International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 11 – No. 4, September 2016 – www.ijais.org

Preventing SQLIA using ORM Tool with HQL

Siddhesh Bhagat
M.E. Scholar
Computer Engineering Department,
Thakur College of Engineering and Technology, Mumbai

R. R Sedamkar, PhD
Professor
Computer Engineering Department,
Thakur College of Engineering and Technology, Mumbai

Prachi Janrao
Assistant Professor
Computer Engineering Department,
Thakur College of Engineering and Technology, Mumbai

ABSTRACT
Web based systems nowadays follow 3-tier architecture for implementation of enterprise application. But these applications are more prone to security breach and loss of confidential information stored in database. One of the more serious attacks is known as Structured Query Language Injection (SQLI).This attack retrieves data without leaving any trace behind. This paper proposes an efficient solution called Object Relational Mapping technique for such kind of attack in a novel way. ORM maps the table architecture with corresponding Objects and uses those objects to retrieve data instead of getting data directly from database .Therefore it creates a indirect barrier from firing SQL query preventing direct access to database. In addition ORM Methodology satisfies desired criteria of loose coupling while coding.

Keywords
T-SQL, ORM, LDAP SQLIA, FCD, SSC, HQL, MIS, CBS

1. INTRODUCTION
We access many applications called web applications over internet with the help of any of the web compliant browsers like Google chrome ,Safari, Internet explorer etc. These are constantly available due to their convenient accessibility and interoperability. But nothing is secure over a network. Therefore web applications are vulnerable to various security threats. SQL Injection Attacks (SQLIAs) are one of the more serious kind of threats [1].

SQLIAs have become increasingly frequent. They pose very serious security risks because they provide attackers unlimited access to the databases that trigger web applications. Web applications interface with databases containing confidential information such as employee names, preferences, credit card numbers, purchase orders etc. Web applications create SQL queries to access these databases using user-provided input. The belief is that Web applications will limit the kind of queries that can be generated to a safe subset of all potential queries, regardless of kind of input by users. However, incomplete input validation can enable attackers to gain complete access to databases. One way in which this occurs is that attackers insert input strings that contain specially built queries. Using these strings the query is sent to the underlying database where the attacker's embedded commands are now driven by the database. the attack now takes place. The fallout are often devastating and can range from theft of sensitive data (for example, employee data) to the destruction of database contents. Web security experts have proposed a wide range of alternative techniques to counter SQLIAs, but most of these solutions have certain limitations that affect their effectiveness. For example, one of the solutions which is based on defensive coding application has been less than successful for three main reasons. First, it is not easy to implement. Second, it deals with only a subset of the possible attacks. Third, the cost and complexity of modifying existing code so that it is compliant with defensive coding practices is very prohibitive. In this paper, we propose a extremely automated method for dynamic detection and prevention of SQLIA. Its approach works by recognizing "trusted" strings in an application and allowing only these trusted strings to be used to create the explanation for the relevant parts of a SQL query such as keywords or operators. The general mechanism utilizing this technique is based on dynamic tainting which marks and tracks certain data in a program at runtime. The type of dynamic tainting used in this approach has several important advantages over other approaches. Many of these other methods depend on complex static analysis in order to find potential risks in the code. Also they produce big rates of false positives and have scalability issues when size of application increases. Therefore to make web application more secure we recommend ORM hibernate tool.

2. LITERATURE SURVEY
In this method, to make SQLIA, it is essential for an attacker to use a space, double quotes and double dashes in his input. The approach is to detect one of the above symbols has been discussed. This approach is a series 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 uses the process of tokenizing the original query and the query with sql injection attack and after tokens are generated they form arrays' elements. On comparing lengths of the output arrays from the two queries SQLIA can be detected. The work given in this method has been implemented using java codes.[4]

