IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

Detection and Prevention of SQL Injection in a Web Application Using Secure Object Relational Mapping Approach

Siddhesh Bhagat[1], Dr. R. R Sedamkar[2], Prachi Janrao[3]

Student, Computer Department, Thakur College of Engineering and Technology, Mumbai, India[1]

Professor, Computer Department, Thakur College of Engineering and Technology, Mumbai, India[2]

Assistant Professor, Computer Department, Thakur College of Engineering and Technology, Mumbai, India[3]

Abstract: Many modern web based systems are presently following 3-tier architecture for implementation of enterprise application. However rare use of modern frameworks and standards in this kind of applications, these applications are more vulnerable to attack that can breach and steal confidential information stored in database. One of the attack known as SQL injection is a very serious and flawless way to retrieve data without leaving any traces behind process. This paper addresses a solution for this kind of serious problem in a novel way which not only provides efficient solution but acquire a modern coding standard which developers follow. This new approach called as ORM technique. ORM is a Object Relational Mapping where we maps the table architecture with corresponding Object and use those objects to retrieve data instead of getting data from database directly . Hence it creates a indirect barrier from firing SQL query which helps us to prevent our important information from direct access. As it also follows standard of coding this ORM Methodology satisfies desired criteria of highly cohesive with loose coupling while coding.

Keywords: ORM, Hibernate, SQLIA, LDAP, FCD, SSC, T-SQL.

I. INTRODUCTION

WEB applications are those applications which we access over internet with the help of any of the web compliant browser like Internet explorer, Google chrome, Opera mini and many more. They are invariably available due to the handiness, suppleness, accessibility, and interoperability that they provide. Unfortunately, over a network nothing is secure. Hence our web applications are more susceptible to many kinds of security threats. SQL Injection Attacks (SQLIAs) are one of the serious such threats [1].

SQLIAs have become increasingly frequent and creates very solemn security risks because they can give attackers unlimited access to the databases that triggers Web applications. Web applications interface with databases that have information such as employee names, preferences, credit card numbers, purchase orders, and so on. Web applications construct SQL queries to access these databases based, in part, on user-provided input. The motive is that Web applications will limit the kinds of queries that can be generated to a safe subset of all potential queries, regardless of what type of input users provide. However, inadequate input validation can enable attackers to get complete access to such databases. One way in which this happens is that attackers can present input strings that contain specially builds a query by using these strings and sends the query to its underlying database, the attacker’s embedded commands are driven by the database and the attack happens. The results of these attacks are often devastating and can range from leaking of sensitive data (for example, employee data) to the destruction of database contents. Researchers have defined a wide range of alternative techniques to address SQLIAs, but many of these solutions have certain limitations that affect their effectiveness and practicality. For example, one known type of solutions is based on defensive coding application, which has been less than successful for three main reasons. First, it is difficult to implement and enforce a meticulous defensive coding discipline. Second, many solutions based on defensive coding deal with only a subset of the possible attacks.

Third, legacy software poses a particularly tricky problem because of the cost and complexity of modifying existing code so that it is compliant with defensive coding practices. In this method, we propose a new highly automated approach for dynamic detection and prevention of SQLIAs. Intuitively, our approach works by recognize “trusted” strings in an application and allowing only these trusted strings to be used to create the explanation relevant parts of a SQL query such as keywords or operators. The general mechanism that we use to implement this technique is based on dynamic tainting, which marks and tracks certain data in a program at runtime. The kind of dynamic tainting that we use gives this method several important advantages over methods based on other tools. Many methods depend on complex static analyses in order to find potential vulnerabilities in the code. These types of conservative static analyses can produce big rates of false positives and can have scalability issues when size of application increased. To make web application more secured and we will use ORM hibernate tool.

II. LITERATURE SURVEY

In this reference [2] author has presented a novel fully automated technique, TransSQL, for preventing SQL injection attacks. The technique is based on the intuition that injection codes implicitly perform a different meaning from general queries. The author presented an elaborate environment based on LDAP for distinguishing legitimate and malicious queries. To complete this task, TransSQL is consisted with preprocessing step and runtime step. In the preprocessing step, the technique uses an existing SQL command to extract from SQL database a file which contains whole information of SQL database. According to the sqldump file, TransSQL generates a duplicated database in LDAP form. In runtime step, TransSQL monitors connection between web applications and SQL databases. Every query would be translated into a LDAP-equivalent query, and then we defined some conditions to identify malicious queries.[2]

In this method, to make SQL injection attack, an attacker should necessary use a space, double quotes and double dashes in his input. The method to detect one of the above symbols has been discussed. This method consists of tokenizing original query and a query with injection and after if it is found that extra symbols used in user input, so the injection is detected. This approach consists of tokenizing the original query and the query with sql injection attack and after tokens are generated they constitute arrays' elements. By comparing lengths of the output arrays from the two queries injection attack can be detected. The work presented in this method has been implemented using java codes [4].

