IJICTR International Journal of Information & Communication Technology Research

ITRC

Volume 5- Number 3- Summer 2013

Technical Note

A Framework for Evaluation of SQL Injection Detection and Prevention Tools

Atefeh Tajpour
Advanced Informatics School
University Technology Malaysia
tajpour81sn@yahoo.com

Suhaimi Ibrahim
Advanced Informatics School
University Technology Malaysia
suhaimiibrahim@utm.my

Received: November12, 2011- Accepted: December 25, 2012

***Abstract*— SQLIA is a hacking technique by which the attacker adds Structured Query Language code (SQL statements) through a web application's input fields or hidden parameters to access the resources. By SQL injection an attacker gains access to underlying web application's database and destroys functionality and/or confidentiality. Researchers have proposed different techniques to detect and prevent this vulnerability. In this paper we present SQL injection attack types and also current security tools which detect or prevent this attack and compare them with each other. Finally, we propose a framework for evaluating SQL injection detection or prevention tools in common criteria. In fact, this paper provides information about current tools for researchers and also helps security officers to choose suitable SQL injection detection tools for their web application security.**

Keyword— ***web application security, web application vulnerability, SQL Injection attack, framework, tool, evaluation, comparison***

1. INTRODUCTION

In recent years, most of our daily tasks are dependent on database driven web applications because of increasing activity, such as banking, booking and shopping. Web has become business-oriented and is the preferred interface for information and services around the world [2] consequently, information must be trustable to web applications and their underlying databases but unfortunately there is not any guarantee for confidentiality and integrity of this information. In particular, remote attacks which exploit one or more vulnerabilities to seize control or break down vulnerable hosts over the Internet are dramatically increasing [4]. Refers to the TOP-10 web applications vulnerabilities for 2007 by OWASP, SQL Injection Attacks (SQLIAs) have known as one of the most common threats to the security of database-driven web application. In other word, there is not enough assurance for confidentiality and integrity of this information. SQLIA is a class of code injection attacks that takes advantage of lack of user input validation. In fact, attackers can shape their illegitimate input as parts of final query string which is operated by databases. Financial web applications or secret information systems could be the victims of SQLIA because attackers can threat their authority, integrity and confidentiality. So, developers addressed some defensive coding practices to eliminate vulnerabilities but they are not sufficient. Some researcher propose firewalls and Intrusion Detection Systems (IDSs) but they are not enough because SQLIA performs through ports used for regular web traffic which usually are open in firewalls. On the other hand, most IDSs focus on the network and IP layers whereas SQLIA works at application layer.

Not only developers try to put some controls in their source code, but attackers also continue to bring some new ways to bypass these controls. These problems motivate the need for a solution to the SQL injection problem. Researchers have proposed some tools to help developers to compensate the shortcoming of the defensive coding [7, 10, 12].

The problem is that some current tools could not address all attack types or some of them need special deployment requirements. Moreover, some tools suffer from weakness in efficiency and effectiveness, performance and stability. For example some tools support special programming language.

Finally, this paper focuses precisely on a framework for evaluating SQL Injection Detection or Prevention tools in different criteria to help users choose an appropriate security tool. Choosing a tool, only according the numbers in the articles, is not reasonable, today. Because some of the results are depended on programming language, operating system, database, attack list as well as equipment which had been used by the authors of tools in evaluation process. This framework provides common criteria for evaluating common measure parameters.

2. OVERVIEW OF SQL INJECTION ATTACK

2.1.Definition of SQLIA

SQLIA is a hacking technique which the attacker adds Structured Query Language code (SQL statements) through a web application's input fields or hidden parameters to access to resources. Lack of input validation in web applications causes hacker to be successful. For the following examples we will assume that a web application receives a HTTP request from a client as inputs and generates a SQL statement as output for the backend database server.

