Survey and Comparative Analysis of SQL Injection Attacks, Detection and Prevention Techniques for Web Applications Security

Pooja Saini
Department of Computer Science & Engg.
Doon Valley Institute of Engg. & Technology
Karnal, India
poojasaini30@gmail.com

Sarita
Department of Computer Science & Engg.
Doon Valley Institute of Engg. & Technology
Karnal, India
saritachaudharydiet@gmail.com

Abstract— Web applications witnessed a rapid growth for online business and transactions are expected to be secure, efficient and reliable to the users against any form of injection attacks. SQL injection is one of the most common application layer attack techniques used today by hackers to steal data from organizations. It is a technique that exploits a security vulnerability occurring in the database layer of a web application. The attack takes advantage of poor input validation in code and website administration. It allows attackers to obtain illegitimate access to the backend database to change the intended application generated SQL queries. . In spite of the development of different approaches to prevent SQL injection, it still remains a frightening risk to web applications. In this paper, we present a detailed review on various types of SQL injection attacks, detection and prevention techniques, and their comparative analysis based on the performance and practicality.

Keywords- *SQL injection attacks, prevention, detection, vulnerabilities.*

*****

I. INTRODUCTION

Nowadays, web applications are common in the online world. Nearly every major company or organization has a web presence and use web applications to provide various online services to users. Some of these web applications use database driven content. Data and Information is the most important business asset in today's environment for achieving an appropriate level of Information Security. Most of these web applications are vulnerable to a variety of new security threats. One of the most threats to web application is SQL injection attack. An SQL Injection Attack (SQLIA) is a type of intrusion whereby a crafted attacker adds malicious keywords or operators into an SQL query and then injects it to a user input box of a web application [1]. This allows the attacker to have illegitimate and unrestricted access to the data stored at the backend database which often contains confidential and sensitive information such as security numbers, credit card number, financial data, and medical data.

According to Open Web Application Security Project (OWASP), SQL injection attacks (SQLIA) stands first in the top 10 threats for web application security in 2013 [2]. Top 10 threats are SQLIA, Cross Site Scripting (XSS), Malicious File Execution, Insecure Direct Object Reference, Cross Site Request Forgery (CSRF), Information Leakage, Improper Error Handling, Broken Authentication, Session Management, and Insecure Cryptographic Storage. In SQL injection attack, attacker provides SQL code rather than the legitimate input in the input fields of the web application in order to vary the meaning of the original SQL query issued by the backend database. Once the attacker gains access to the database, it can alter any sensitive information or even modify the web application. To implement security guidelines inside or outside of the database, database security needs to be monitored.

Detection and prevention of SQL injection attacks are a topic of active research in the academia and industry. To achieve these purposes, automatic tools and security system were implemented, but none of them were complete or accurate enough to guarantee an absolute level of security of web applications. The aim of the paper is to review various types of SQL injection vulnerabilities, attacks, and prevention techniques and also present the comparative analysis of various SQL injection prevention techniques and attack types.

II. SQL INJECTION BACKGROUND

A. Why SQL injection is a major threat to web application security?

Injecting a web application is the synonym of having access to the data stored in the database. The data sometimes could be confidential and of high value like the financial secret of a bank or list of financial transactions or secret information of some kind of information system, etc. An unauthorized access to this data by a crafted attacker can threat their integrity, confidentiality and authority. As a result, the system could bear heavy loss in giving proper services to its users or it may face complete destruction. SQL injection is most commonly used by hackers to steal data from information systems of organizations. If it happens against the information systems of a hospital, the confidential information of the patients may be leaked out which could threaten their reputation or may be a case of depreciation [3]. These attacks are designed not only to crack the security and steal the entire content of the database, but also, to make superficial changes to both the database schema and contents. Hence, SQL injection could be very threatening in many cases depending on the platform where the attack is projected and it gets success in injecting rogue users to the target system.

B. *SQL Injection Vulnerabilities*

An SQL injection is a kind of injection vulnerability in which the attacker tries to inject arbitrary pieces of malicious data into the input fields of an web application, when processed by the application, causes that data to be executed as a segment of code by the backend SQL server, thereby giving undesired results which the developer of the web application did not anticipate, leveraging almost a complete compromise of system in most cases. Three types of the most common security vulnerabilities - Type I, II and III are found in web programming languages are presented in Table I.