In this author have bestowed a novel fully automated technique, T-SQL, for preventing SQLIA. The method is based on the intuition that injection codes implicitly perform a various meaning from general queries. They presented an detailed environment based on LDAP for differentiate legitimate and malicious queries. To sheer this task T-SQL is consisted with preprocessing step and runtime step. In the preprocessing step, the method uses an existing SQL command to passage from SQL database a file which contains whole information of SQL database. According to the sqldump file, T-SQL generates a duplicated database in LDAP form. In runtime step, T-SQL supervisors connection between web applications and SQL databases. Every query would be iterated into a LDAP-equivalent query, and then they defined some states to identify malicious queries.[2]

In this approach they depict two character distribution models, the FCD and SCC models. They have demonstrated that the SCC model is good at detecting SQL injection attacks in

general, as well as being more accurate than the FCD model in overall comparison. It also evaluate the models' effectiveness at detecting the UNION and Tautology classes of SQLIA. While the SCC model is better than FCD model at detecting both of these types of SQLIA They have also explained that character distribution models are much better for detecting UNION attacks than this Tautology attacks. This conclusion was not completely unexpected due to the precise nature of Tautology attacks. It also showed that the SCC model is best for detecting muddle attacks. The method handled by parsing the query part of HTTP requests and generates view for each output file. It does not required access to the source code and modification of existing software modules. In addition to that they explain the proposed approach does not need user interaction or the introduction of user defined data types to reduce false alerts.[3]

This method states that many web applications employ a middle-ware technology designed to request information from a relational database management system in SQL. SQLI 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 method, they proposed a structure for development of run-time monitors used to do post-deployment observation of the software to detect and prevent tautology based SQLIAs. Thus using this planned model this ensure 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 intend to automate the entire process of using the proposed structure to develop the run-time monitors and also extend this structure to detect and prevent the other types of attacks.[11]

3. PROPOSED METHODOLOGY

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

First is an application without preventing SQLIA 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 many of the confidential data can be fetched by breaking authentication process.

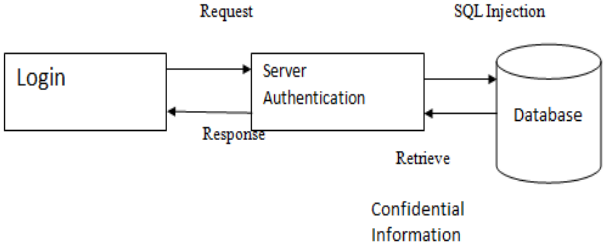

Figure 1: Without Prevention of SQLIA

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

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

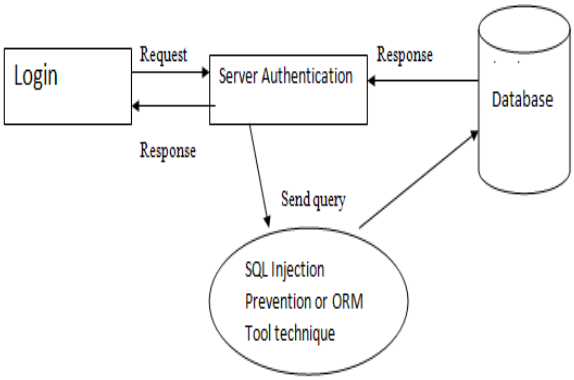

Figure 2: Preventing access to Database using SQLIA

3.1 Hibernate

It is a ORM tool used to map the objects in java into its corresponding table automatically without righting a single endemic SQL Query. It helps in making your system independent of database merchants. It reduces the coding by removing iterative codes called as boilerplate codes that are used for database connection every time whenever it require fetching or inserting data inside database. Its configuration file helps us to create tables for mapping the required Objects. Creating relationship among Objects automatically and most importantly it provides facility of caching data, which saves frequently used data items also in database.