1. an attacker sends the malicious HTTP request to the web application
2. creates the SQL Statement
3. submits the SQL Statement to the back end database

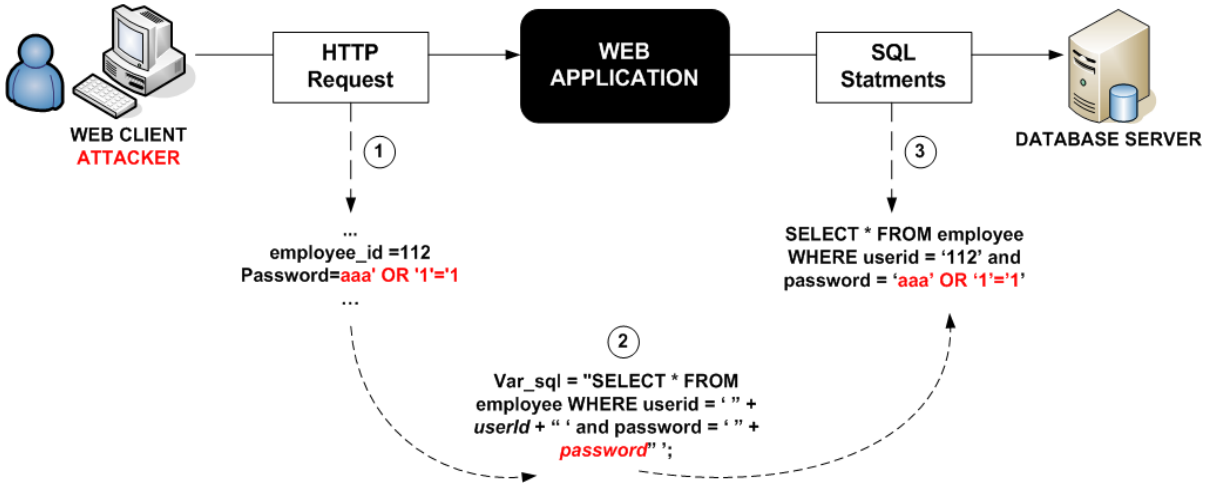

Figure 1. Example of a SQL Injection Attack

The above SQL Statement is always true because of the Boolean tautology we appended (OR 1=1) so, we will access the web application as an administrator without knowing the right password. By using SQLIAs, an attacker may be able to read, modify, or even delete database information. In many cases, loss of sensitive or confidential information can lead to problems such as identity theft and fraud.

2.2. Sql Injection Attack Types

There are different methods of attacks that, depending on the goal of attacker, are performed together or sequentially. For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query. There are more explanation and examples in [28],[20]. Tautologies: This type of attack injects SQL tokens to the conditional query statement to be evaluated always true. Illegal/Logically Incorrect Queries: when a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. Union Query: By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application.

Piggy-backed Queries: In this type of attack, intruders exploit database by the query delimiter, such as ";", to append extra query to the original query. With a successful attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate.

Stored Procedure: Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. As stored procedure could be coded by programmer, so, this part is as inject able as web application forms. Depend on specific stored procedure on the database there are different ways to attack.

Alternate Encodings: In this technique, attackers odify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode.

Because by this way they can escape from developer's filter which scan input queries for special known "bad character". For example attacker use char (44) instead of single quote that is a bad character. Inference: By this type of attack, intruders change the behaviour of a database or application.There are two well-known attack techniques that are based on inference: blind-injection and timing attacks.

- Blind Injection: Sometimes developers hide the error details which help attackers to compromise the database. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements.
- Timing Attacks: A timing attack lets an attacker gather information from a database by observing timing delays in the database's responses. This technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time.

2.3. SQL Injection Detection And Prevention Tools

Researchers have proposed a wide range of techniques to address the problem of SQL injection. These techniques range from development best practices to fully automated tools for detecting and preventing SQLIAs. In this section, these proposed tools would be reviewed and the advantages and disadvantages associated with each tool would be summarized [1]. It is noticeable that there are more techniques that have not implemented as a tool yet. This paper emphasizes on tools not techniques such as [23],[24]. Further information about techniques is available in [20].

Huang and colleagues [18] propose WAVES, a black-box technique for testing web applications for SQL injection vulnerabilities. The tool identifies all points a web application that can be used to inject SQLIAs. It builds attacks that target these points and monitors the application how response to the attacks by utilizing machine learning.

JDBC-Checker [12],[13] was not developed with the intent of detecting and preventing general SQLIAs, but can be used to prevent attacks that take advantage of type mismatches in a dynamically-generated query string. As most of the SQLIAs consist of syntactically and type correct queries so this technique would not catch more general forms of these attacks.

