Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 4 (2012) 790 – 796

Procedia Technology

C3IT-2012

Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks

Kanchana Natarajan[a], Sarala Subramani[b]

[a,b]*Department of IT, School of Computer Science & Engineering, Bharathiar University, Coimbatore-641046, Tamilnadu, India*

Abstract

Security and privacy of database-driven web applications are extremely multifaceted against web intruders. One of the most dangerous cyber attacks is the SQL-injection attack, which simply creates huge loss to commercial vendors. Research deliberates to provide SQL-injection free (SQL-IF) secure algorithm to detect and prevent SQL-injection attacks (SQLIAs). In this paper, we have re-addressed several detection methods to conflict against the proposed SQL-IF secure algorithm. The generated algorithm has been integrated into the runtime environment while the implementation has been done through Java. The algorithm describes the method that how we follow the procedures for preventing SQL-injection attacks. We presented the SQL-IF secure algorithm and logic of the generated code. Comparison of similar types of attack along with different features is performed. The empirical results and its evaluation prove that the algorithm works efficiently to detect the SQLIAs.

Keywords: SQL-injection attack; SQL query; Dynamic method; SQL-injection free algorithm; Runtime environment;

1. Introduction

Software has pervaded all over the world from past two decades and faces many fascinating challenges. Web applications have become obligatory in humans day-to-day life while some of the frequently used functional web applications such as online banking, web mail, online auctions, online sales retails, social networks and blogs are the foremost targeted spots for the human web attackers. Web vulnerabilities have made tremendous growth in web applications whereas the web developers fail to meet global standards of designing framework and writing programming code. It is necessary to perform proper input sanitization, syntax validation and follow the security guidelines to secure for prevention of the major loopholes during the programming phase.

Many commercial and open source tools exist in market with specialized features even though researchers have analyzed and proved not even a single detection scanner provides best result for all the

doi:10.1016/j.protcy.2012.05.129

categories of vulnerability. It is highly challengeable task for security-oriented developers to build reliable tools that provide easier approach to handle the security issues. Vulnerability detection scanners are highly intense, used most often among large organizations as they not detect potential vulnerability. Some scanners are not detecting stored vulnerabilities and others are very particular to detect Cross Site Scripting (XSS), Cross Channel Scripting (XCS), Information Leakage, etc. even with some major limitations [1, 2, 3]. The research concentrates by taking SQL injection vulnerability which is one of the common critical attacks on web applications. An existing study performed in 2007 shows the result of about 70% of database-centric web applications are under high risk being hacked by the attackers; eventually most of them are SQL injection attacks [6, 7 and 10].

1.1 Background of the SQL Injection Attacks (SQLIAs)

SQL injection attack (code injection) is the most common and easiest type of vulnerability technique adopted by the web attackers through data-driven web applications. By using simple SQL commands such as Select, Where, Insert, Delete and Update, the malicious attackers efficiently re-structure the actual SQL code (statements) and executes vulnerable code into the web applications. Once nasty attacker attains their goal they can easily access sensitive information, modify secured data, executes the data, and even they may collapse the entire application. Since the privacy of the database administrator loses their role by unauthorized accesses of malicious. SQL injection attacks are more lucrative for attackers as they mainly focus to stolen bank account, credit card numbers, etc. This type of security issues on web applications is more susceptible, can be handled by the authentication of users. Many forms of SQL injection attacks exist. Most common takes the benefits of erroneously passed parameters, erroneous type handling, erroneous use of SQL statements, for e.g. (‘ OR) 1 = -- ’). Various types of SQL injection attacks are available such as tautologies, illegal/logically incorrect queries, UNION query, Piggy-backed queries, Stored Procedures, Blind SQL, Timing Attack, Alternate Encoding and etc. Defeating these types of attacks is not simple since the attacker actually changes the behaviour of predefined SQL queries [5].