This method explains that many web applications employ a middleware technology designed to request information from a relational database management system in SQL speech. SQL injection is a one of the techniques hackers enlist to attack underlying databases. These attacks reshape the SQL queries, thus altering the behavior of the program for the use of the hacker. Several solutions exist to prevent SQLIAs at the application layer, but no fix solution other than using parameters while coding exist to protect stored procedures in the database layer against SQLIAs. In this paper, it present a fully automated technique for detecting, preventing and result of SQLIA attacks in stored procedures. The technique explains the intended SQL query behavior in an application in the form of an SQL-graph, as a one-time offline steps using static analysis of the stored procedure present in the source code [6].

In this technique they describe two character distribution models; the FCD and SCC models. They have shown that the SCC model is good at detecting SQL injection attacks in general, as well as being more accurate than the FCD model overall. it also assess the models' effectiveness at detecting the UNION and Tautology classes of SQL injection attacks. While the SCC model is better to the FCD model at detecting both of these types of SQL injection attacks, they have also explained that character distribution models are much better for detecting UNION attacks than this Tautology attacks. This result was not completely unexpected due to the precise nature of Tautology attacks. It also showed that the SCC model is effective at detecting muddle attacks. The approach handled by parsing the query part of HTTP requests and generates view for each file. It does not required access to the source code and modification of existing software modules. Additionally, they explained that this proposed approach does not need user interaction or the introduction of user defined data types to reduce false alerts [7].

In this method, they proposed a structure for development of runtime monitors used to do post-deployment monitoring of the software to detect and prevent tautology based SQLIAs. Thus using this proposed framework this ensures that the quality and security of software is achieved not only through its pre-deployment phase also during its post-deployment phase and any possible misuse of vulnerability in the software by an outsider attacker is found and prevented. It further intends to automate the entire process of using the proposed structure to develop the runtime monitors and also extend this structure to detect and prevent the other types of attacks [9].

III.PROPOSED METHODOLOGY

It is a two-step process which is explained with the help of work flow mentioned in the below figures.

First is a simple login application which is developed by traditional way of implementation using 3-tier architecture of client server database model where SQL injection is detected with the help of test cases and it is observed that most of the confidential data can be fetched by breaching authentication process.

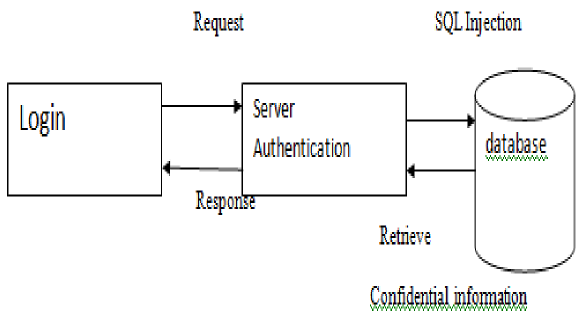

Fig. 1 Normal architecture with SQL injection possibility

Second is a secured way of architecture where SQL Preventer plays a significant role by not allowing direct access and filter all queries which can breach security and cause SQL injection.

This Preventer is consisting of an Object relational Mapping tool called as ORM Hibernate tool.

It is a ORM tool used to map the objects in java into its corresponding table automatically without righting a single native SQL Query. It helps in making your system independent of database vendors. It reduces the coding by removing repetitive codes called as boilerplate codes that are used for database connection every time whenever we require fetching or inserting data inside database. Its configuration file helps us to create tables for required Objects.

