able to find it. SQL rand necessitates code changes on the part of the application designer.

Spoil Based Methodology [4]: Ke Wei, M. Muthuprasanna, Suraj Kothari altered a PHP script translator to monitor polluted data in terms of characters. This strategy makes use of a delicate setting. if an unreliable information has been applied to generate specific SQL token categories, the SQL questions are dismissed. One common disadvantage of this methodology is that it expects changes in relation to the runtime condition, which reduces the convey ability.

Dynamic and static analysis combined: S Z. Su and G. Wassermann [5] discuss a sentence structure-based method for recognizing and stopping questions with SQLIAs using the SQLCHECK tool. They add a unique image to client-provided partitions in inquiries and extend the standard SQL sentence structure with a generation rule. The expanded language structure is used to generate a parser. If there are no SQLIAs in the created inquiries after including client inputs, the parser effectively parses the produced question at runtime. This method makes use of Screen code is used

to match the progressively produced question against the inquiry model. If a created question is not devoured by NDFA, it is considered an assault. AMNESIA's precision is dependent on the static examination. The led an investigation utilizing five genuine web applications, as well as two understudy developed web applications, and linked its AMNESIA answer for every application. They discovered their solution came to an end. the majority the SQLIAs in their collection of attacks without producing any possible false positives. Their response tosses an exemption In relation to each SQLIA, the designer handles as well as works the logic of attack and recovery. Regrettably, using this method will necessitate significant changes to the source code. It is required for run time approval.

III. GENERALIZED METHODOLOGY

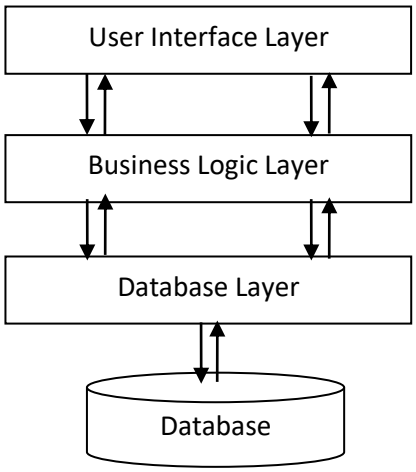
1. Based on the user's input, the system creates specific rules. It then permits traffic in accordance with the rules, to enter or exit.
2. The system also detects and alerts the user to well-known attacks.
3. In order to identify the hotspots where the SQLIA vulnerability arises, we will first talk about the three-tier logical view design of online applications. Web application architecture with three tiers:

(1). The user interface tier is the front end of the web application. Based on the user's input, it interacts with the other layers.

(2). Business logic tier: This tier deals with processing user requests. Server-side programming logic is used forms the middle layer between the database tier and the tier.

(3). Database tier: The database server is included in this tier. It is useful for data storage and retrieval.(Shown in Figure-2)

Figure-2: 3-Tier Web Architecture



3.1. ESSENTIAL SQL INJECTION PRINCIPLES

SQL injection is a web security attack that exploits poorly designed web form input elements by using SQL statements. This puts the confidentiality and integrity of sensitive user data at risk. SQLIA occurs between the UI and the business logic layers. Let us look at an example to better understand SQL injection.

Take a look at an example SQL statement with two input parameters: SELECT * FROM tablename WHERE user='' and password=' '.

When a hacker attempts to gain access to an application by inputting SQL statements rather than the actual username and password, this is referred to as a SQLi attempt. If a hacker enters 'OR '1' = '1' --, the statement is transformed into,

SELECT * FROM tablename WHERE user='' OR '1' For example, limiting the use of special characters. However, in all applications, users cannot be forced to limit their input size or use special characters. Client-side script protection is also easily circumvented. This method can handle tautology and incorrect queries. It is unable to address the threat posed by blind injection techniques. As a result, we prefer server-side validation techniques.

IV. PROCESS:

Shown in figure 3.

