Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.7, No.6, 2016

A Discriminative Survey on SQL Injection Methods to Detect Vulnerabilities in Web applications

Nrottam Chaubey[1], Sumit Sharma[2]
1.PG scholar, CSE, VIST, Bhopal, INDIA
2.HOD, CSE department , VIST, Bhopal, INDIA

Abstract
SQL Injection Attacks are extremely sober intrusion assaults on web based application since such types of assaults could reveals the secrets and safety of information. In actuality, illegal personnel intrude to the web based database and then after consequently, access to the information. To avoid such type of assault different methods are recommended by various researchers but they are not adequate since most of implemented methods will not prevent all type of assaults. In this paper we did survey on the various sorts of SQL Injection attacks and on the various present SQL Injection Attacks avoidance methods available. We analyzed that the existing SQL Injection Attacks avoidance methods will require the client side information, one by one and then authenticate which will create typical the developer's job to write different validation codes for every web page which is receiving in the server side.
Keywords: SQL Injection, Attacks, Vulnerability, WWW, XSS

1. INTRODUCTION

The tremendous use of the Web as an quick means of data dissemination and various other transactions including those having financial drawbacks (consequences), has essentially made it a key component of today's Internet architecture. These applications and their underlying data store keep necessary secure data. The small mismanage will leads to the millions dollars loss which could greatly affect the clients. Hence, it is necessary to protect these applications from targeted assaults now a day's many activities are done by dynamic web application. For example several people pay their bills, book the hotels (restaurants) or and give the online exam through the dynamic websites instead of spending time for commuting. It is very necessary that the private information of the client should follow the CIA trade (i.e. Confidentiality, Integrity and Authentication)information of people must be kept secret and confidentiality and integrity of them must be provided by developer of web application but unfortunately there is no any guarantee for preserving the underlying databases from current assaults [1].

Web applications are vulnerable (insecure) to outside assaults, in which unauthorized person easily threaten the application's underlying database. In the 2002 Computer Security Institute and FBI both these organizations presented a scenario in which, on a yearly basis(year by year), over half of nearly all web data stores have to pass through at least one security breach(web assault) .In the survey performed by the Imperva a center for defense services which have included more than 250 Web applications from enterprise collaboration, online banking, e-commerce, and supply chain management sites and their vulnerability assessment shows that at least 92% of Web applications are insecure to some form of malicious intrusions [2]. Represent U.S. industry maintenance such as the Sarbanes-Oxley Act pertaining to information security, try to force very strict security compliance by application vendors [3] and there is a great need to found means of satisfying these security requirements.

The Structural Query Language Injection (SQLI) assault were hired when an assaulter changes the logic, semantics or syntax of a SQL query by inserting new malicious SQL keywords or operators. When there is no input validation phenomena SQL Injection Assaults are hired. In reality, assaulters can shape their illegal input as component of final query that will operate by databases. Web applications or confidential information systems could be the victims of this vulnerability because assaulters by abusing this vulnerability can threat their authority, integrity and secretly. So, programmers should utilize some different coding pattern to avoid this vulnerability but they are not sufficient enough. SQLIAs are also capable of escaping traditional tools such as firewalls and various Intrusion Detection Systems (IDSs) because they are hired through workstation ports utilized for regular (permanent) web flow (traffic) usually are open in firewall. On the other side many IDSs aims on the network layer and IP layers for the security but SQLIAs occupies their place at application layer.

Many analysts have proposed a range of technique to get a rid of from these assaults through defensive coding style [4], [5]. SQL-Injection Attack (SQLIA) constitutes as one of the great assaults against web applications. Due to the deficiency in input validation, an assaulter can be able to directly access the data store. Any web application exposed on the WWW or even in the simple intranet could be vulnerable to SQLIAs. Although the major causes that lead to SQLIAs are well known, but they still exist because of lack of effective mechanism for detecting and preventing them. Secure way of programming could, protect against various types of SQLIAs. This entire process for protecting Several approaches have been proposed in the studies to prevent

SQLIAs in the application layer, which will combine the two procedures firstly combine static analysis of application level programs and secondly runtime validation of dynamically generated SQL-queries with inclusion of client inputs.