TABLE I. TYPES OF VULNERABILITIES

Vulnerability type	**Description**
Type I	Validation of the user supplied data is not properly defined or sanitized and allowed to be executed with intention. Attacker takes advantage of poor input validation can utilize malicious code to conduct attacks.
Type II	Lack of clear dissimilarity between data types accepted as input in the programming language used for the web application development.
Type III	Delay of operation analysis till the runtime phase where the current variables are considered rather than the source code expressions.

C. *Types of SQL injection attacks*

Seven different types of injection attacks are performed together or sequentially depending on the goal of attacker [4]. For a successful SQLIA, the attacker should append a syntactically correct command to the original SQL query. The following seven types of SQLIAs are presented in Table II.

TABLE II. DIFFERENT TYPES OF SQL INJECTION ATTACKS

Type of attack	**Working Method**
Tautologies	SQL injection queries are injected using the conditional OR operator such that the query always evaluates to be TRUE.
Logically Incorrect Queries	The attacker tries to gather information from the rejected error messages about the type and structure of the backend database of a web application to find useful data facilitating injection to the database.
Union Query	This type of attack can be done by inserting a UNION query into a vulnerable parameter which returns a dataset that is the union of the result of the original first query and the results of the injected query.
Stored Procedure	Most of the databases have standard set of procedures that extend the functionality of the database and allow for interaction with the operating system. The attacker tries to execute store procedures using malicious SQL injection codes.
Piggy-Backed Queries	The attacker tries to inject additional malicious queries along with the original query resulting the database receives multiple SQL queries for execution. Vulnerability of this kind of attack is dependent of the kind of database.
Inference attack	The intruder changes the behaviour of a database of web application.
Blind Injection	- The attack is applied on well secured databases which do not return any usable feedback or descriptive error messages. The attack is created in the style of true/false statement.
Timing Attacks	- The attacker designs a conditional statement and injects through the vulnerable parameter and gather information based on time delays in the response of the database.
Alternate Encodings	The injected text is modified so as to avoid detection by defensive coding practices and also many automated prevention techniques. It is usually combined with other attack techniques.

III. DETECTION OF SQL INJECTION

There are two major tasks to protect a web application from SQL Injection attacks [5]. Firstly, there is an extreme need of a technique to detect and exactly identify SQL injection attacks. Secondly, proficiency of SQL Injection Vulnerabilities (SQLIVs) is a must for protecting a web application. So far, many frameworks have been suggested to detect SQL injection vulnerabilities in web applications. Here, some of the pronounced solution and their working methods are discussed in brief.

A. *Shin et al.'s approach*

In this approach the authors applies SQLUnitGen tool which is compared with FindBugs, a static analysis tool. SQLUnitGen, a static analysis based tool that automates testing for identifying input manipulation vulnerabilities [6]. The proposed approach is shown to be efficient as the fact that the false positive was completely absent in the tests. However for different schemes, false negatives at a small number were noticed.

B. *Fu et al approach*

The authors suggested the Static Analysis Framework for Discovering SQL Injection Vulnerabilities (called as SAFELI) in order to detect SQL Injection Vulnerabilities during compiling [7]. The static analysis tool performed a White-box Static Analysis and used a Hybrid Constraint Solver. In case of White-box we found the Static Analysis, the proposed approach considered the byte-code and dealt mainly with strings. While on the other hand, the Hybrid Constraint Solver implemented the methods to an efficient string analysis tool which is able to dealt with integer, Boolean and variables of string.

C. *Roichman and Gudes's Scheme*

This scheme was developed based on fine-grained access control to the databases of web applications [8]. The database access is monitored and supervised by the built in database access control of web applications. This is a solution to the vulnerability of the SQL session traceability. Moreover, it is a framework applicable to almost all database applications. This scheme is shown to be efficient in the fact that the security and access control of the database of web applications is transferred from the application layer to the database layer.