1.2 Methods to Detect and Prevent SQLIAs

Many research authors explored a number of methods to detect and prevent SQLIAs; the most chosen techniques are static analysis, dynamic analysis, combined static and dynamic analysis, web framework, defensive programming and machine learning techniques. The method of *static analysis* are extreme were it analyzes the code for vulnerability by without actually executing the code. Software metrics and reverse engineering are some forms of static analysis. Model checking, data flow analysis, abstract interpretation and use of assertions in source code are the several techniques of static code analysis. The method of *dynamic analysis* can be performed automatically by the analysis of vulnerabilities during the execution of web applications which avoids thousands of tests by doing several times manually. Example: CANDID tool. Both the techniques have merits and demerits and therefore variations are identified from the efficacy. However the research study analyzed with various existing works and it has been proved dynamic analysis (*penetration testing*) tool is effective to test the web applications [1, 4, 8]. Penetration testing tools are easy to use and assure to provide security information systems to their users by fixing the security weaknesses before they get exposed. The major advantages of penetration (dynamic) testing are: (a) Not necessary to change the development lifecycle (b) Avoids static analysis challenges (c) No need for the source code, (d) Deployment-security.

The method of combined *static and dynamic analysis* can compensate the limitations of each method, which is considered as highly proficient against SQLIAs but it is very complicated. One of the best examples for such a method is AMNESIA tool. It uses static analysis to analyze the web-application code and automatically build a model of the legitimate queries that the application can generate. At runtime, the technique monitors all dynamically-generated queries and checks them for compliance with the

statically-generated model. When the technique detects a query that violates the model, it classifies the query as an attack, prevents it from accessing the database, and logs the attack information. The *web framework* method is a filtering method of user input parameters. This method is proven to be in-effective while it is not able filter some special characters. The *machine-learning* method is the most commonly used method whereas the method results in high false positives and low detection rate. Example: WAVES tool [1, 5, 9, 11 and 12].

2. SQL-Injection Free (SQL-IF) Secure Algorithm

The newly proposed algorithm is based on dynamic technique which violates over SQL injection attacks. This algorithm concentrated to develop IF (Injection Free) attacks; where as a special type of test suite is developed to detect SQL injection attacks. Systematic flow diagram of this proposed approach is shown in Fig 1 which depicts the SQL-IF secure algorithmic work flow.