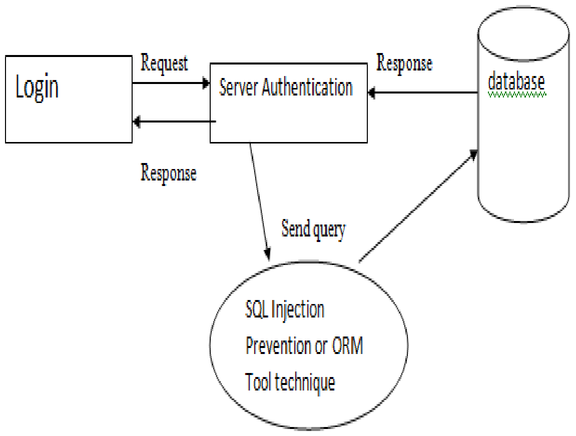

Fig. 2 Login using SQL preventer tool

A. **Hibernate**
Create relationship among Objects automatically. And most importantly it provides facility of caching which saves frequently used data items also

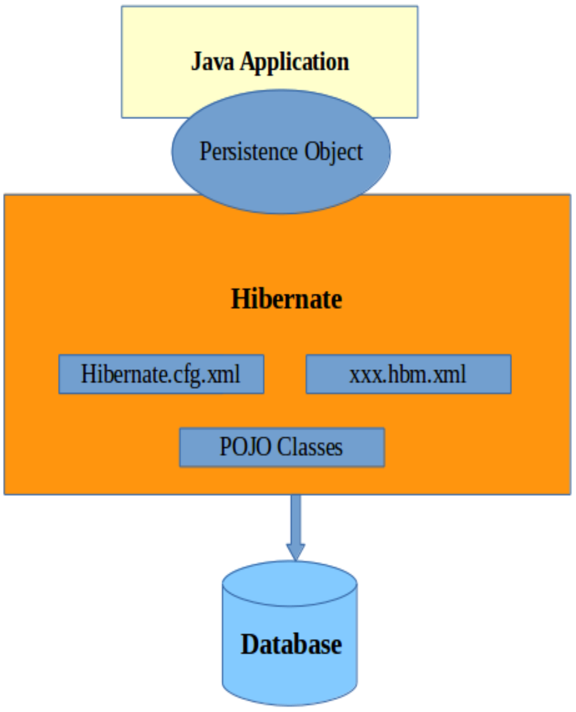

Fig. 3 Hibernate architecture

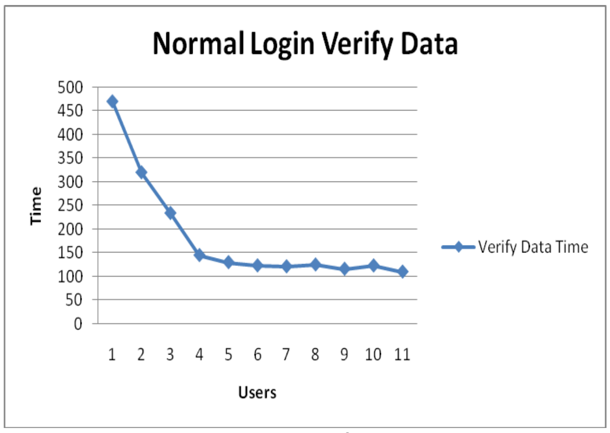

Fig. 4 Normal login verify data line graph

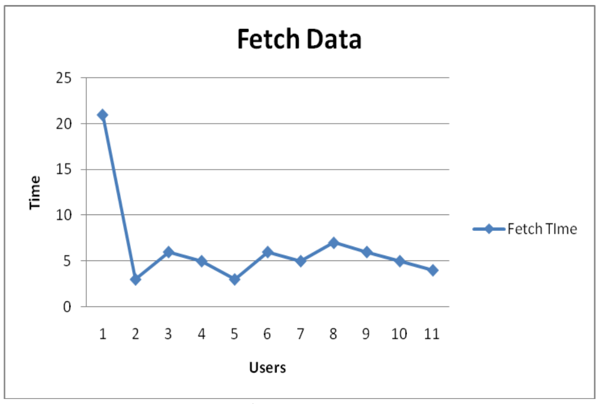

Fig. 5 Fetch data for normal login line graph

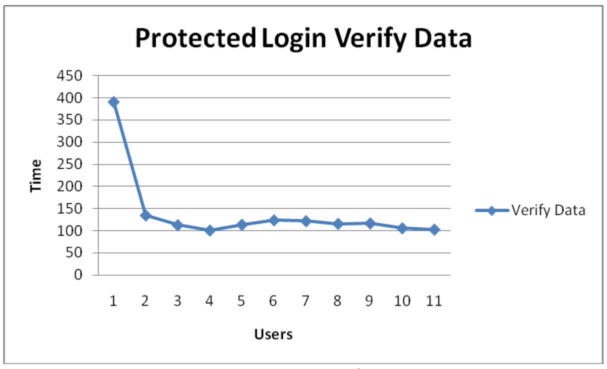

Fig. 6: Protected login verify data line graph

IV. RESULT ANALYSIS

A. Performance Analysis
Here first Normal Login timings and data fetch timing readings are taken and its impact on increasing number of users is observed and then next with an overhead of our SQL preventer it is again check to see performance changes.