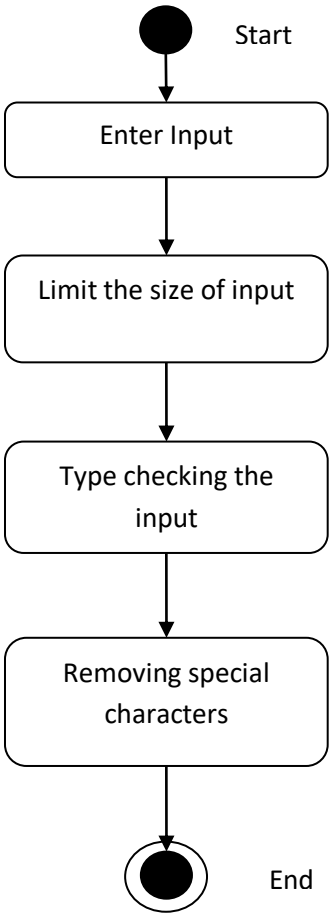


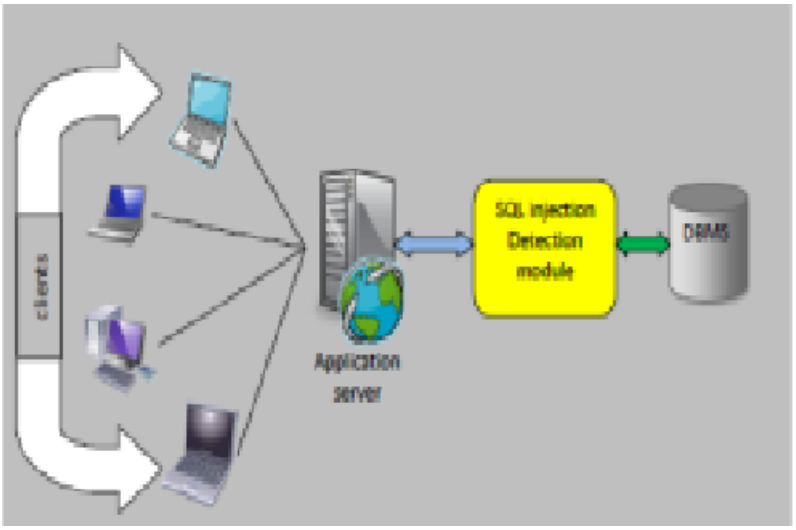
Figure 3: Flow chart of the method.

V. RESULT AND ANALYSIS:

That algorithm's primary benefit is that it makes using a multi-core architecture to minimize the response time. And there is always a difference between predicting and not predicting. By calculating hit count, it predicts the possible list to search in order to minimize response time. This algorithm's main complexity is in three procedures. The first step is token separation, the second is token to integer conversion, and the primary search in the link list is done in the third stage. Because different databases have distinct key word sets, function names, and syntax, token separation depends on the database we are using. Token separation is used by all

current implementations. The complexity of the token to integer conversion procedure is O (n), where n is the total number of literals in all of the query's tokens, due to the fact that each literal must be scanned exactly once in order to obtain its Ascii decimal value. In the worst case scenario, every solitary link list must be traversed in the third method, the primary searching operation, if the mismatch occurs at the end of every list and the search is unsuccessful. Shown in figure 4.

Figure 4: Proposed System Architecture.



TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME DATA_TYPE

Table 1:- SQL injection

==========

Data base1	dbo1	Steve	UserId	int
Data base2	dbo2	Naveen	UserName	varchar
Data base3	dbo3	Patrick	Password	varchar

Table for SQL injection shown in table-1.

VI. CONCLUSIONS:

In general the web applications use a technology that is designed in such a manner so as to ask for information from a RDMBS in SQL parlance. SQL injection is a widespread method being employed by hackers for attacking the web-based applications. With the attacks of such type the SQL query gets modified, and so is the performance of the program affected and thus is an advantage for the hacker.

Over here we present a method for distinguishing and putting a check on SQLIA incidents. The technique extracts the planned SQL query behavior for an application which will be in the form of a prearranged series of tokens, as a onetime offline Process which uses stagnant scrutiny of the application code. Now the database is then certified against the complete diverse incoming SQL query at runtime for capturing all nasty SQL queries,

just before they are sent to the database server for execution. Modern processor architecture can be used to perform the search in a multi threaded manner It can predict all potential right lists for the incoming query by intimating the hit count calculation, which minimizes the search time and reaction time. By transforming the token to its corresponding integer value and saving the tokens in the database as integer values, we may avoid the additional calculation costs associated with string matching. This method will help to efficiently confine all of the many types and modes of SQLIA execution in a way that is clear and doesn't require any modifications to the underlying application source.

Since PL/SQL code blocks cannot be handled by this security module, which can only manage a single SQL query, we will eventually enhance our solution to include them. We will work to develop our approach to address all of those problems because, as was previously said, this technique has a disadvantage in that it cannot identify such types of assaults produced by merely changing the data value of a query without affecting the structure of the query.

VII. REFERENCES

[1]. OWASPD-Open Web Application Security Project. "Top ten most critical Web OWASPD-Open Web Application Security Project. "Top ten most critical Web Application Security Risks", https://www.owasp.org/index.php/Top_10_2010-Main.

[2]. W. G. Halfond, J. Viegas, and A. Orso. A Classification of SQLInjection Attacks and Countermeasures. In Proc. of the Intl. Symposium on Secure Software Engineering, Mar. 2006.

[3]. Atefeh Tajpour, Suthaimi, Maslin Masrom. SQL Injection Detection and Prevention Techniques .In Proc. International Journal of Advancements in Computing Technology Volume 3, Number 7, August 2011.

[4]. Ke Wei, M. Muthuprasanna, Suraj Kothari , "Preventing SQL Injection Attacks in Stored Procedures" Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06 IEEE).

[5]. Z. Su and G. Wassermann "The essence of command injection attacks in web applications". In ACM Symposium on Principles of Programming Languages (POPL'2006), January 2006.

[6]. S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pages 292–302, June 2004.

[7]. William G.J. Halfond and Alessandro Orso," Preventing SQL Injection Attacks Using AMNESIA" ICSE'06, May 20–28, 2006, Shanghai, China ACM 06/0005.

[8]. G. Buehrer, B.W. Weide, P.A.G. Sivilotti, Using Parse Tree Validation to Prevent SQL Injection Attacks, in: 5th International Workshop on Software Engineering and Middleware, Lisbon,Portugal, 2005, pp. 106–113.

[9]. R.A. McClure, and I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 88- 96, 15-21 May 2005.

[10]. P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Trans. Inf. Syst. Secur., 13(2):1–39, 2010.

[11]. Shaukat Ali, Azhar Rauf, Huma Javed. SQLIPA: An Authentication Mechanism Against SQL Injection. In Proc. European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4 (2009), pp 604-611.