Although these approaches will prevent SQLIAs at the application layer, very small importance is given on securing objects kept in the database layer such as stored procedures which are also largely vulnerable to SQL Injection Attacks. Stored procedures occupies important place in the present -day relational database stores. They are responsible for adding the extra layer of abstraction level into the design of a software system, i.e. This extra layer, at the particular degree, secure some design confidential from the potentially un authorized personnel's, such as relational table designs. By using the mechanism of stored procedures, one can be ensured that the information is stored in the data store safely. In these databases, the developer utilize dynamic SQL queries i.e., SQL statements are built at runtime according to the different client inputs. On the basis of this variable feature degree to construct SQL statements according to different requirements can be easily managed, but have a threat from SQL Injection Attacks, the problem is the impractical nature of various present techniques because they could not address all assault types or have not been implemented yet. Also some of them require modifying web application code or extra working source. However, the main aim of this paper is to introduce all types of SQL injection attacks and to evaluate present approaches which will detect and then after prevent these assaults.

Rest of the paper is organized as follow: section 2 describes about SQL injection and its various attacks, in section 3 we gives detail analysis of detection and prevention techniques of SQL injection, section 4 presents motivation of our study and finally we concluded our paper in section 5.

2. SQL INJECTION (SQLIA)

In SQL injection attack the intruder adds Structured Query Language code to login box of a web form to make the modifications. SQL injection vulnerability allows an assaulter to perform malicious functioning directly to a web application's underlying database and destroy secretly.

A. VARIOUS SQL INJECTION ATTACK TYPES

The various forms of SQLIAs are possible. For a successful SQLIA the unauthorized person should put a semantically and syntactical correct command to the inventive SQL query. Currently the following types of SQLIAs in accordance to the researches [6], [7] are presented.

Tautologies:

In such category of assault the intruder injects SQL tokens (malicious keywords) to the conditional query statement which will evaluate always true. This type of assault utilized to bypass security control and to gain access to information by avoiding vulnerable input which utilize the WHERE clause in the SQL.

"SELECT * FROM customer WHERE custid ='111' and password ='bbb' OR '1'='1'" As the tautology statement (1 = 1) has been included to the query it will always evaluated to be true Illegal/Logically Incorrect Queries: are rejected, and message is returned back from the database which will provide the necessary useful debugging information. This error messages help the intruder to guess the parameters which will then after be utilized by the intruder to hire the assaults on the target web application. In the example below assaulter makes a type mismatch error by injecting the following text into the pin input:

Original URL:http://www.krch.polimLitieventil?idnav=8864

SQL Injection: http://www.krch.polimi.itleventil?idnav=8864'Error message showed: SELECT name FROM Customer WHERE id =7764\' from the message error given below we can identify out the entries of customer fields will be returned from the retrieved information the assaulters can easily hire the assaults.

Union Query: The UNION keyword is utilized in such queries which will join the malicious query with the simple one

Piggy-backed Queries: The query delimiter mechanism is utilized in this type of assault, intruder make the breach in security of database by appending extra other query to the original query. If the assault is proper then it shows its impact .The first query will get easily evaluated as true, whereas following queries could be illegitimate. The intruder can inject any SQL command to the web store in our example, assaulter inject "0 character; drop table client" into the pin input instead of Boolean logical value. Then the web application would produce the query of the following format:

SELECT info FROM clients WHERE login='pre'

AND pin =000; drop table clients Because of ";" character, database recognize both queries

Now executes both these queries. The second query is illegal in nature from the database prospective. One can discover that the database will not require separation character in various queries, so to discover this type of assault, to scan the special character is not very effective solution.

Stored Procedure:

Stored procedure is the extra abstraction layer which is a part of database that Developer could set an extra on the database store .The programmer is responsible for coding the stored variables , so, this part is as inject able as

web application forms, different stored procedures have the different way to be get breached
. In the practical example, assaulter exploits
Parameterized stored procedure CREATE PROCEDURE DBO. Is Authenticated @clientId varint2, @pwd varint2, @pin char
AS EXEC("SELECT accounts FROM clients
WHERE login= '"+@clientName+"' and
pass='"+@password+"' and pin="+@pin);
GO
The stored procedures will give the true or false on the basis of validation. Since in SQLIA, intruder input ", SHUTDOWN; - -" for clientname or password. Afterward the stored procedure produces the following query SELECT accounts_no. FROM clients WHERE login='doe' AND pass =; SHUTDOWN; -- AND pin= on the execution of this type of assault which works as piggy-back assault. The query will be evaluated and the after the second which is illegitimate and executed and causes database shut down. So we can simply say that the stored procedures are vulnerable to the various malicious assaults on the internet as web application.