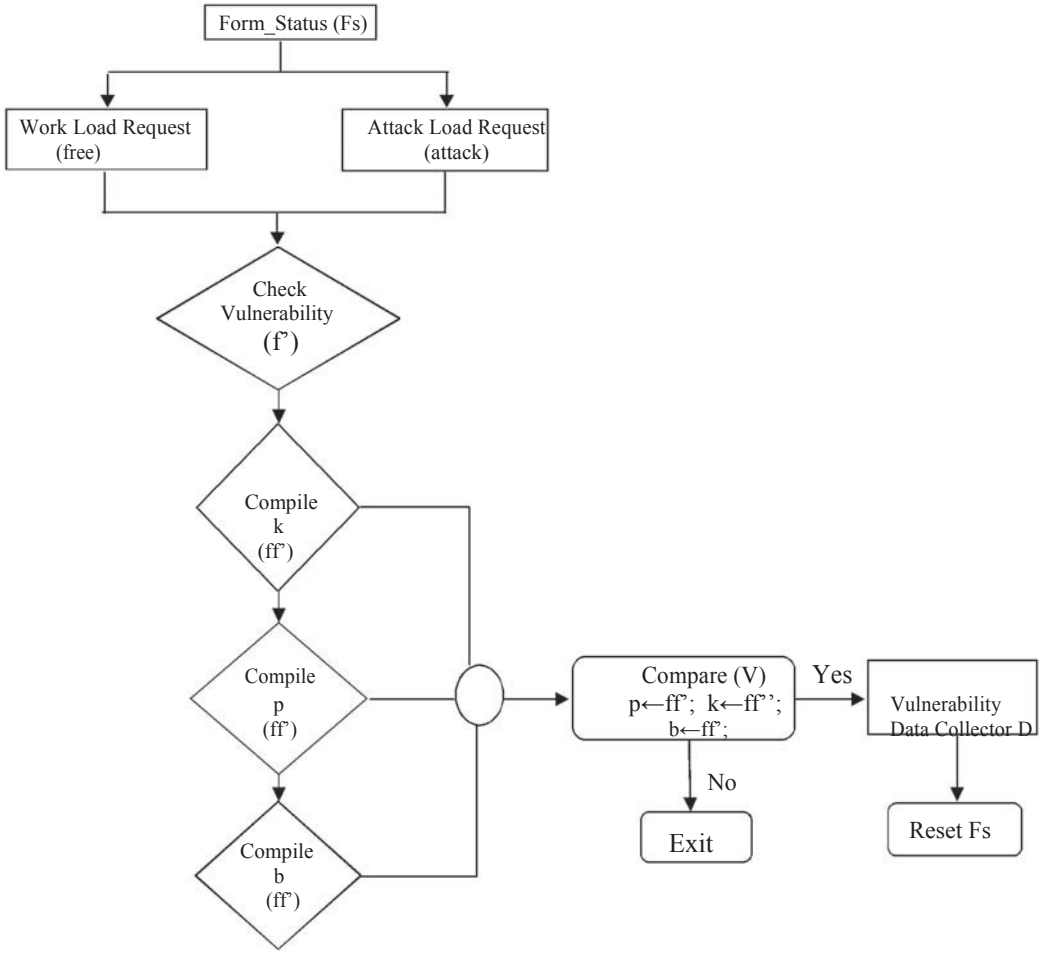

Fig 1: Systematic control flow diagram of proposed SQL-IF approach

The below shown Algorithm1 performs its function by assigning the Form Status *(FS)* as attack and free with collection of fields obtained from the collection of forms. The Form *(fm')* is obtained from the

set of forms *(Fm)* whereas values of every field are obtained from the form *(Fm')*. Three well-defined functions are generated inside the method called CheckVulnerability (f') to check for any special characters, keywords and Boolean characters.

Algorithm 1: Proposed SQL-IF Secure Algorithm

```
Algorithm SQLIAD (Form F)
01  Input: Fm denotes the collection of forms with collection of Fields.
02  Input: Enumerate Form_Status FS = {attack, free}
03  Input: Default Value to FS as free
04  Output: FS
05          for each fm' ∑ Fm do
06          for each f' ∑ fm' do
07                  f' ← fields contains the values
08          if f' is not empty string then
09                  FS ← output from the method CheckVulnerability (f')
10          if FS as attack
11                  D ← Add the field f' in the collection
12      Reset the Http requests to issue warning;
13  return FS;
```

Generic detection method 1: Generated to check for Special characters, keywords and Boolean keywords

```
CheckVulnerability (f'):

    for each f' fields do
        if f' is a non-empty string then
            // to check for special characters in the input fields and parameters
        p = collection of compiled special characters like {(['',&+=<><=>=])}
        for each tokens ff' ∑ f' then
            ff'' = compile ff' to make all the input tokens neutralize.
                v = comparison of P and ff';
         if v is not true then return v;
            // to check for keywords in the input fields and parameters
      k = collection of keywords {union|select|intersect|insert|update|delete|drop|truncate}
            for each tokens ff' ∑ f' then
            ff'' = compile ff' to make all the input tokens neutralize.
                v = compare ff'' with k;
        if v is not true then   return v;
            // to check for Boolean characters in the input fields and parameters
         b = collection of Boolean characters {' or '|'or'|'AND'|'and'"}
            for each tokens ff' ∑ f' then
            ff'' = compile ff' to make all the input tokens neutralize.
                v = compare b with ff';
    return v;
```

The detection method is elaborated by given Generic detection method1, *CheckVulnerability (f')* which checks for the special characters, keywords and Boolean characters in the input fields. It compiles a collection of special characters like {(['',&+=<><=>=])} and a collection of keywords like union, select, intersect, insert, update, delete, drop, truncate and Boolean characters like 'or '|'or', 'AND',' and with the actual input parameters to make all the input values neutralized to the database. When the mismatch is to be found in the parametric values it is directly send to the vulnerability data collector and resets the Http request to warnings. The algorithm detects the SQLIAs which can be applied for any real web-based applications wherever the user and the database interacts. The detection algorithm is implemented through Java platform which is feasible to secure applications against SQLIAs.