D. SQL-IDS Approach

Kemalis and Tzouraman proposed a specification based mechanism for the detection of vulnerabilities of SQL injection [9]. The query specific detection allowed the web application to perform direct analysis at inconsequential computational overhead without producing false positives or negatives. The proposed approach is shown to be very efficient in operation; however, it requires more analysis and comparison with accessible detection techniques under a shared and flexible benchmarking environment.

E. Thomas et al.'s Scheme

To remove SQL injection vulnerabilities this scheme proposed an automated prepared statement generation algorithm [10]. The research work was implemented using using four open source proposals namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. On the basis of analysis results, their prepared statement code was able to auspiciously replace 94% of the SQL injection vulnerabilities in four open source proposals. The analysis was carried out using only Java with a limited number of proposals. Hence, the use of web application of the same approach and tool for different settings still remains an open research concern to explore.

F. Haixia and Zhihong's Scheme

The authors proposed a protected database design testing for web applications [11]. They suggested few methods: detection of possible input points of SQL Injection, generation of test cases automatically; then finally finding the database vulnerability by running the test cases to make a simulation attack to a web application. The proposed scheme is efficient as it detect the input points of SQL injection exactly on expected time.

G. Ruse et al.'s Approach

To detect SQL injection vulnerabilities Ruse et al. proposed an approach that used automatic test case generation [12]. The proposed framework is based on creating a specific prototype that dealt with SQL queries axiomatically. This scheme also identifies the dependency between sub-queries. On the experimental basis, the proposed approach is shown to be efficient to specifically identify the causal set and obtain 85% and 69% reduction respectively while analysis on few samples.

IV. PREVENTIVE TECHNIQUES OF SQL INJECTION

A strong and effective preventive measure can remove or at least block all the available vulnerabilities in a web application and thus it could protect it against various types of attacks that take advantage of the vulnerabilities. We enlist twelve preventive techniques that could be employed before and during running the system. It should be noted that these approaches not only detect SQL Injection, but also take necessary measures so that the vulnerabilities are not exploited by the rogue entities. So, these are different from the approaches mentioned in the earlier section in the point that they do more than just detection of SQL Injection.

A. SQLrand Scheme

Boyd and Keromytis proposed a SQLrand approach using randomized SQL query language, targeting a particular Common Gateway Interface application [13]. This scheme provides a framework that allows developers to create queries using randomized instructions instead of normal SQL keywords. The proxy filter prevents queries to the database and de-randomizes the keywords. SQL code injected by an attacker would not have been constructed using the randomized instruction set. Therefore, injected commands would result in a syntactically incorrect query. The proposed scheme has a good performance: 6.5 ms is the maximum latency overhead imposed on every query.

B. SQL DOM Scheme

McClure and Krüger suggested a framework SQL DOM (a set of classes that are strongly-typed to a database schema) [14]. They intently consider the current flaws while accessing relational databases from the Object Oriented Programming Languages point of view. They mainly target on identifying the hindrance in the interaction with the database via Call Level Interfaces. The SQL DOM object prototype is the proposed solution to implement these issues through building a protective environment for communication.

C. Parse Tree Validation Approach

Buehrer et al. compared the parse tree framework of a particular statement at runtime and its original statement [15]. They terminated the execution of statement unless there is a contest. This methodology was experimented on a web application of student using SQLGuard. However, this approach is shown to be efficient, but it has two major limitations: additional overheard computation and listing of black and white input.

D. SQLCHECK Approach

With SQLCHECK Su and Wassermann [16] implemented their algorithm on a real time environment. This approach checks whether the input queries approve to the expected ones defined by the web application programmer. A confidential key is applied for the user input delimitation. The experiment of SQLCHECK shows no false positives or false negatives. The overhead runtime rate is very less and could be executed directly in many other web based applications using different languages.

E. DIWeDa Approach

For the backend databases, Roichman and Gudes proposed Intrusion Detection Systems [17]. Authors used DIWeDa (Detecting Intrusions in Web Databases), a model which acts at the session level rather than the SQL statement or transaction stage, to detect the intrusions in web based applications. The proposed approach is shown to be efficient and could identify SQL injections and business logic violations. There is a need to be tested against new types of SQL injection attacks and requires a great need of accuracy improvement.

