A Proposed Architecture for Query Anomaly Detection and Prevention against SQL Injection Attacks

T.K. George
Research Scholar
Cochin University of Science and Technology

Poulose Jacob, PhD
Professor
Cochin University of Science and Technology

ABSTRACT
SQL injection is a predominant type of attack which targets web applications and databases. SQL injection bypasses the authentication logic and breaks the confidentiality of the database or manipulates the database. It helps the attacker to obtain unauthorized access into the back end database. Vulnerability exists within a web application when it does not provide a proper validation system for the data entered by the user in the input field. Vulnerability scanners aid in checking vulnerabilities embedded in a web application and has the potential to test invalid forms of input query. However, the limitation lies in the reduction of system availability due to denial of service, especially in case of false positives. In this paper, an approach which focuses on query template based detection of SQL injection attack and reconstruction of queries is proposed. Thus the proposed architecture can mitigate the denial of service and increase the availability by potentially reconstructing malicious queries.

Keywords
SQL Injection, Authentication, Vulnerability, Validation, Malicious, Reconstruct.

1. INTRODUCTION
An SQL Injection Attack (SQLIA) increasingly targets online applications. The traditional security measures adopted by organizations are not sufficient enough to deal with new vulnerabilities and their attack spectrum. Although numerous protection strategies against SQLIA have been developed and implemented with the support of security tools and vulnerability scanners, there are frequent attacks on database servers and compromise of critical applications within business organizations. SQLIAs have been described as one of the most serious security threats to web applications [1].

In SQL Injection Attacks [2], the crafted codes are directed at the database server and these codes compromise critical information. A crafted injection through a less secure online application can cause major compromises in terms of data loss and information disclosure on critical database servers. Most of these attacks are targeted on web forms and login entry screens of an application. However, there are limitations to mitigation strategies or tools developed against SQLIA and implemented strategies become obsolete due to the inconsistencies existing in those insecure applications. The availability and effective utilization of appropriate security tools is still an issue that has to be dwelled upon by many of the application users.

Defensive programming and effective input validation techniques can handle some of the vulnerabilities, however new exploits can overcome all the security barriers installed by programmers. SQL Injection prevails as one of the top ten vulnerabilities and threats to online businesses targeting backend databases. It is observed that SQL injection appears only in a small proportion of applications and are yet making huge impact on business organizations through data theft or compromising database servers. Figure1 shows the percentage of various vulnerability classes as reported in OWASP [1].

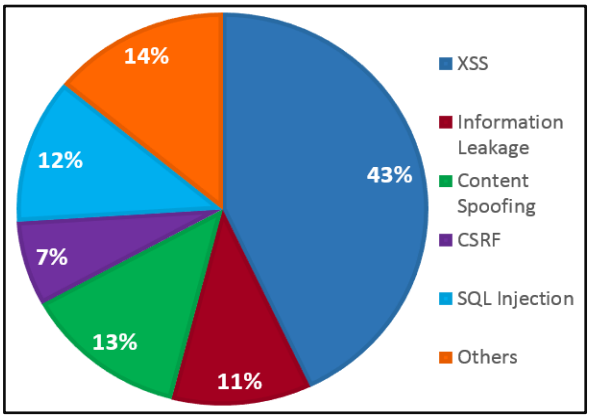

Fig 1: Top 10 Vulnerability class listed

If web applications are heavily dependent on databases and interactions through web forms, then user input validations will be a major concern. Section 2 includes the motivation of the research and related work is discussed in Section 3. Section 4 describes the proposed architecture of SQLIA Detection and Reconstruction of Queries and its advantages. The implementation of the system is given in Section 5 followed by the conclusion and future scope in Section 6.

2. MOTIVATION
Though there are in-depth researches on SQL Injection Attacks and protection strategies, most of the research studies are focused on ensuring database access to legitimate users only. They do not suggest prevention techniques against SQL Injection Attacks carried out by legitimate users. There are some approaches that do not require modification or reconstruction of queries generated by user input on web applications in order to detect SQLIA. Methods based on pattern matching and behavior modeling make use of manually or semi-automated constructed patterns. There are a number of reports that alert the false positives and DoS (Denial of Service), which contradict the availability aspects of security property [3].