CANDID [7], [27] modifies web applications written in Java through a program transformation. This tool dynamically mines the programmer-intended query structure on any input and detects attacks by comparing it against the structure of the actual query issued. CANDID's natural and simple approach turns out to be very powerful for detection of SQL injection attacks.

In SQL Guard [10] and SQL Check [5] queries are checked at runtime based on a model which is expressed as a grammar that only accepts legal queries. SQL Guard examines the structure of the query before and after the addition of user-input based on the model. In SQL Check, the model is specified independently by the developer. Both approaches use a secret key to delimit user input during parsing by the runtime checker, so security of the approach is dependent on attackers not being able to discover the key. In two approaches developer should modify code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query.

AMNESIA combines static analysis and runtime monitoring [16],[17]. In static phase, it builds models of different types of queries which an application can legally generate at each point of access to the database. Queries are intercepted before they are sent to the database and are checked against the statically built models, in dynamic phase. Queries that violate the model are prevented from accessing the database. The primary limitation of this tool is that its success is dependent on the accuracy of its static analysis for building query models.

Web SSARI [15] use static analysis to check taint flows against preconditions for sensitive functions. It works based on sanitized input that has passed through a predefined set of filters. The limitation of this approach is adequate preconditions for sensitive functions cannot be accurately expressed so some filters may be omitted.

Security Fly [14] is another tool that was implemented for java. Despite of other tool, chases string instead of character for taint information. Security Fly tries to sanitize query strings that have been generated using tainted input but unfortunately injection in numeric fields cannot stop by this approach. Difficulty of identifying all sources of user input is the main limitation of this approach.

Positive tainting [1] not only focuses on positive tainting rather than negative tainting but also it is automated and does need developer intervention. Moreover this approach benefits from syntax-aware evaluation, which gives developers a mechanism to regulate the usage of string data based not only on its source, but also on its syntactical role in a query string.

IDS [6] use an Intrusion Detection System (IDS) to detect SQLIAs, based on a machine learning technique. The technique builds models of the typical queries and then at runtime, queries that do not match the model would be identified as attack. This tool detects attacks successfully but it depends on training seriously. Else, many false positives and false negatives would be generated.

Another approach in this category is SQL-IDS [8] which focuses on writing specifications for the web application that describe the intended structure of SQL statements that are produced by the application, and in automatically monitoring the execution of these SQL statements for violations with respect to these specifications.

SQL Prevent [11] is consists of an HTTP request interceptor. The original data flow is modified when SQL Prevent is deployed into a web server. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module. Consequently, HTTP request from thread-local storage is fetched and examined to determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLIA.

Sw addler [3], analyzes the internal state of a web application. It works based on both single and multiple variables and shows an impressive way against complex attacks to web applications. First the approach describes the normal values for the application's state variables in critical points of the application's components. Then, during the detection phase, it monitors the application's execution to identify abnormal states.

3.COMPARISON

In this section, the SQL injection detection or prevention tools presented in section IV would be compared together. It is noticeable that this comparison is based on the evaluation which the authors of tools have done empirically. They used a testbed for their tool. In particular, they used a set of web applications and a set of inputs for those applications that included both legitimate inputs and SQLIAs.

3.1. Comparison of SQL Injection Detection/Prevention Tools Based on Deployment Requirement

Each tool with respect to the following criteria was evaluated: (1) Does the tool require developers to modify their code base? (2) What is the degree of automation of the detection aspect of the tool? (3) What is the degree of automation of the prevention aspect of the tool? (4) What infrastructure (not including the tool itself) is needed to successfully use the tool? The results of this classification are summarized in Table1.