F. *Ali et al.'s Scheme*

Ali et al proposed the hash value scheme to further improve the user authentication method [18]. They used the hash values of username and password. SQLIPA (SQL Injection Protector for Authentication) model was developed in order to test the framework. The hash values of username and password are created and calculated at runtime for the first time the particular user registered itself. On few sample data the proposed framework was tested. This scheme had an overhead of 1.3 ms, which requires more improvement to reduce the overhead time and also requires to be tested with larger amount of user's record.

G. *Manual Approach*

MeiJunjin used manual approach to prevent SQLI (SQL Injection Input) manipulation flaws [19]. In manual approaches, code review and defensive programming are applied. In code review, it is a low cost mechanism in detecting bugs, however, this approach requires deep knowledge on SQLIAs. In defensive programming, an input filter is implemented to disallow users to input malicious keywords or characters. This is attained by using white lists or black lists.

H. *Automated Approach*

MeiJunjin [19] also used automated approaches. The authors implemented two frameworks, Static analysis FindBugs and web vulnerability scanning. Static analysis FindBugs approach detects bugs on SQLIAs, gives message when an SQL query is made of variable. However, for the web vulnerability scanning, it uses software agents to poke, scans web applications and detects the SQL injection vulnerabilities by examining their observance to the attacks.

I. *WebSSARI (Web application Security by Static Analysis and Runtime Inspection)*

The SQLIA prevention in stored procedures is executed by a combining static analysis and runtime analysis [20]. In the databases the stored procedures are subroutines which the web applications can make call to. For commands identification the static analysis used is achieved through stored procedure parser and the runtime experiment by using a SQLChecker for input identification. Huang et al. [21] proposed a combination of runtime monitoring and static analysis to fortify the protection of major vulnerabilities. The scheme was effective and however it failed to abolish the SQLIVs. This scheme was only able to list the input either white or black.

J. *AMNESIA*

Junjin suggested an AMNESIA (Analysis and Monitoring for Neutralizing SQL Injection Attacks) mechanism that combines runtime monitoring and static analysis [22]. In the static phase, AMNESIA uses static analysis to build models of different types of queries an application can legally generate at each point of access to the database. In dynamic phase, AMNESIA prevents all queries before they are sent to the database and checks each query against the statically built models. Queries that breach the model are identified as SQLIAs and prevented from executing on the database. The proposed framework is efficient considering the fact that it emphasizes on attack input precision. The input of attack is exactly matched with arguments method. The only limitation of this scheme is that it involves a number of steps using different tools.

K. *Dynamic Candidate Evaluation Approach*

Bisht et al. proposed CANDID (CANdidate Evaluation for Discovering Intent Dynamically) [23] for automatic prevention of SQL Injection attacks. The proposed approach dynamically excerpts the query structures from every SQL query location which are designed by the web application programmer. Hence, it solves the matter of manually altering the web application to create the prepared statements. Though this framework is shown to be efficient for some cases, it fails in many other cases. It is not efficient when dealing with external functions and when applied at a wrong stages. Due to limited capability of the approach sometimes it also fails.

L. *Removing SQL query attribute values*

Authors proposed an approach to detect SQL injection attacks is based on static and dynamic investigation [24]. It removes the attribute values of SQL queries at runtime and compares them with the SQL queries analyzed in advance to detect the SQL injection. When execute the application each dynamical generated query is compared or performs XOR operation.

V. COMPARATIVE ANALYSIS

It would be difficult to give a clear finding which scheme or approach is the best as each one has some confirm benefits for specific types of settings (i.e., systems). We analyzed how various schemes work against the identified SQL Injection attacks. The symbol (√) is used for techniques that can successfully detect all attacks of SQL injection type. The symbol (X) is used for techniques that are not able to detect all attacks of that type. Though many approaches have been identified as detection or prevention techniques, only few of them were implemented in practicality. Hence, this comparison is based on analytical evaluation only.