Blind Injection:
The error generated messages which will provide some needed information to unauthorized personnel to hire the assaults will be hiding by the developer. In which the generic page will be shown to the intruder. So the SQLIA would be more difficult but we cannot say that it is not impossible. The intruder can still breach information by asking a series of True/False questions through SQL declaration. Suppose two possible injections into the
login:
SELECT ids FROM clients WHERE
1 =0 -- AND pass = AND
login='doe' and
Pin=0
SELECT ids
login='doe' and
pin=0
If the application nature is secured, both queries above queries will not get successful, because of the strong input validation. But if there is the situation of no input validation, then the unauthorized person can try the chance. In the First step the illegal accessing client submit the query and receives an error message in the output because of "1=0".and at this point the intruder is not able to understand the reason for error weather it is because of logical error or input validation . Then the assaulter submits the second query which forever true. But in case of no login error message, then the assaulter identifies the login vulnerable to injection.

Timing Attacks:
Due to the delay in the responses from the database the intruder can discover some information to hire the assaults. In this technique we will uses the if-then statement which cause to SQL evaluation engine to execute a long running query or we can say the a time delay statement on the basis of injected logic . This category of assault is very similar to the blind injection and the intruder can then measure the time the page takes to load and the after evaluate whether the injected statement is accurate. This method utilizes an if-then statement for injecting queries. W AITFOR is a keyword will be utilized to cause the indefinite delays.

Alternate Encodings:
Alternate coding mechanisms such as hexadecimal, ASCII, and Unicode etc will be utilized in the injection query by using alternate encoding, such as. And by this ways the intruder can easily bypass its injected query in the web storage space". For instance, assaulter exercises char (44) in place of single quote, which will denote the bad character. This technique will be very dangerous if it will be implemented in the joint majority with the other technique, because it can easily target various layers of the target application, so the developers have to keep mind to avoid such assaults (alternate coding).

3. SQL INJECTION DETECTION AND PREVENTIONTECHNIQUES

As the deploy of defensive coding or OS hardening but they are not enough to prevent SQLIAs to web applications so researchers have proposed some of techniques to assist developers. Huang and colleagues [8] propose WAVES, a black box technique for testing web applications for SQL injection deficiencies the tool identify all points a web application that can be utilized to insert SQLIAs. It construct assaults that aim these points and observe the application how response to the assaults by utilize machine learning IDBC-Checker [9] was not invented for the purpose of detection and prevention As most of the SQLIAs contains the of syntactical and type correct queries format so this technique would not catch more general forms of these assaults. Wassermann and Su have proposed Tautology Checker [10] that utilizes static analysis to prevent tautology assault. The important limitation of this technique is that its scope is limited to tautology and cannot detect or prevent other types of assaults Xiang Fu and Kai Qian [11] proposed the design of a static analysis framework, called SAFELI for identifying SQLIA vulnerabilities at compile time. SAFELI statically monitor the MSIL (Microsoft Symbolic intermediate language) is byte code of ASP .NET web applications using symbolic execution. SAFELI can analyze the source code and will be able to identify delicate vulnerabilities that cannot be discovered by black-box vulnerability scanners. The main drawback of this technique is that this approach can discover the SQL injection assaults only on Microsoft based product.

CANDID Modified the web applications which are written in Java through the program transformation method [6,4]. This tool dynamically mines the programmer-intended query structure on any input and detects assaults by comparing it against the structure of the actual query issued. CANDID's natural and easy procedure turns out to be very efficient for detection of SQL injection vulnerabilities. In SQL Guard [12] and SQL Check [13] queries are checked at runtime based on a model which is expressed as a grammar which inputs legal requests only. SQL Guard determines the pattern of the query prior and after the addition of client- input based on the model. In SQL Check, the model is specific independently by the developer. These mechanisms uses the secure secret key to bound the client input during parsing check phenomena done by runtime checker, so security of the approach depends on the inability of the intruder to detect the key. In these mechanisms developer should modify code to utilize a special intermediate library or physically inserts special keywords into the code where the value which is input by the client is composed with dynamically generated query. AMNESIA combines static analysis and runtime monitoring [14], [15]. Queries are intercepted before they are sent to the database and are checked against the statically built models, in dynamic phase. Queries that violate the model are prevented from accessing to the database.