TABLE1. COMPARISON OF TOOLS BASED ON DEPLOYMENT REQUIREMENT

No	Tool	Modify Code base	Detection	Prevention	Additional Infrastructur
1	AMNESIA [16]	No	Auto	Auto	None
2	IDS [6]	No	Auto	Generate report	IDS system-Training set
3	JDBC Checker [12]	No	Auto	Code suggestion	None
4	Securify[14]	No	Auto	Auto	None
5	SQLCHECK [5]	Yes	Semi Auto	Auto	Key management
6	SQLGaurd [10]	Yes	Semi Auto	Auto	None
7	WAVES [18]	No	Auto	Generate report	None
8	WEBSSARY [15]	No	Auto	Semi auto	None
9	CANDID [7]	No	Auto	Auto	None
10	SQL_IDS [8]	No	Auto	N/A	None
11	Swaddler [3]	No	Auto	Auto	Training
12	Positive Tainting [1]	No	Auto	Auto	None
13	SQLPrevent [11]	No	Auto	Auto	None

Table1 determines the degree of automation of tool in detection or prevention of attacks. Actually automatically detection and prevention is an ability of the tool that provides user satisfaction. Also table shows that which tool needs to modify the source code of application. Moreover, additional infrastructure that is required for each tool that usually leads to inconvenience for users is illustrated.

3.2. Comparison of Sql Injection Detection/Prevention Tools Based on Attack Types

Proposed tools were compared to assess whether it was capable of addressing the different attack types presented in section2. Tables 2 summarize the results of this comparison.

The symbol "•" is used for tool that can successfully stop all attacks of that type. The symbol "-" is used for tool that is not able to stop attacks of that type. The symbol "°" refers to tool that the attack type only partially because of natural limitations of the underlying approach.

As the table shows the stored procedure is a critical attack which is difficult for some tools to stop it. It is consisting of queries that can execute on the database. However, most of tools consider only the queries that generate within application. So, this type of attack make serious problem for some tools.

Table2 Comparison of Tools with Respect to Attack Types

	Attack \ Tool	SQL_IDS[8]	Swaddler[3]	IDS[6]	CANDID[7]	AMNESIA[16]	SQL Check[5]	SQL Guard[10]	JDBC-Checker[12]	WebSSARI[15]	SecuriFLY[14]	WAVES[18]	Positie Tainting[1]	SQLPrevent[11]
1	Tautologies	•	°	°	°	•	•	•	°	•	°	°	•	•
2	Illegal/ Incorrect	•	°	°	°	•	•	•	°	•	°	°	•	•
3	Piggy-back	•	°	°	°	•	•	•	°	•	°	°	•	•
4	Union	•	°	°	°	•	•	•	°	•	°	°	•	•
5	Stored Proc	•	°	°	°	-	-	-	°	•	°	°	•	•
6	Infer	•	°	°	°	•	•	•	°	•	°	°	•	•
7	Alter Encodings	•	°	°	°	•	•	•	°	•	°	°	•	•

3.3. Comparison of Tools Based on Evaluation Parameters

The authors of proposed tools have evaluated their tools in common parameters: efficiency, effectiveness and performance, flexibility and stability. The results of this classification are summarized in Table 3. Definition of the measured parameters [11], [29]:

Efficiency

- False positive: is a false alarm. It is when the tool incorrectly categorizes a benign request being as a malicious attack.
- False negative: occurs when a malicious attack is not recognized, so the tool lets it pass normally.

Effectiveness

- Attacks Detection: the percentage of real attacks, correctly detected.
- Attacks Prevention: the percentage of real attacks correctly blocked after being detected.

Flexibility

Different Types of SQLIAs: the ability of the tool to detect/prevent different types of SQL Injection attacks such as those presented in section II.

Performance

- Detection Overhead: is the time spent for a detection of a SQLIA once the tool is running.

– Prevention Overhead: is the time spent to detect and block (prevent) a SQLIA once the tool is running.

Stability

Environment Independence

– Web Applications: the possibility to test the tool on different types of web applications, such as open source/commercial, large/small.
– Databases: testing on web applications that use different backend databases, such as open source (e.g. MySQL) commercial (e.g. Oracle).
– Programming Languages: the ability of the tool to work on web applications written in different programming languages, such as J2EE, .NET, PHP and so On.
– Operating Systems: the ability of the tool to run on different OS such as Windows and Linux.
– Application Servers: the possibility to run the tool in a network using different type of Application Server such Tomcat.
  – Which parameters are important?
  – How important parameters could be measured?

4. Detailed Framework Diagrams

The data flow through the framework starts with “Create Testbed” and continue with ”Perform SQLIAs without Tool”, “ Install Tool”, ”Re-perform SQLIAs with Tool” and “Analyze result”.