TABLE III. COMPARATIVE ANALYSIS OF VAROIUS APPROACHES AND SQL INJECTION ATTACKS

Approach	**Tautology**	**Logically Incorrect Queries**	**Union Query**	**Stored Procedure**	**Piggy Backed Queries**	**Inference**	**Alternate Encodings**
SQLrand	✓	✓	✓	X	✓	✓	✓
SQL DOM	✓	✓	✓	X	✓	✓	✓
SQL CHECK	✓	✓	✓	X	✓	✓	✓
DIWeDa	X	X	X	X	X	✓	X
SQLIPA	✓	X	X	X	X	X	X
Automated Approaches	✓	✓	✓	X	✓	✓	X
WEbSSARI	✓	✓	✓		✓	✓	✓
AMNESIA	✓	✓	✓	X	✓	✓	✓

Approach	**Tautology**	**Logically Incorrect Queries**	**Union Query**	**Stored Procedure**	**Piggy Backed Queries**	**Inference**	**Alternate Encodings**
CANDID	✓	X	X	X	X	X	X
SQLGuard	✓	✓	✓	X	✓	✓	✓
Remove attrubute value	✓	✓	✓		✓	✓	✓

Table III shows the schemes and their defense capabilities against various SQLIAs. This table shows the comparative analysis of the SQL Injections prevention techniques and the attack types. Two attack types, stored procedures and alternate encodings, caused problems for most techniques. In case of stored procedures, the code that generates the query is stored and executed on the database. Many of the techniques considered focused only on queries generated within the application. Expanding the techniques to also enclose the queries generated and executed on the database is not truthful and would, in general, require substantial effort. For this reason, attacks based on stored procedures are ambiguous for many techniques. The attacks that are based on alternate encoding are also difficult to handle. Only three techniques, SQLCheck, AMNESIA, and SQLGuard precisely address these types of attacks. The reason why these techniques are advantageous against such attacks is that they use the database parser to clarify a query string in the same way that the database would. Other techniques that score well in this section are either developer-based techniques (i.e., WebSSARI) or techniques that address the problem by using a standard (Application Programming Interface) API (i.e., SQL DOM). It is important to note that we did not take precision into account in our assessment. Most of the techniques that we consider are based on some conservative analysis or assumptions that may result in false positives.

TABLE IV. COMPARATIVE ANALYSIS OF VARIOUS APPROACHES AND TYPES OF TASKS

Approach	**Tasks**	
	Detection	***Prevention***
SQLrand	✓	✓
SQL DOM	✓	✓
SQL CHECK	✓	X
DIWeDa	✓	✓
SQLIPA	✓	X
Automated Approaches	✓	✓
WEbSSARI	✓	✓
AMNESIA	✓	✓
CANDID	✓	X
SQLGuard	✓	X
Removing SQL query attrubute value	✓	X

Table IV shows the major approaches to deal with SQL injection and classify them based on their features. For the comparison purpose, we split the techniques into two groups: prevention focused and detection focused techniques. Prevention focused techniques are techniques that statically identify SQL injection vulnerabilities in the code, propose a different development criterion for applications that generate SQL queries, or add checks to the application to enforce best defensive coding practices. Detection focused techniques are techniques that detect attacks mainly at runtime. The prevention focused techniques adequately handle all of the attack types considered. We believe that, overall, the prevention focused techniques performed well because they integrate the best defensive coding practices in their prevention mechanisms. Most of the detection focused techniques perform fairly uniformly against the various SQL injections attack types.

VI. CONCLUSION

In this paper we have presented a detailed survey on various types of SQL Injection attacks, vulnerabilities, and detection and prevention techniques and evaluated techniques based on their performance and practicality. We compared SQL injection detection and prevention techniques analytically. We also compared these techniques in terms of their deployment and evaluation criteria. This research outcome helps to measure the security level of web applications using proposed tools, to detect vulnerabilities of online applications and to protect applications against using proposed secure coding approaches. As a future work, we would like to develop a hash function based authentication scheme that can efficiently tackle the innovative SQL injection attacks and fix as much vulnerability as possible.

ACKNOWLEDGMENT

I am sincerely thankful to Er. Sandeep Jain, Head and staff of Department of Computer science and Engg, Doon Valley Institute of Engg. & Technology, Karnal. We also wish to thank all the anonymous reviewers for their valuable suggestions, who helped in improving the manuscript.