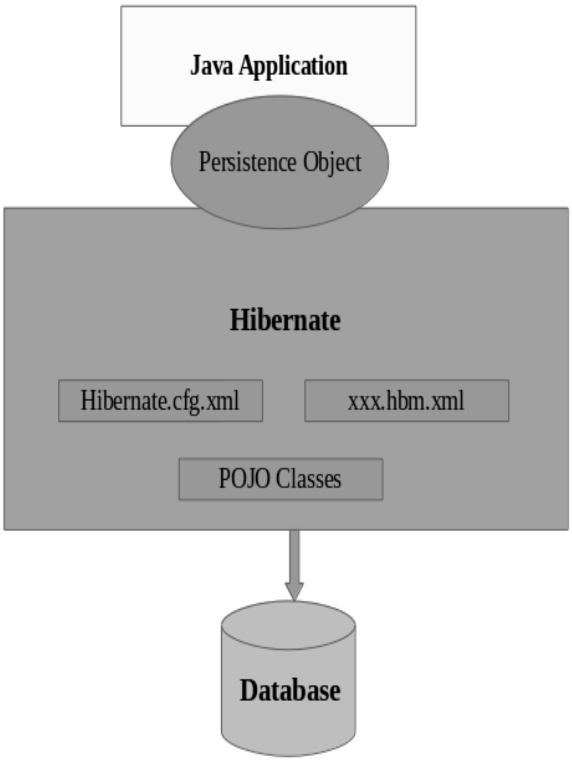

Figure 3: Hibernate Block Diagram

4. RESULT ANALYSIS

4.1 Performance Analysis

Here first without prevention of SQLIA with 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 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 without prevention of SQLIA 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.

Table I: Without Preventing SQLIA

Without Secured Access to database Timing in Miliseconds	Verify Data	Insert Data	Fetch Data
1st	470	25	21
2nd	320	23	3
3rd	234	27	6
4th	145	21	5
5th	130	29	3
6th	123	27	6
7th	121	23	5
8th	125	21	7
9th	116	23	6
10th	123	21	5
11th	110	22	4

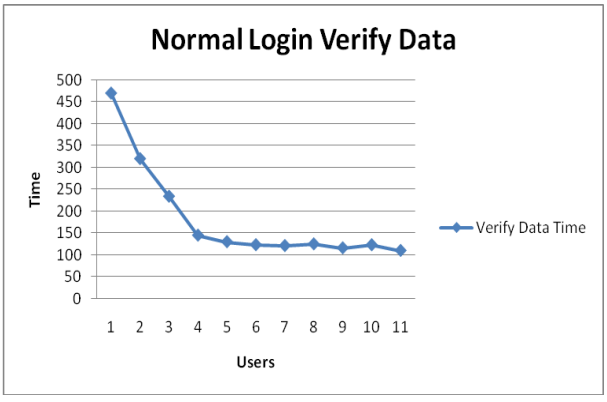

Figure 4: Without Preventing SQLIA

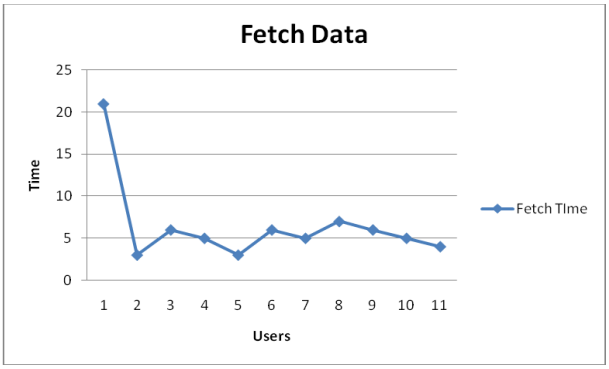

Figure 5: Retrieving Data for without preventing SQLIA

Table 2: Secured Access to Database using ORM Tool

Protected Query Miliseconds	Verify Data	Fail try for SQL Injection	Insert Data	Fetch Data
1st	390	11	296	23
2nd	134	9	25	4
3rd	112	8	23	4
4th	100	7	26	5
5th	113	8	27	4
6th	123	12	26	7
7th	121	10	24	5
8th	115	9	27	8
9th	116	8	23	5
10th	105	7	25	4
11th	102	9	23	5