4.1. Create Testbed

Figure2 illustrates that the testbed is made up by five main components which are all related to each other, in fact, mostly choosing a component depends on the others. For example, if "Vulnerable web applications” is written in PHP then, MySQL should be selected as Dtabase, consequently “Operating System” and “Application Server” should be selected compatible with database and programming language.

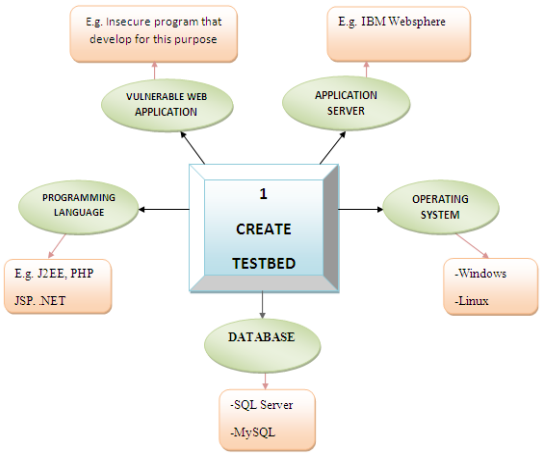

Figure2. Create Test bed

Actually, this step is complex and time-consuming because a vulnerable web application should be settled with other components which work together, then evaluation can be done.

4.2.Perform SQLIAs without Tool

After preparing the test bed, we perform different types of SQLIAs on the vulnerable web application without the security tool that is not installed yet. It is noticeable that the chosen web application in test bed is insecure and vulnerable to SQL Injection, so it should be possible to perform SQLIAs. This role can be done not only manually, but also with the support of automatic tools for penetration tests or scripts.

For a successful SQLI attack a pairs of data: vulnerable page and a parameter should be considered. For example:"login.jsp" as a vulnerable page and "username and password" as parameters. For each of these parameters, possible attack should be identified. Then an attack list and a benign list for the insecure web application could be written as a set of scripts and to submit the created lists automatically. Also run a penetration test using that set is effective.

A penetration test is a method to assess the security of a computer system or network by simulating an attack by a malicious user. This includes an active surveillance system for all potential weak points. Unknown hardware and software failures and weaknesses in operating procedures or technical countermeasures or improper system configuration can cause these weak points. This analysis will be performed by a potential attacker and can involve active exploitation of vulnerabilities. The intent of a penetration test is to determine the feasibility and impact of an attack if successfully done. In fact, it is part of a complete security audit.

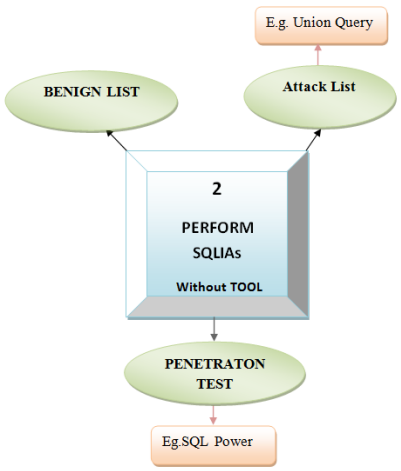

Figure3. Perform SQLIA without Tool

4.3.Install Tool

After examining the web application by a successful set of SQLIAs, the security tool should be installed for the assessment in this regard.

In fact this phase is fully depended on the selected tool for evaluation which means each tool has special process, characteristics and limitations. A few of them with small manipulation can run despite of those tools which need major configuration in their source core and environment to adopt them with new situation.

4.4.Re-Perform SQLIAs with Tool

The process in this phase is exactly as the same as phase 2. The only difference is, before this phase, security tool has been installed to guarantee the web application. Here the same attack list, benign lists will be repeated and penetration testing as explained above will be performed.

4.5.Analyze Result

After following the above phases there are different results which should be analyzed. So with the support of tables, charts and graphs, we can observe each parameter, with different results. For example, there are results for false negative and false positive. Consequently some useful conclusion and judgment on the SQLIAs security tool could be achieved.

4.6. Change Parameter and Loop

Once a complete loop of all the 5 phases has been done, it is suggested that the whole process to be repeated using other vulnerable web applications or database, operating system and application server.