REFERENCES

[1] Kindy, D.A. and Pathan, A.-S.K., “A Survey on SQL Injection: Vulnerabilities, Attacks, and Prevention Techniques”, (Poster)Proceedings of The 15th IEEE Symposium on Consumer Electronics (IEEE ISCE 2011), June 14-17, Singapore 2011, pp. 468-471.

[2] Top 10 2013-A1-Injection, available at: http://www.owasp.org/index.php/Top_10_2013- A1-Injection, last accessed 11 June, 2013.

[3] Bojken, A. Shqiponja, A. Marin, and Xh. Aleksander,"Protection of Personal Data in Information Systems",International Journal of Computer Science, Vol. 10, No. 2,July 2013, ISSN (Online): 1694-0784.

[4] Prasant Singh Yadav, Pankaj Yadav, K.P.Yadav “A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications”, IJRREST: International Journal of Research Review in Engineering Science and Technology, Volume 1, Issune1, June 2012.

[5] A. Tajpour; M. JorJor Zade Shooshtari, "Evaluation of SQL Injection Detection and Prevention Techniques," Proc. of CICSyN, 2010, pp.216-221, 28-30 July 2010.

[6] Y. Shin, L. Williams and T. Xie, "SQLUnitGen: Test Case Generation for SQL Injection Detection," North Carolina State Univ., Raleigh Technical report, NCSU CSC TR 2006-21, 2006.

[7] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. "A Static Analysis Framework for Detecting SQL Injection Vulnerabilities", COMPSAC 2007, pp.87-96, 24-27 July 2007.

[8] Roichman, A., Gudes, E." Fine-grained Access Control to Web Databases".In: Proc. of 12th SACMAT Symposium, France 2007.

[9] K. Kemalis, and T. Tzouramanis, "SQL-IDS: A Specification based Approach for SQLinjection Detection", SAC'08. Fortaleza, Ceará,Brazil, ACM: pp. 2153 2158, 2008.
Programs", 10th Annual International Symposium on Applications and the Internet pp.31 – 37, 2010.

[13] S. W. Boyd and A. D. Keromytis. "SQLrand: Preventing SQL Injection Attacks" In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pages 292–302, June 2004.

[14] R.A. McClure, and I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 88- 96, 15-21 May 2005.

[15] G. Buehrer, B.W. Weide, P.A.G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", In: 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, pp.106–113. 2005.

[16] Z. Su and G. Wassermann "The essence of command injection attacks in web applications". In ACM Symposium on Principles of Programming Languages (POPL'2006), January 2006.

[17] A. Roichman, E. Gudes, "DIWeDa - Detecting Intrusions in Web Databases". In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 313–329. Springer, Heidelberg 2008.

[18] S. Ali, SK. Shahzad and H. Javed, "SQLIPA: An Authentication Mechanism Against SQL Injection," European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4, pp 604-611, 2009.

[19] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of ITNG '09, pp.1411-1414, 27-29 April 2009.

[20] K. Amirtahmasebi, S. R. Jalalinia, S. Khadem, "A survey of SQL injection defense mechanisms," Proc. Of ICITST 2009, vol., no. pp.1-8, 9-12 Nov. 2009.

[21] Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T., and Kuo, S.-Y. "Securing Web Application Code by Static Analysis and Runtime Protection", Proc. of 13th International Conference on World Wide Web, NewYork, NY, pp. 40-52, 2004.

[22] M. Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of the 6th Int. Conf. on Information Technology: New Generations, Las Vegas, Nevada, pp. 1411-1414, April 2009.

[23] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks", ACM Trans. Inf. Syst. Secur., 13(2):1–39, 2010.

[24] I. Lee, S. Jeong, S. Yeoc, J. Moond, "A novel method for SQL injection attack detection based on removing SQL query attribute", Journal Of mathematical and computer modeling, Elsevier 2011.

[10] s. Thomas, L. Williams, and T. Xie, "On automated prepared statement generation to remove SQL injection vulnerabilities. Information and Software Technology" 51, 589–598, 2009.

[11] Y. Haixia, N. Zhihong, "A database security testing scheme of web application," Proc. of ICCSE '09 , pp. 953-955, 25-28 July 2009.

[12] M. Ruse, T. Sarkar and S. Basu. "Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of