3. Implementation and Evaluation

The proposed algorithm proves the detection rate can be defined from the number of attacks versus number of detection. It is not possible to detect the attack in real time also dynamic analysis method is not at a best solution for the detection and prevention of SQL injection attacks. It is suitable for finding the vulnerabilities in the web applications.

3.1 Experimental results

The implementation of the proposed SQL-injection free technique has been done under Java Programming Language and Structured Query Language. The proposed technique is generic in any web applications and it is language independent, can be implemented in any programming language. Source code analysis plays a major role in the detection of SQL injection attacks. At the initial stage, we implemented the algorithm in the real time web application and the results obtained are positive. Since the web application is free from SQLIAs and further analyses with vulnerable web application is required. Fig 2: shows the results of vulnerability results during the initial stage.

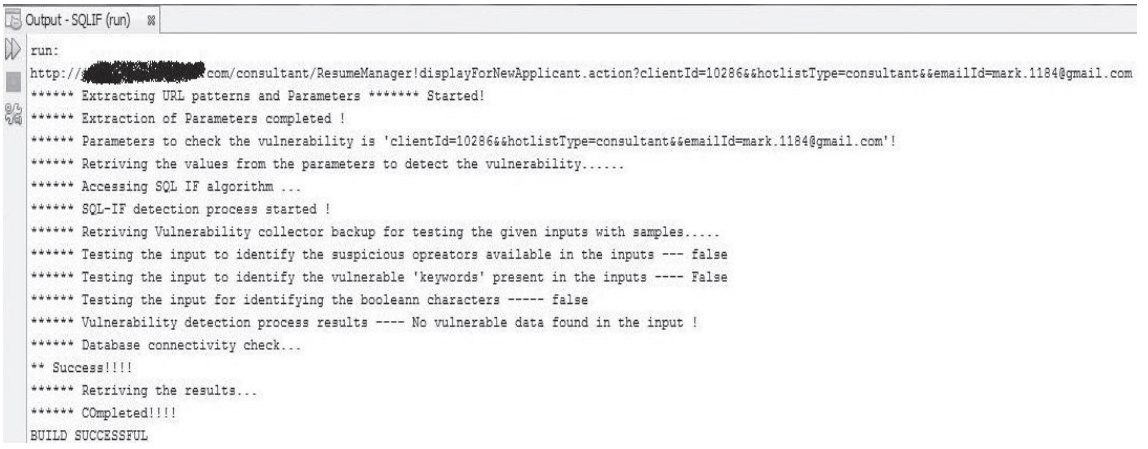

Fig 2: Results from the real time application with no vulnerability

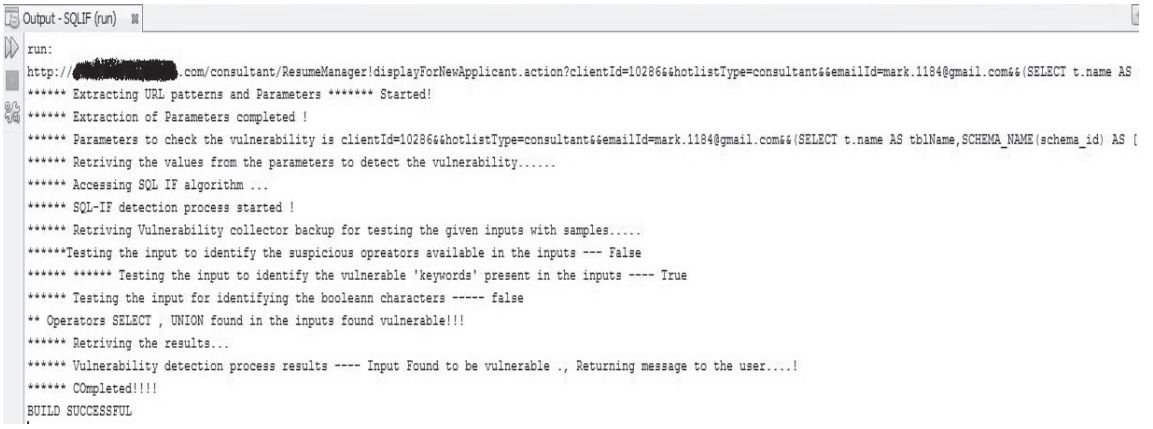

Fig 3: Results from the real time application with SQL-IF vulnerability detection

At the next stage the proposed generic code has been tested with vulnerable web application and the results obtained are negative. Hence the proposed algorithm filtered the Union Queries and some special

keywords which are found to be vulnerable input to the database. Fig 3: shows the results with vulnerable SQL query and detection of the same.