In addition, it is useful to change attack list and benign list, also penetration test to get valuable results. Iteration of the evaluation process is essential to obtain useful results because each time that the process is repeated, different result may be obtained. In fact the number of repetition is depended on the examiners when he or she can trust to result to achieve the goal. On the other hand, it is clear that only one iteration is inadequate. Evaluation takes a long time, especially if is done particularly and correct.

5.COMPLETE EVALUATION FRAMEWORK

Figure 4, summarizes all steps of evaluation of the proposed framework. It shows important components related to each other and all of them are effective in measuring the parameters that show the ability and characters of a SQL injection detection or prevention tool. Also this framework shows how these components are related to each other. Moreover, by this framework the logic of data follow is understandable for viewers. Each phase identifies the related components and the result for each phase such as programming language, web application vulnerability, application server, database and operating system with Stability that could be measured by phase1.

The Table4 gives an overall view about components, output and also parameters that will be measured in each phase. This table gives some information in detail about the processes in the framework. The proposed framework is understandable and clear enough to be utilized for the evaluation of different SQLIAs detection/prevention tool. In fact, it does not have limitation and provides a standard trend and common procedures for evaluation process. It provides common criterion and useful results. Then, based on these results, the comparison between tools can be done correctly and security officers can choose appropriate tools for their web application.

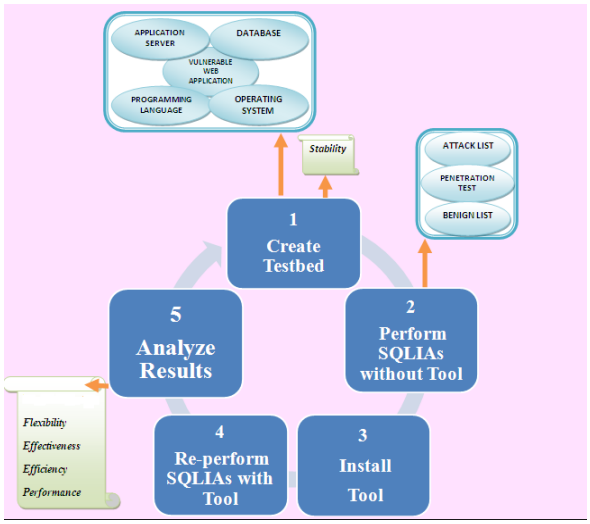

Figure4. Proposed Evaluation Framework

6.CONCLUSION

In this paper we presented how SQL injection attacks disclose the web application security and explained all SQL injection attack types. Then we compared the tools in terms of their ability to stop SQLIA, deployment requirement (modifying source code, additional infrastructure and automation of detection or prevention) and common evaluation parameters (efficiency, effectiveness, stability, flexibility and performance).

Moreover, this paper emphasises that the value of some parameters are dependent on test bed, so, for a complete evaluation, common criteria in a common test bed should be considered to prove the strength and weakness of SQL injection detection or prevention tools. So we proposed a framework for empirically evaluation of these tools because we believe that analytically comparison is not enough for choosing a tool.