Here it has been observed that after introducing SQL preventer tool it's a reasonable amount of delay occurs as compare to normal login which is acceptable for standard development. And in any case it is preventing data by not allowing SQL injection based data and makes it more efficient by storing or caching frequently used data.

V. CONCLUSION

This paper presented a framework based approach which confirms with coding standards followed by modern developers for protecting web application from SQLIA. Our approach consists of indirect access of database to prevent SQLIAs by using any of the ORM tools. That treats tables as synonymous Objects and provide alternative of native SQL queries to protect confidential data. It has been tested with this approach Hibernate ORM tool which is an Object Relational Mapping tools which support HQL which automatically converts persistent objects into a table required for any database.

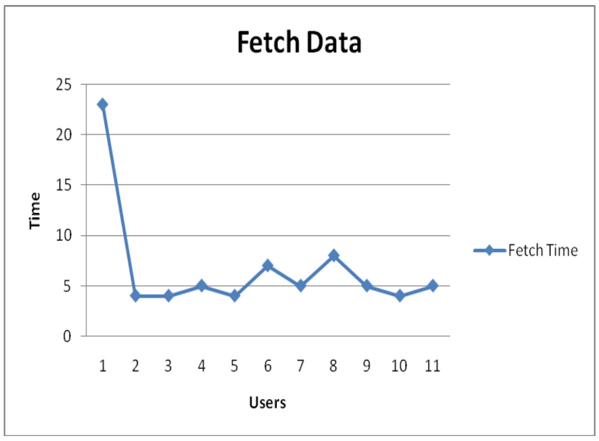

Fig. 7 Fetch data for preventive line graph

Our preliminary test shows that there is a marginal overhead in timing consumes by SQL Preventer which first intercepts any direct request to fetch data from database via its own way which filters SQL injections and it is acceptable for any kind of real time Enterprise application development.

VI. FUTURE SCOPE

Through hibernate is very prevalent now a days for using big projects however our testing was confined with medium sized web applications. In future one can test this approach with big banking applications, MIS Systems many more which are real enterprise application consisting from pre sign on to post sign on Activity with so many complex modules which requires high level of special security designing.

REFERENCES

[1] William G.J. Halfond, Alessandro Orso, Member, IEEE Computer Society, and Panagiotis Manolios," WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", Member, IEEE Computer Society IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 1, JANUARY/FEBRUARY 2008.

[2] Kai-Xiang Zhang, Chia-Jun Lin Engineering, Shih-Jen Chen, Inst Yanling, Hao-Lun Huang, Fu-Hau Hsu Computer Science & Info. Engineering TransSQL:" A Translation and Validation-based Solution for SQL-Injection Attacks", 2011 First International Conference on Robot, Vision and Signal Processing

[3] C. Anley. Advanced SQL Injection In SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002.

[4] NTAGWA BIRA Lambert, KANG Song Lin," Use of Query Tokenization to detect and prevent SQL Injection Attacks", IEEE2010.

[5] Nuno Antunes and Marco Vieira. Detecting SQL Injection vulnerabilities in web services. IEEE, 2009.

[6] Ke Wei, M. Muthuprasanna, Suraj Kothari," Preventing SQL Injection Attacks in Stored Procedures", Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06) 1530-0803© 2006 IEEE.

[7] Mehdi Kiani, Andrew Clark and George Mohay," Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks", The Third International Conference on Availability, Reliability and Security

[8] W. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks," Proceeding of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), 2005.

[9] Ramya Dharam and Sajjan G. Shiva," Runtime Monitors for Tautology based SQL Injection Attacks",ICS 2002

[10] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," Proc. Fourth ACM Symp. Operating System Principles, Oct. 1973.

[11] Y. Xie and A. Aiken, "Static Detection of Security Vulnerabilities in Scripting Languages," Proc. 15th Usenix Security Symp., Aug. 2006.

[12] C. Anley, "Advanced SQL Injection In SQL Server Applications," white paper, Next Generation Security Software, 2002.

[13] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities," Proc. IEEE Symp. Security and Privacy, May 2006.