International Journal of Computer Applications in Engineering Sciences
[VOL I, ISSUE IV, DECEMBER 2011] [ISSN: 2231-4946]

Font Level Tainting: Another Approach for Preventing SQL Injection Attacks

V. Krishna Pratap[1], S. Tulasi Prasad[2], Ch. Srinivasa Rao[3]

[12]*Computer Science and Enginnering, JNTU Kakinada, Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh. India*
[3]*Computer Science and Enginnering, JNTU Kakinada, Sri Sarathi Institute Of Engineering And Technology, Nuzvid, Krishna Dt, Andhra Pradesh. India*.

[1]pratapv9@gmail.com,
[2]tulasiprasadsariki@gmail.com,
[3]srinivascsitengg@gmail.com

***Abstract*- the font level tainting is the another new approach for preventing sql injection attacks, that involves comparing the meta strings library with the sql statements that includes the characters including the different font levels in the user input, to prevent them if found any and protecting the web applications against sql injection is discussed in this paper. this paper includes the strange idea of combining the declarative method and the quest method. sql injection is the main problem that occurs with web application security. it gives the attackers unauthorized access to the database containing the web applications which in turn leads to the cause of defects in the web applications. this is very serious. in declarative method point of view, it exhibits detection mode for sql injection, that uses the coupled way routing arrangement of amino acid code formulated from web application form parameter sent via the web server. on the other hand from the quest method point of view, it analyzes the transaction to find out the malicious access. in declarative method it uses an approach called schatten algorithm, not only to prevent the sql injection attacks, but also reduces the time and space complexity. this system was able to stop all of the successful attacks**

***Keywords*— sql injection, security, prevention, declarative method, schatten algorithm, dbms quest**

I. INTRODUCTION

The modern web world always expects the organization to concentrate more on providing the security on web application. The Dangerous situation faced by all the organizations is to protect their most valuable data against malicious access, Interrupts and corruptions, i.e. the Attacks. The developers of the programs will have to show their keen interest in developing the applications with usability than combining the security policy rules. The major security issue is an *Input validation issue*, if an attacker finds that an application makes unfounded assumptions about the type, length, format, or range of input data. Then the attacker can furnish a malicious input which compromises an application. The public interfaces exposed by an application become the only source of attack, when a network and host level entry points are fully secured. The SQL Injections attacks, Buffer Overflow attacks and cross site scripting attacks, are the major threat in the web application security through this input validation security issues [11]. The SQL Injection attacks breach the database mechanism such as Integration, Authentication, Availability and authorization [8]. There are many number of total cyber vulnerabilities in the Web world which are input validation vulnerabilities.

Since 2002, nearly 20% of the input validation issues are SQL Injection vulnerabilities (SQLIV's) and therefore, 10% of total cyber vulnerabilities. SQL injection attacks involve placing SQL statements in the user input for corrupting or accessing the Database [11]. Even the SQL Injection attacks can bypass the security mechanism such as Firewall, cryptography and traditional Intrusion detection systems. If the trend of providing web-based services continues, the prevalence of SQLIV's is likely to increase. The most worrying aspect of SQL Injection attacks is Easy to perform, even if the developers of the application are well known about this type of attacks. The basic idea behind in this attack is that the attacker will copy the data that a web application sends to the database aiming at the modification of SQL Query that will be executed by DBMS software. Input validation issues can allow the attackers to gain complete access to such databases.

The Technologies vulnerable to SQL Injection attacks are Dynamic Scripting Languages like ASP, ASP.net, PHP, JSP, CGI etc. In addition, all types of database have been severely vulnerable in such type of SQL Injection Attacks. Researchers have proposed a different techniques to provide a solution for SQLIAs (SQL Injection attacks), but many of these solutions have limitations that affect their effectiveness and practicality. Researchers have indicated that solution to

these types of attacks may be based on defense coding practices. But it's not efficient because of three reasons. First, it is very hard to bring out a rigorous defensive coding discipline. Second, many solutions based on defensive coding address only a subset of the possible attacks. Third, legacy software poses a particularly difficult problem because of the cost and complexity of retrofitting existing code so that it is compliant with defensive coding practices. In this work, an attempt has been made to increase the efficiency of the above techniques by a combinatorial approach for protecting web application against SQL Injection attacks.

The remainder of the paper is organized as follows: Section2 contains background and related work; Section3 describes our proposed approach. Section4 describes the conclusion and future work.