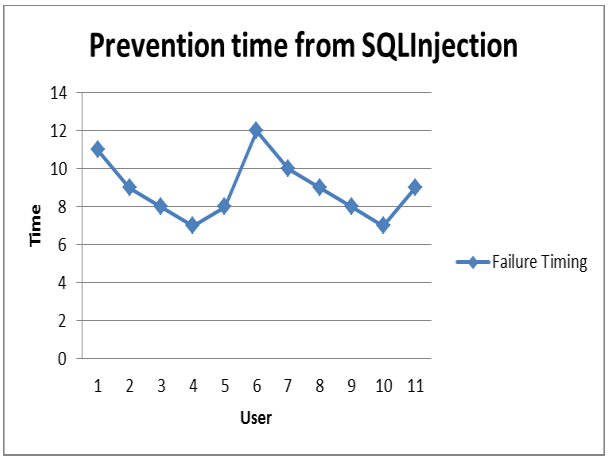

Figure 7: SQLIA by time prevention

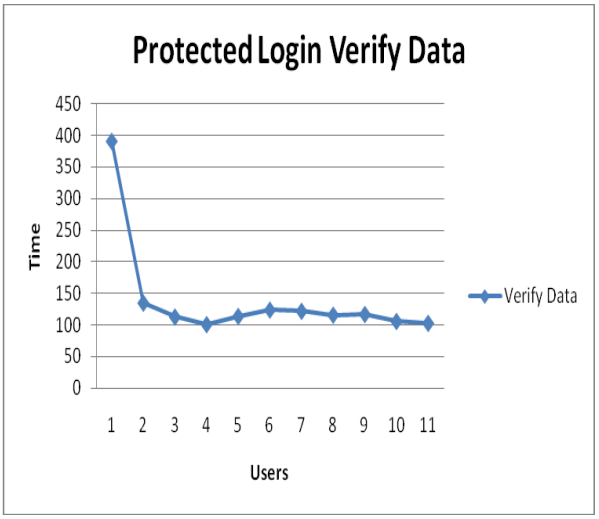

Figure 6: Secured Access to Database

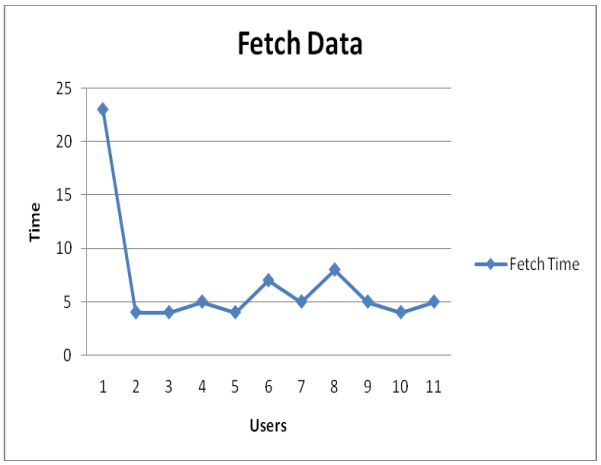

Figure 8: Retrieving data for Secured Access to Database

5. CONCLUSION

This paper proposed a framework based solution which conforms with coding standards adopted by current developers for protecting web application from SQLIA. Its Solution consists of indirect access of database to prevent SQLIA by using any of the ORM tools. This treats tables as synonymous Objects and provides alternatives of native SQL queries to protect sensitive data. Hibernate Object Relational Mapping(ORM) tool supports HQL by automatically converting persistent objects into a tables required for any database. Tests results have shown that there is a marginal overhead in timing consumed by SQL Preventer which first intercepts any direct request to fetch data from database which filters SQLIA. This is acceptable for any kind of real time Enterprise application development.

6. FUTURE SCOPE

Nowadays hibernate is prevalent in big projects. Testing was confined to medium sized web applications. In future one can test this method with big projects like core banking applications (CBS), MIS Systems and other real enterprise applications requiring high level of web security designing.

7. REFERENCES

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

[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] 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

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

[5] 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.

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

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

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

[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] J. C. Anderson & D. W. Gerbing. "Structural equation modeling in practice: A review and recommended two-step approach". Psychological Bulletin, vol.103, no.3, pp.411-423. 1988.