The limitation of this tool is that it is completely dependent on the accuracy of its static analysis framework for building query models. WebSSARI [16] utilize static analysis to observe taint flows against preconditions for various types of sensitive functions. It works based on sanitized input that has passed through a pre defined set of filters. The limitation of approach is adequate preconditions for sensitive functions cannot be accurately expressed so some filters may be omitted. Livshits and Lam [17] utilize static analysis techniques to detect vulnerabilities in software. Java Static Tainting utilizes information flow techniques to detect when tainted input has been utilized to make a SQLIA. The primary limitation of this approach is that it can detect only known patterns of SQLIAs and it can generate a relatively high amount of false positives because it utilizes a conservative analysis. Java Dynamic Tainting [18] is another tool that was implemented for java. Despite of other tool, chase string instead of character for taint information and try to sanitize the large query strings which have been generated by the tainted input but unfortunately injection in numeric fields cannot prevent by this approach Difficulty of identifying all sources of client input is the main limitation of this approach. Two similar approaches by Nguyen-Tuong [19] and Pietraszek [20] modify a PHP interpreter to track precise per- character taint information.

A context sensitive analysis is utilized to detect and reject queries if certain types of SQL tokens has been constructed by illegitimate input. Limitation of these two approaches is that they require rewriting code. SQL DOM [21] utilizes database queries encapsulation for trustable access to databases. They utilize a type-checked API which cause query building process is systematic. Consequently by API they apply coding best practices such as input filtering and strict client input type checking. The drawback of the approaches is that developer should learn new programming paradigm or query-development process. Positive tainting [22] not only focuses on positive tainting rather than negative tainting but also it is automated and does need developer intervention. IDS [23] utilize an Intrusion Detection System (IDS) to identify SQLIAs, depends on a machine learning method. The technique builds models of the typical queries and then at runtime, queries that do not match the model would be recognized as assault. This tool discovers assaults successfully other than it depend on training seriously.

Else, many false positives and false negatives would be generated. Another approach in this category is SQL-IDS [5] which focus on writing specifications for the web application that explain the required structure framework of SQL statements that are generated by the applications. A proxy filtering system that intensifies input validation rules on the data flowing to a Web application is called Security Gateway [24]. In this technique for transferring parameters from web-page to application server, developers should utilize Security Policy Descriptor Language (SPDL). So developer should know which data should be filtered and also what patterns should apply to the data. SQLPrevent [25] is consists of an HTTP request interceptor. The original data flow is modified when SQLPrevent is deployed into a web server. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module. Consequently, HTTP request from thread local storage is fetched and examined to determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLIA. Swaddler [26] analyzes the internal state of a web application. It works based on both single and multiple variables and shows an impressive way against complex assaults to web applications. First the approach describes the normal values for the application's state variables in critical points of the application's components. Then, during the detection period, it observed the application's execution to recognize abnormal states.

There are several other techniques have been proposed in literature to avoid SQLIAs which when utilized combined with some other techniques can be prove to be very efficient .

4. MOTIVATION

The above presented SQL injection prevention techniques verifies authenticity of each data element individually .consider an example, if any client Inputs to a login form than the various server will verify the given client name and given password separately. As we know their growing demand of web application and the complicated data requirements the data submitted by the client is also become complicated. In present scenario we can see lot of complex HTML form that client has to fill up and submit. For example, a simple medical store needs to enter various product details into one form. Hence, we can easily understand the complexity of data validation into server side. In a simple web application each individual page has different types of HTML form which is essential because each form is dedicated for different work, such as client registration, product creation, product purchase etc. Now existing SQL injection prevention techniques handles each data separately.

That makes the developer job complicated. The developers have to write code for the validity checking of each html-form separately. Since each form has different types of complexity it is currently not possible to make the validating process as generic for all the forms. This also makes the maintenance job difficult as the data validation policy is different in each from submitting server code Different web applications manage the validation rules differently., Also some framework available which allows automated server-side validation and the developer can do very less work to manage it. But complex validation requirements cannot be satisfy by all those automatic validating systems. Also for most of these validating systems, developers have to enter validation rules separately. Sometimes we need to send multiple information's to the server. For example, online customer can purchase 10 items at a time. In that case 10 insertions required in the corresponding database table.

5. CONCLUSION

Web applications are make threats by SQL Injection Attacks (SQLIAs) because this kind of assault could compromise privacy and integrity of information in databases. To prevent this sort of assault many methods have been projected by so many researchers but they are not enough because most of these methods could not prevent all type of assaults. In this survey paper we have examined different types of SQL Injection attacks and also the study different existing SQLIAs detection and prevention techniques already available. We have shown that the existing SQLIAs detection and prevention techniques could validate the client side data in singular manner. This makes difficult the developer's job to write validation codes for every query receiving on page on the server. In Future researches can we done on the method in which a single tool is used to detect injection attacks in web based application written in different languages.