II. BACKGROUND AND RELATED WORK

SQLIA's is one of the main issues in database security, which easily affects the database without the knowledge of both the user and the database administrator. It is a technique that may corrupt the information in the database i.e. deletes or changes the full database or records or tables. To exploit the database system, some vulnerable web applications [9] are used by the attackers.

These attacks not only make the attacker to breach the security and steal the entire content of the database but also, to make arbitrary changes to both the database schema and the contents. SQL injection attack could not be realized about information compromization until long after the attack has passed in many scenarios, the victims are unaware that their confidential data has been stolen or compromised. SQL Injection attacks can be performed by attackers [12] just with the help of simple web browser. The following section describes the attacks with an example.

Generally the Authenticated users have username and password such as,

Username: john

Password: 1000

The SQL Query format will be as follows,

Select * from table where username='john' and pwd='1000';

The above query then retrieves the needed records from the database where username and pwd is available in the database or it shows some error messages to the browsers. The unauthorized users or the attackers inject the following SQL Injection in this field:

Username: john

Password: 1000

Then the dynamic SQL query constructed from the above information is,

Select * from table where username='john' and pwd=1000;

In this SQL statement, the actual username is '*john*'. And the modified name did by the attackers while generating the Query is 'john'. Here both the words are same considering whether they are characters or not. But the first word 'john' is fully taken in the font *"Times new Roman"* format, Where as the modified name given by the attacker, 'john' includes the alphabets '*j', 'o', 'n'* in '*Times new Roman'* format, and the alphabet 'h' in *"Wide Latin" format.* Hence, the attacker will now easily attack the database just by transferring the malicious code. Here both the usernames are set of characters so that even by using the Character Level Tainting, all the characters in the given query is matched with the strings in the Meta Strings Library and then the query was sent to the database. Here the Character Level Tainting can't be able to find the type of malicious code that was sent in different fonts because of having no information on the consideration of the font formats of the sent query and hence it performs the operation. The result of this query performs SQL Injection attacks.

A. Exploiting INSERT

The Web Sites like Banking, when registration, it allows the user to feed inputs and store it. INSERT statement allows the user input to store in the back end. The misuse of INSERT statements by the attacker results in many rows in the database with corrupt data.

B. Exploiting SELECT

SQL injection is not only a straight forward attack but also it has some background tricky attack is present. Most of the time attackers would see some error message and will have to reverse engineer their queries.

C. Authorization diversions (SQL manipulation)

This attack allows the attacker access the total information in the database [15]. The example of this attack is discussed in the above section.

Direct Vs Informative (SQL manipulation):–

Both Direct and informative are the types of SQL Injection attacks in SQL manipulation. In direct attack, the input data become part of the SQL statement formed by the application. Attacker will change font format of the characters in a way that the modified character that was

easily matched with the character in Meta Strings Library can manipulate the SQL statement. The error message has been returned if the injection was successful.

In Striking injection, the injected string has the code at the extra location (width) occupied by the modified character in the statement. In order to manipulate the SQL statement successfully input string must contain its characters in different font formats which is same as the characters in Meta Strings Library and should also contain the malicious code at the extra location (width) occupied by the modified character that attacks the DB whenever this modified character is considered by the database.

D. Exploiting System Stored Procedures (Function call)

Database uses stored procedures to perform database Administrative operations. Attacker uses stored procedures to corrupt the database system and it's a most harmful attacks. If attacker is able to inject SQL string successfully then attacker can make use of these stored procedures. Access to these procedures depends on the privileges of the user on the database.

SELECT usrid, details, username from user where username like 'john'; to execmaster.dbo.xp_cmdshell "dir.

The above injected SQL query will then enters easily into the details of "John" in the database and can easily make transactions or can perform malicious actions, execute the operating system command DIR.The following subsection describes in details about the related works based on SQL Injection Attacks. A number of researches had been taken to provide solutions for SQL Injection Attacks.

R. Ezumalai and G. Aghila [1], has proposed the character level tainting approach for preventing SQLIAs. It includes three modules to detect the security issues. 1. Monitoring module has got the statement from the web application which can decide whether it can send the statement to database for execution. 2. Analysis module uses Hirschberg algorithm to compare the statement from the specifications. 3. Specifications comprise the predefined keywords and send it to analyze module for comparisons. It performs well for finding the malicious code if any, by checking the given statement in terms of characters. If it founds any wrong character other than the specified, it just blocks the query and gives the information of the type of the attack.