TABLE 3. COMPARISON OF TOOLS BASED ON EVALUATION PARAMETERS

No	Tool	Efficiency	Effectiveness	Flexibility	Performance (ms)	Stability	
						Programming Language	Equipment for evaluation
1	AMNESIA [16]	0	100	All	Negligible	JAVA	N/A
2	IDS [6]	F.P=0.06 F.N=N/A	N/A	All /P	0.5	All	Server: 2 GHz Pentium 4 with 1 GB of RAM Linux 2.6.1. Apache web server (v2.0.52), the MySQL database (v4.1.8), and PHP-Nuke (v7.5).
3	JDBC Checker [12]	F.P=low F.N=N/A	N/A	All /p	Negligible	Java /JDBC	N/A
4	Securify [14]	F.P=N/A F.N=0	100	All	14.4	Java	Client: AMD Opteron 150 machine with 4GB of memory running Linux Server: 2 GHz AMD Athlon XP with 256MB of memory running Linux
5	SQLCHECK [5]	0	100	6	2	All	Linux kernel 2.4.27,2 GHz Pentium M processor and 1 GB of memory
6	SQLGaurd [10]	N/A	N/A	6	3	J2EE	The web server is a 733MHz windows 2000 machine, 256MB RAM
7	WAVES [18]	F.P=N/A F.N=2.6	N/A	All/P	N/A	ASP/PHP	Unix
8	WEBSSARY [15]	F.P=10.3	N/A	All	N/A	All	N/A
9	CANDID [7]	F.P=0 F.N=N/A	100	All	12	Java	Client: 2GHz Pentium processor and 2GB of RAM,Server: a Red Hat Enterprise GNU / Linux machine.
10	SQL_IDS [8]	0	100	All	5	All	Client: AMD Athlon 1GHz, with 256 MB RAM and Microsoft Windows 2000. Server : Apache Tomcat (ver. 5.5.23) and Microsoft SQL Server 2000
11	Swaddler [3]	F.P=low F.N=0	N/A	All /P	N/A	All	Client: 3.6GHz Pentium 4 with 2 GB of RAM running Linux 2.6.18. Server: Apache web server (version 2.2.4) and PHP version 5.2.1
12	Positive Tainting [1]	0	100	All	6	Java	Client: Pentium 4, 2.4Ghz, with 1GB memory. Server: a dual-processor Pentium D, 3.0Ghz, with 2GB of memory, running GNU/Linux 2.6
13	SQLPrevent [11]	F.P=0 F.N=0	100	All	3	All	Client: a 1.8 GHz Intel Pentium 4, 512 MB RAM, Windows XP SP2.Host: 350 Mhz Pentium II processor and 256 MB of memory, Windows 2003 SP2.

TABLE 4. PHASES OF EVALUATION FRAMEWORK

	PHASE	COMPONENT	OUTPUT	MEASURED PARAMETERS
1	**Create Testbed**	-Vulnerable web Application - Applications Server - Programming Language - Database - Operating System	A testbed made up of a vulnerable web application running on a configured network	-Stability → Environment Independence
2	**Perform SQL Injection without Tool**	- Penetration Test -Attack List - Benign List	Web application successful penetrated	
3	**Install Tool**	-Secure tool	Secure web application	
4	**Re-perform SQLIAs with Tool**	- Penetration Test - Attack List - Benign List	Safe web application, not penetrated anymore	-Flexibility → Types of SQLIAs -Efficiency →(False positive, False Negative) -Effectiveness → (Attacks Detection/ Prevention) - Performance → (Detection Overhead, Prevention Overhead)
5	**-Analyze Result -Change Parameters and Loop**		Results ,comments, statistics on measure parameters	

REFERENCES

[1] W. G. Halfond, A. Orso, Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks, 14th ACM SIGSOFT international symposium on Foundations of software engineering 2006, ACM.

[2] Saeed Sharifian, Seyed Ahmad Motamedi, and Mohammad Kazem Akbari, "Estimation-Based Load-Balancing with Admission Control for Cluster Web Servers," ETRI Journal, vol.31, no.2, 2009, pp.173-181.

[3] M. Cova, D. Balzarotti. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications. Recent Advances in Intrusion Detection, Proceedings, Volume: 4637, pp: 63-86, 2007.

[4] Jong-Hyouk Lee, Seon-Gyoung Sohn, Beom-Hwan Chang, and Tai-Myoung Chung, "PKG-VUL: Security Vulnerability Evaluation and Patch Framework for Package-Based Systems," ETRI Journal, vol.31, no.5, Oct. 2009, pp.554-564.

[5] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. ACM SIGPLAN Notices. Volume: 41, pp: 372-382, 2006.

[6] F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach to the Detection of SQL Attacks. Detection of Intrusions And Malware, And Vulnerability Assessment, Proceedings, Volume: 3548, pp: 123-140, 2005.

[7] P. Bisht, P. Madhusudan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Transactions on Information and System Security Volume: 13, Issue: 2, 2010.

[8] K. Konstantinos and T. Tzouramanis. SQL-IDS: A Specification-based Approach for SQL Injection Detection. Symposium on Applied Computing. USA: ACM, 2008.

[9] A. S. Christensen, A. Moller, and M. I. Schwartzbach. Precise Analysis of String Expressions. In Proc. 10th International Static Analysis Symposium, SAS '03, volume 2694, pp 1-18. Springer-Verlag, June 2003.

[10] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), 2005.

[11] F.Monticelli, PhD SQLPrevent thesis. University of British Columbia (UBC) Vancouver, Canada.2008.