REFERENCES

1. [1] C. S. Institute. Computer crime and security survey (2002)
2. [2] W. Inc Only 10% of web applications is secured against common hacking techniques. . (2004)
3. [3] K. Beaver. Achieving sarbanes-oxley compli-ance for web applications through security testing (2003)
4. [4] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: dy- namic candidate evaluations for automatic prevention of SQL injection attacks," ACM Trans. Inf. Syst. Secur., vol. 13, no. 2, pp. 1–39, 2010.
5. [5] K. Kemalis and T. Tzouramanis, "Sql-ids: a specification-based approach for sql-injection detection," in Proceedings of the 2008 ACM symposiu on Applied computing, ser. SAC '08. ACM, 2008, pp. 2153–2158.
6. [6] S. Bandhakavi, P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "Candid: preventing sql injection attacks using dynamic candidatevaluations," in Proceedings of the 14th ACM conference on Compute and communications security, ser. CCS '07, 2007, pp. 12–24.
7. [7] X. Jin and S. L. Osborn, "Architecture for data collection in database intrusion detection systems," in Proceedings of the 4th VLDB conference on Secure data management, ser.
8. SDM'07, 2007, pp. 96–107.
9. [8] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai, "Web application security assessment by fault injection and behavior monitoring," in Proceedings of the 12th international conference on World Wide Web, ser. WWW '03, 2003, pp. 148–159.
10. [9] G. Wassermann, C. Gould, Z. Su, and P. Devanbu, "Static checking of dynamically generated queries in database applications," ACM Trans. Softw. Eng. Methodol., vol. 16, no. 4, Sep. 2007.
11. [10] G. Wassermann and Z. Su, "An analysis framework for security in web applications," in In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2004, 2004, pp. 70–78.

12. [11] X. Fu and K. Qian, "Safeli: Sql injection scanner using symbolic execution," in Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications, ser. TAV-WEB '08, 2008, pp. 34–39.
13. [12] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent sql injection attacks," in Proceedings of the 5th international workshop on Software engineering and middleware, ser. SEM '05, 2005, pp. 106–113.
14. [13] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," SIGPLAN Not., vol. 41, no. 1, pp. 372–382, Jan. 2006.
15. [14] W. G. J. Halfond and A. Orso, "Amnesia: analysis and monitoring for neutralizing sql-injection attacks," in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ser. ASE '05, 2005, pp. 174–183.
16. [15] ——, "Combining static analysis and runtime monitoring to counter sql-injection attacks," SIGSOFT Softw. Eng. Notes, vol. 30, no. 4, pp. 1–7, May 2005.
17. [16] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in Proceedings of the 13th international conference on World Wide Web, ser. WWW '04, 2004, pp. 40–52.
18. [17] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in java applications with static analysis," in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14, ser. SSYM'05, 2005, pp. 18–18.
19. [18] V. Haldar, D. Chandra, and M. Franz, "Dynamic taint propagation for java," in Proceeding of the 21st Annual Computer Security Applications Conference, ser. ACSAC '05, 2005, pp. 303–311.
20. [19] A. Nguyen-tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans, "Automatically hardening web applications using precise tainting," in In 20th IFIP International Information Security Conference, 2005, pp. 372–382.
21. [20] T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," in Proceedings of the 8th international conference on Recent Advances in Intrusion Detection, ser. RAID'05, 2006, pp. 124–145.
22. [21] R. A. McClure and I. H. Kr¨uger, "Sql dom: compile time checking of dynamic sql statements," in Proceedings of the 27th international conference on Software engineering, ser. ICSE '05, 2005, pp. 88–96.
23. [22] W. G. J. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter sql injection attacks," in Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, ser. SIGSOFT '06/FSE-14, 2006, pp. 175–185.
24. [23] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of sql attacks," in Proceedings of the Second international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, ser. DIMVA'05. Springer-Verlag, 2005, pp. 123–140.
25. [24] D. Scott and R. Sharp, "Abstracting application-level web security," in Proceedings of the 11th international conference on World Wide Web, ser. WWW '02, 2002, pp. 396–407.
26. [25] P.Grazie, "Phd sqlprevent thesis," Ph.D. dissertation, University of British Columbia(UBC) Vancouver, Canada, 2008.
27. [26] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An approach for the anomaly-based detection of state violations in web applications," 2007.