Konstantin's et al [3], proposed a mechanism to detect SQL injection with novel-specification based methodology. This approach utilizes specifications that define the intended syntactic structure of SQL queries that are produced and executed by the web application. The main disadvantage of this paper is, to compare the SQL statement with the predefined structure at run time, the computational time is overhead.

Wassermann and Su [6], proposed a static framework to analyze and filter the user inputs. According to them, their approach has restricted to discover only logic-based attacks. I.e. Attacks that always result in true or false SQL statements.

Marco Cova et al 19], propose a mechanism to the anomaly-based detection of attacks against web applications. Swaddler analyzes the internal state of a web application and finds the relationships between the application's critical execution points and the application's internal state. By doing this, he is able to identify attacks that attempt to bring an application in an inconsistent, anomalous state, such as violations of the intended workflow of a web application. The main disadvantage is, as the number of executed basic blocks increases, the overhead grows linearly due to instrumentation and detection overhead associated with each basic block in the program.

Xiang Fu et al [1], propose the design of a static analysis framework (called SAFELI) for identifying SIA (SQL Injection attacks) vulnerabilities at compile time which statically monitors the MSIL (Microsoft Symbolic intermediate language) byte code of an ASP.NET Web application, using symbolic execution. SAFELI can analyze the source code information and will be able to identify very delicate vulnerabilities that can't be discovered by black-box vulnerability scanners. The disadvantage of this work is that this approach can discover the SQL injection attacks only on Microsoft based product.

Livshits and Lam [21], proposed another static analysis approach for finding the SQL injection using vulnerability pattern approach. Vulnerability patterns are described here in this approach. The main issues of this method, is that it can't detect the SQL injection attacks patterns that are not known beforehand.

William G.J. Halfond and Alessandro Orso [2], proposed a mechanism to prevent SQL injection at run time. AMNESIA uses a model based approach to detect illegal queries before it sends for execution to database. In its dynamic method, the technique uses run time monitoring method to inspect each and every query which is passed to its techniques. Here the AMNESIA requires the modification of the web application's source code. They also describes the mechanism to keep track of the positive taints and negative taints which had outlined a new automated technique for preventing SQLIAs based on the novel concept of positive tainting and on flexible syntax-aware evaluation. It can check the SQL statement with these taints and will generate the alarm, if it finds any problems. The advantage of this mechanism is it imposes a low execution overhead, and it doesn't require any modification of the run time system even at application level.

Buehrer et al [12], proposed a static mechanism that filters the SQL Injection. By comparing the parse tree of a SQL statement before and after input, the SQL statements allow the SQL statements to execute only when the parse trees match. Their study on using one real world web application and their application of SQLGUARD solution to each application has stopped all of the SQLIAs without generating any false positive results, and their solution required the developer to rewrite all of their SQL code to use their custom libraries

Along with them, many authors have discussed much number of various techniques to prevent SQL Injection attacks through many ways in static and dynamic analysis and also in DBMS auditing methods. But all these methods reported to have a various pros and cons of its own proposal. In this paper, a new attempt has been proposed and worked out against SQL Injection attacks.

III. OUR APPROACH

Our approach against SQLIAs is based on Declarative method approach that easily addresses the security problems related to input validation. This approach describes two modules which are used to detect the security issues. Efficacy module has got the statement from the web application which includes both the Hirschberg Algorithm to analyze the Statement into the set of characters, and Schatten algorithm to compare the each character with respect to the Desenlace module. After the each character in the statement is scrutinized, if it finds any suspicious activity like finding the characters that are given in different formats, it acts as an active agent to stop the transaction and audit the attacks.

Desenlace module includes the Meta strings library which comprises the predefined keywords and is updated with new type of information in terms of the font details including the font format, font color, and font resolution, font size for each and every character. If both Efficacy module and Quest module has satisfied, it provides the complete transaction. The following figure 1 clearly portrays the architecture of the system to prevent the SQL Injection attacks using this new approach. The following section outlines each module's work in detail.

F*igure 1: Font Level tainting approach for Preventing SALIAs*

A. Desenlace Module:

Desenlace module includes the Meta strings library which comprises the predefined keywords and is updated with new type of information in terms of the font details including the font format, font color, and font resolution, font size for each and every character.