[12] C. Gould, Z. Su, and P. Devanbu. JDBC Checker:A Static Analysis Tool for SQL/JDBC Applications. In Proceedings of the 26th International Conference on Software Engineering Formal Demos, 2004.

[13] Wassermann, G; Gould, C; Su, Z, et al. Static Checking of Dynamically Generated Queries in Database Applications. ACM Transactions on Software Engineering and Methodology. Volume: 16, Issue: 4, 2007.

[14] M. Martin, B. Livshits, and M. S. Lam. Finding Application Errors and Security Flaws Using PQL: A Program Query Language. ACM SIGPLAN Notices, Volume: 40, Issue: 10, pp: 365-383, 2005.

[15] Y. 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 Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[16] W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In Proceedings of the IEEE and ACM International Conference on Automated Software Engineering , USA, Nov 2005.

[17] W. G. Halfond and A. Orso. Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), USA, May 2005.

[18] Y. Huang, S. Huang, T. Lin, and C. Tsai. A Testing Framework for Web Application Security Assessment. Computer Networks, Volume: 48 Issue: 5, Pp: 739-761, 2005.

[19] CSI Computer Security Institute, The Computer Crime and Security Survey 2007.

[20] A.Tajpour, Suhaimi Ibrahim, M. Masrom, *"SQL Injection Detection and Prevention Techniques*, International Journal of Advancements in Computing Technology, Volume 3, Pp: 82-92, 2011.

[21] Huang, Y. u., Hang, F. C., Tsai, H . C., Lee, D. T., and Kuo, S. Y. Securing Web Application Code by Static Analysis and Runtime Protection. In Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[22] NavigiliI, R., Velardi, P. Quantitative and Qualitative valuation of the OntoLearn Ontology Learning System. Proceedings of the 20th international conference on Computational Linguistics, ACM, 2004.

[23] D. Scott and R. Sharp. Abstracting Application-level Web Security. IEEE Transactions on Knowledge and Data Engineering, Volume: 15, Issue: 4, 2003.

[24] V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. ACM SIGPLAN Notices, Volume: 40, Issue: 10, pp: 365-383, 2005.

[25] Atefeh Tajpour, Suhaimi Ibrahim, Maslin Masrom, "Evaluation of SQL Injection Detection and Prevention Techniques". International Journal of Advancements in Computing Technology (IJACT),2011, Kore.

[26] Daswani, N., Kern, C. and Kesavan, A. (2007). Foundations of Security. Apress.

[27] S. Bandhakavi, P. Bisht, P. Madhusudan, CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, 2007, USA, ACM.

[28] W. G. Halfond, J. Viegas and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," College of Computing Georgia Institute of Technology IEEE, 2006.

[29] Atefeh Tajpour, Suhaimi Ibrahim, Mohammad Sharifi, "Web Application Security by SQL Injection Detection Tools". International Journal of Computer Science Issues (IJCSI), 2012.

Atefeh Tajpour She received her B.Sc.degree in Computer Engineering from Iran University of Science and Technology in 1995 and her M.Sc. degree in Information Security from University Technology Malaysia in 2010. She has more than 12 years experience in application programming and system analysis in Iran. She is currently working toward the PhD degree in computer science in University Technology Malaysia. Her interest is in web application security. She has published different articles in international conference and journals that have been indexed by IEEE, Computer Society, Elsevier, Scopus and ISI. She is also a reviewer of IEEE international conferences as well as a member of editorial board of International Journal of Advanced Computer Research.

Suhaimi Ibrahim received his B.Sc. degree in Computer Science (1986), his M.Sc. in Computer Science (1990), and his Ph.D. in Computer Science (2006). He is an Associate Professor attached to Dept. of Software Engineering, Advanced Informatics School (AIS), University Technology Malaysia International Campus, Kuala Lumpur. He currently holds the post of Deputy Dean of AIS. He is an ISTQB certified tester and has been appointed a board member of the Malaysian Software Testing Board (MSTB). He has published many articles in international conferences and international journals such as the International Journal of Web Services Practices, Journal of Computer Science, International Journal of Computational Science, Journal of Systems and Software, and Journal of Information and Software Technology. His research interests include software testing, requirements engineering, web services, software process improvement, mobile and trusted computing.