3. RELATED WORK
SQLRand [4][5] suggests a randomized SQL query language to detect and abort queries which contain injected code. This is done by modifying a query by appending a random number with it followed by placing a proxy server between the client's web server and application server. The function of this proxy server would be to receive the request and pass it on to the database server. If the request is embedded with SQLIA, it will not recognize the query and will end with a rejection.

CANID [6] proposes a test set creation by extracting query structures from every SQL query location in web application source code for avoiding SQLIA and also suggests web app code changes. The verification method concludes with the issuing of the actual query if the test set matches.

AMNESIA [7] uses a combination of static analysis and dynamic analysis to analyze web application codes and also to monitor dynamically generated queries. This is followed by the building of a query model with all possible queries identified by the hotspot. The runtime monitoring mechanism will reject or report the queries that violate the model.

DoubleGuard [8] is a java based application. In order to avoid object duplication, lots of statistical analysis and structure mapping is required.

POSITIVE TAINTING [9] suggests identifying trusted data by considering trust marked strings and performing syntax aware evaluation. Tracking and taint marking should be accurate with a right level of precision.

SQL DOM [10] creates class tables and methods for possible operation. Database structure mapping will be done manually to avoid object duplication.

4. SQL DETECTION AND RECONSTRUCTION OF QUERIES

4.1 Proposed Architecture

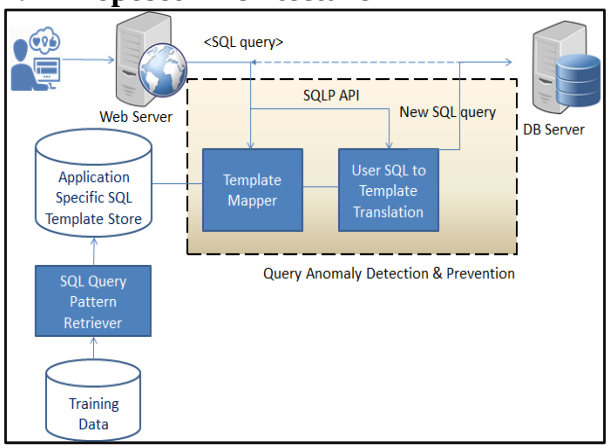

Fig 2: Proposed Architecture

Figure 2 shows the proposed architecture. The main components of the proposed architecture include a SQL query pattern retriever, a template mapper and a User SQL to Template Translation for the reconstruction of queries [13]. SQL query pattern retriever extracts a set of features by analyzing training data. Based on this extracted feature set, the system generates the query template without the help of a developer. The generated SQL templates (application specific) are stored in the system". Each query template must also be stored according to the specifications using JSON format [14][15]. Template mapper receives the input query and maps it with the query stored in the template store. If the injected query is detected, it will pass through the template translation module for reconstruction of the query and avoids Denial of Service.

The proposed research is divided into three phases. The first phase is focused on detecting SQLIA in the authentication of web pages. Most of the SQLIA happens while authenticating a web page. The web developer also has to provide a standard query with which every query is matched for anomaly. The system places the requirement that each incoming query should match with the developer supplied template.

To begin with the first phase, a set of (150) SQL injection queries have been chosen. These queries are classified into various sub types based on the structure of the SQL Queries. They are further classified based on the technique used for injecting the query. The proposed SQLIA detection and mitigation approach initially checks every incoming query for SQLIA, and in case an anomaly is detected, the approach reconstructs the query according to the standard query template specifications and forwards the same to the database server.

In the second phase, strategies for detection and mitigation of SQLIA through URL are framed and implemented. Training methods for automatic construction of query templates from training data are also proposed. Similar to Natural Language Processing (NLP) techniques, there will be Structured Query Language Processing (SQLP) methods to process SQL constructs that are defined; and SQLP retrieves the required input from requested SQL and places the values in an expected query template[14],[15]. Only the newly formed query is submitted to the database server.

An SQLIA detection approach based on features is also proposed as part of the work. These features reflect on all standard methods used to inject an anomaly into SQL. This can be compared with other existing SQLIA detection algorithms to analyze its performance. If the existing approach detects SQLIA, then there is an immediate decision of reconstruction of the query. This must be repeated through a scan for SQLIA and verification followed by the transmission of this information to the database server [16].

In the third phase of the approach, the SQLIA is verified with the existing DoS based vulnerability detection systems to find the total number of queries which were successfully reconstructed, number of queries forwarded, number of queries acted upon by the database server, and finally the number of false positives actually processed by the system [17].