B. Efficacy Module:

In Efficacy module, it gets an input from the web application and it compares the statement with the Meta strings library included in the Desenlace Module, if founds any error message it attempts to block the query. It uses the Hirschberg algorithm to analyze the statement into set of characters. It uses Schatten algorithm for comparison of each character with Meta Strings library and prevent SQL Injection attacks if found. The time complexity of this algorithm is O (nm) and space complexity is O (min (nm)).

Schatten Algorithm:

Begin

1. Consider a statement 'Q_i' from the Web application.
2. Consider the character 'C_i' of 'Q_i'.
3. Consider a statement 'Q_j' from the Meta Strings Library.
4. Consider the character 'C_j' of 'Q_j'.
5. For i=j values, compare C_i (Qi) with C_j (Q_j).
6. If C_i (Q_i)==C_j (Q_j).
7. Then execute the C_i.
8. End-if
9. Increment i, j values.
10. Go to step-5
11. If $C_i(Q_i)$!=$C_j(Q_j)$.
12. Then block the Statement
13. Alert the Generation
14. Report the Result
15. End-if

End

C. SQL Injection code

Select * from table where username='john' and the pwd=1000;

The algorithm describes the way how we follow the procedure for preventing the SQL Injection Attacks.

Let, the generated query Statement be = 'Q_i',

The Statement in the Updated Meta Strings Library= 'Q_j'.

The alphabet of the generated query 'Q_i' = 'C_i',

The alphabet of the String 'Q_i' in Meta String Library='C_i'.

Let i & j be the position values of both the generated and actual statement.

The Meta String Library is now to be updated including the Font details of a specific character, C of a statement Q. Let us first consider the Statement 'Q_i' from the Web application, and we choose a character 'C_i' of 'Q_i'. Now 'C_i' is to be compared with the 'C_j' of the actual statement 'Q_j' included in the Meta Strings Library, in terms of their Font Details. If both the Characters are matched, i.e. $C_i(Q_i) == C_j(Q_j)$, then the character is considered as secure and was sent to the database server for the further transactions. This process repeats till the total statement is accepted.

If at any position, when C_i (Q_i)! $=C_j$ (Q_j), then the character is to be considered in a way that it is in different font and hence the total statement should be blocked. Further the Details of the Prevention are to be reported. Comparing to Hirschberg algorithm, this approach is advanced as it considers the font formats along with the strings for checking; where as Hirschberg principle is unable to check the font details initially.

To be efficient with this approach, all the web pages that are developed along the Internet World have to consider a font format universally; So that it can reduce the chance of un-authorized attacks using malicious codes. In the real world, different font formats are included in developing a web page. It is user friendly to the Developers. But it may be also an alternate way to attack the database of the server. So by using this approach, web page is dynamically developed.

D. Dbms Quest

DBMS Quest enables DBA's to position the usage of database resources and authority [11]. When Quest is enabled, the DBMS will produce a trail checking of database operations, which in turn the each checked database operation produces a trail of checking the information. It includes the details on what database object was impacted, who & when performed the operation.

Depending on the level of Quests supported by the DBMS, the data that was actually changed was recorded. But it has some limited functionality to predict the attacks. It is very useful to find that what type of operation has been made on prevention of attacks. For example End user is a customer that he can log on to the bank and he can see his personal or his academic record of Transactions and money details. Only option given to the users is to check their information. i.e. (Select operation) Instead of select operation, any deletion or update operation is made; attackers could login in to the system and to do some malicious actions, like transforming the money to their accounts.

So this Quest method try to block not only SQL Injection attacks and also some other attacks [18]. The restriction of this DBMS Quest method is to prevent the attackers view some other records, that select operation has been made. It never generates any alarms. Signature based method itself work effectively against SQL Injection attacks well and also this DBMS Quest also provides an support to this method to work effectively against SQL Injection attacks. The Integrated system is under development and the partial results shows encouraging output.

IV. CONCLUSION

This paper presented a highly automated approach for protecting Web applications from SQLIAs. Our approach consists of, Updating of Meta Strings library, Using Schatten algorithm to compare the given statement with updated Meta strings library, find out the SQL Injection attacks. Using DBMS Quest methods to find out the transactions and reports the generation in case of SQL injection attacks.

Schatten algorithm is used to detect the SQL Injection attacks in order to reduce the time and space complexity and it provides the complete execution after analyzing the DBMS Quest. Our approach also provides advantages over the many existing techniques environments, reduces the Time and Space complexities. Moreover, it requires no modification of the runtime system as it is defined at the application level, and hence imposes a low execution overhead.

REFERENCES

[1] R.Ezumalai, G.Aghila's "*Combinatorial approach for preventing SQL Injection attacks*", 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6-7, March 2009.

[2] William G.J. Halfond, Alessandro Orso, PanagiotisManolios, "*WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation*", IEEE Transaction of Software Engineering Vol 34, Nol, January/February 2008.

[3] Konstantinos Kemalis and Theodoros Tzouramanis, "*Specification based approach on SQL Injection detection*", ACM, 2008.

[4] Stephen Thomas and Laurie Williams "*Using Automated Fix Generation to Secure SQL Statements*", International workshop on Software Engineering and secure system ", IEEE, 2006.

[5] V. Benjamin Livshits and Monica S. Lam, "*Finding Security Vulnerabilities in Java Applications with Static Analysis*", ACM, 2005.

[6] Z. Su and G. Wassermann, "*The Essence of Command Injection Attacks in Web Applications*", 33rd ACM SIGPLAN, SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, 2006, pp. 372-382.

[7] Sruthi Bandhakavi, "*CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations*", ACM, 2007."*Combinatorial Approach for Preventing SQL Injection Attacks"*

[8] Ashish kamra, Elisa Bertino, Guy Lebanon, "*Mechanisms for database intrusion detection and response*", Data security & privacy, Pages 31-36, ACM, 2008.

[9] 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 Proc. Of the 13th Intl. World Wide Web Conference (WWW 04), pages 40-52, May 2004.

[10] David Geer, "*Malicious Bots Threaten Network Security*", IEEE,

Oct 8, 2008.

[11] Xiang Fu, Xin Lu, Boris Peltsverger, Shijun Chen, "*A Static Analysis Framework for Detecting SQL Injection Vulnerabilities*", IEEE Transaction of computer software and application conference, 2007.

[12] G.T. Buehrer, B.W.Weide and P.A.G. Sivilotti, "*Using Parse tree validation to prevent SQL Injection attacks*", In proc. Of the 5thInternational Workshop on Software Engineering and Middleware (SEM'056), Pages 106-113, Sep. 2005.

[13] W.G. J. Halfond and A. Orso, "*Combining Static Analysis and Run time monitoring to counter SQL Injection attacks*", 3rd International workshop on Dynamic Analysis, St. Louis, Missouri, 2005, pp.1.

[14] Christina Yip Chung, "*DEMIDS: A Misuse Detection System for Database Systems*", Integrity and internal control information systems, Pages: 159 - 178, ACM, 2008.

[15] N. Jovanovic, C. Kruegel, and E. Kirda, "*Pixy: A Static Analysis tool for detecting web application vulnerability*", in 2006 IEEE Symposium on Security and Privacy, May 2006.

[16] O. Maor and A. Shulman, "*SQL Injection Signature Evasion*", White paper,Imperva, Apr 2004.

[17] Nguyen-tuong, S. Guarnieri, D. Greene, J.Shirley, and D. Evans, "*Automatically hardening web applications use Precise Tainting*", In Twentieth IFIP Intl, Information security conference (SEC 2005), May 2005.

[18] R. McClure and I. Kruger, "*SQL DOM: Compile Time Checking of Dynamic SQL Statements*", Inproc of the 27th Int. Conference on Software engineering (ICSE 05), pages 88-96, May 2005.

[19] Xin Jin, Sylvia Losborn, "*Architecture for data collection in database intrusion detection system*", Secure data management, Springer link, 2007.

[20] Mehdi Kiani, Andrew Clark, George Mohay, "*Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks*", Third International Conference on Availability, Reliability and Security - Volume 00, , Issue, 4-7 Page(s):47-55, IEEE, March 2008. [21] V.B. Livshits and M.S. Lam, "Finding Security vulnerability in java Applications with static analysis", In proceedings of the 14th Use nix Security Symposium, Aug 2005.

[21] Marco Cova, Davide Balzarotti, Viktoria Felmetsger, and Giovanni vigna, "*Swaddler: An approach for the anamoly based character distribution models in the detection of SQL Injection attacks*", Recent Advances in Intrusion Detection System, Pages 63-86, Springer link, 2007.