4.2 Advantages of the Proposed Architecture

The advantages of the proposed approach are increased system availability by reducing DoS and a strong focus on reconstruction of Injected Queries. This approach does not analyze web application but only checks the query (SQL) before reaching the specific database server. It also works only between the web server and the database server.

The proposed architecture can detect different types of SQLIA's such as Tautology based injection, Statement injection, Union query, Logically incorrect queries, Stored procedures, Piggy backed queries and Alternate encoding queries. It can be expanded for the detection of Inference based on Blind Injection, out of band injection through Server variables (Injection through URL). Moreover, it can be modified further to detect Second Order Injection attacks [11] [12].

5. IMPLEMENTATION OF SQLIA DETECTION AND MITIGATION SYSTEM

The proposed system is developed by using java based application development. It can be implemented as an API or placed as a proxy server which can be placed in between web

server and database server. Following are some of the Sample screens designed for the detection, mitigation and reconstruction of queries.

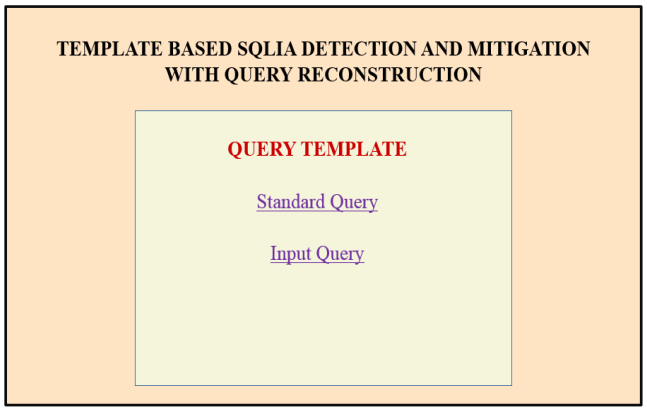

Fig 3: Main Screen for Query Verification

5.1 Sample format of Injected Queries

a. SELECT *FROM administrators WHERE username= ‘ ’ AND password= ‘ ’ OR ‘1’=‘1’;

In the above query, username is set to null and in the password field, there is a single quote for closing the password field and define a tautology (‘1’=’1’) with logical OR.

b. SELECT *FROM administrators

WHERE username= ‘ ‘ AND password= ‘ ‘ OR ‘1’=‘1’; DROP TABLE users;

Here the username is set to null and in the password field, Put a single quote for closing the password field, define a tautology (‘1’=’1) with logical OR. Place a single quote and semi-colon for closing the password field. There is a new SQL query for drop the table from database.

c. SELECT *FROM administrators

WHERE username=’’ UNION ALL SELECT LOAD_FILE (‘/etc/passwd’)--

The above query can be described as In username field there exist a single quote for closing the username field and Use UNION ALL and write a new SQL query to load all the contents from (‘/etc/passwd’) folder. Put double hyphen signs to comment the remaining query. The password of the above query is set to null.

d. SELECT * from admin where username = '' or 1=1--

Above given Query the username field, there exists a single quote for closing the username field. And define a tautology (1=1) with logical OR. Comment the remaining query with double hyphen (--).The password is set to null.

e. SELECT * FROM administrators

WHERE username =’ ’; exec(char(0x73687574646f776e))--AND password=’ ‘;

In the username field of the above query, there is a single quote for closing the username field. There is a semi-colon and write an SQL query for Shutdown the database and put a semi-colon and comment the remaining query using double hyphen (--).The password field is set to null.

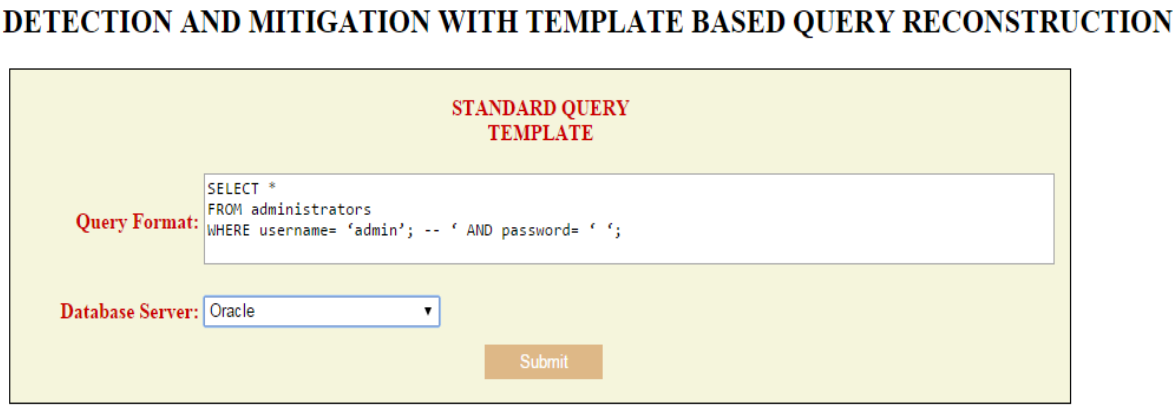

Fig 4: Query Template for Verification

6. CONCLUSION AND FUTURE WORK

Existing approaches mainly focus on DoS in the presence of suspected injected queries, while the proposed approach concentrates on the extraction and reconstruction of a query, which increases the availability of the system. The proposed system offers automatic template construction, SQLIA detection and mitigation algorithms for the authentication of webpages, via URL or through HTML controls. Automatic template extraction strategies can be built for other types of web pages, other than authentication page. Similarly, SQLIA anomalies for different SQL constructs follow varying strategies thus the reconstruction of algorithms can be further expanded to have better availability and performance in complex systems.

7. REFERENCES

[1] Kumar, Pranaw, and R. K. Pateriya. "A Survey on SQL injection attacks, detection and prevention techniques". Computing Communication & Networking Technologies (ICCCNT), 2012 Third International Conference on. IEEE, 2012.

[2] Vanitha, A., and N. Radhika. "Multidimensional Analysis of SQL Injection Attacks in Web Application."

[3] Bosworth, Seymour, and Michel E. Kabay, eds. Computer security handbook. John Wiley & Sons, 2002.Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[4] Boyd, Stephen W., and Angelos D. Keromytis. "SQLrand: Preventing SQL injection attacks." Applied

Cryptography and Network Security. Springer Berlin Heidelberg, 2004.

[5] Avireddy, Srinivas, et al. "Random4: an application specific randomized encryption algorithm to prevent SQL injection." Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on. IEEE, 2012.

[6] Bisht, Prithvi, Parthasarathy Madhusudan, and V. N. Venkatakrishnan. "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks." ACM Transactions on Information and System Security (TISSEC) 13.2 (2010): 14.

[7] Halfond, William GJ, and Alessandro Orso. "AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005.

[8] Ezumalai, R., and G. Aghila. "Combinatorial approach for preventing SQL injection attacks." Advance Computing Conference, 2009. IACC 2009. IEEE International. IEEE, 2009.

[9] Halfond, William GJ, Alessandro Orso, and Panagiotis Manolios. "WASP: Protecting Web applications using positive tainting and syntax-aware evaluation." Software Engineering, IEEE Transactions on 34.1 (2008): 65-81.

[10] McClure, Russell A., and Ingolf H. Krüger. "SQL DOM: compile time checking of dynamic SQL statements." Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on. IEEE, 2005.

[11] Johari, Rahul, and Pankaj Sharma. "A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection." Communication Systems and Network Technologies (CSNT), 2012 International Conference on. IEEE, 2012.

[12] Ezumalai, R., and G. Aghila. "Combinatorial approach for preventing SQL injection attacks". Advance Computing Conference, 2009. IACC 2009. IEEE International. IEEE, 2009.

[13] Parker, Donn B. "Toward a new framework for information security." FLY(2002): 501.

[14] Parker, Donn B. "Toward a new framework for information security." FLY(2002): 501.

[15] Xie, Yichen, and Alex Aiken. "Static Detection of Security Vulnerabilities in Scripting Languages." USENIX Security. Vol. 6. 2006.

[16] Buehrer, Gregory, Bruce W. Weide, and Paolo AG Sivilotti. "Using parse tree validation to prevent SQL injection attacks." Proceedings of the 5th international workshop on Software engineering and middleware. ACM, 2005.

[17] Halfond, William G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." Proceedings of the IEEE International Symposium on Secure Software Engineering. Vol. 1. IEEE, 2006.