3.2 Advantages

The main advantage is no need for further code modification and provides optimized runtime analysis. Our technique uses the same existing approach and major difference lies with source code analysis and procedures followed to detect SQL-injection attacks. Proposed algorithm is implemented with simple SQL queries and the experimental result provides security guarantees.

3.3 Comparison of detection and prevention methods by various SQL injection attacks

The PSR-Algorithm and Tautology-checker use the static analysis method. The efficiency of these methods differs by static analysis of SQL queries and does not integrated into the runtime environment. Novel method and Safe Query Objects use both the static and dynamic method but fails to detect stored procedure type attacks. The proposed algorithm in this paper does not use any complex analysis techniques while the methodology differs by search methods for vulnerability. The comparison results are shown in Table 1.

Table 1
Comparison of detection and prevention methods with various features and attack types

Detection/Prevention Methods	Static Analyzer	Dynamic Analyzer	Detailed Output Info	Illegal Queries	Piggy-backed Queries	Stored procedures	Union Queries	Real Time
PSR – Algorithm [1]	√	x	x	x	x	x	x	x
Novel Method [5]	√	√	x	x	x	x	x	√
Safe Query Objects [5]	√	√	x	√	√	x	√	x
Tautology-checker [5]	√	x	x	x	x	x	x	x
Web App. Hardening [5]	x	x	x	√	√	x	x	x
Proposed SQL-IF	x	√	√	√	x	x	√	√

4. Conclusion and Future Work

The proposed generic algorithm is substantial in scrutiny of its simple detection mechanism against SQL injection attacks. Testing of web applications for SQL injection attack is a significant step for ensuring its performance and quality. The proposed algorithm performs much faster and endowed with proficient solution to resolve against SQL injection attacks. The paper work has analyzed with various detection methods and the proposed method cannot only be implemented on web applications also can be used on any applications which interacts towards databases.

The future research will be considerate to construct SQL parser. Generation of parser to detect critical vulnerabilities is another one complex approach. Also dynamic checking complier can be designed to harden the web applications in three-tier internet services for protecting from SQL Injection attacks (SQLIAs). Both the approaches were quite feasible to achieve effectiveness and efficiency.

References

1. Stephen Thomas, Laurie Williams, Tao Xie, On automated prepared statement generation to remove SQL injection vulnerabilities, *Journal of Information and Software Technology*, *Elsevier Ltd*, 2009, pages: 589-598.
2. Abdul Bashah Mat Ali , Ala' Yaseen Ibrahim Shakhatrehb, Mohd Syazwan Abdullahc, Jasem Alostadd, SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks, *Journal of Procedia Computer Science, Elsevier Ltd, 2010, pages: 453-458.*
3. Dimitris Mitropoulos, Diomidis Spinellis, SDriver: Location-specific signatures prevent SQL injection attacks, *Journal of Computers & Security, Elsevier Ltd,* 2009, pages: 121-129.
4. Joa˜o Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, and Rui Neves, Vulnerability Discovery with Attack Injection, *IEEE Transactions on Software Engineering,* 2010, Vol. 36, pages: 357- 370.
5. Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values, *Journal of Mathematical and Computer Modeling, Elsevier Ltd*, 2011, pages: 1-11.
6. Shaukat Ali, Azhar Rauf, Huma Javed, SQLIPA: An Authentication Mechanism against SQL Injection, *European Journal of Scientific Research,* 2009, Vol.38, pages: 604-611.
7. Ivano Alessandro Elia, José Fonseca, Marco Vieira, Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study, *21st IEEE International Symposium on Software Reliability Engineering*, 2010, pages: 289-298.
8. J. Park, B. Noh, SQL injection attack detection: profiling of web application parameter using the sequence pairwise alignment, *Journal of Information Security Applications, LNCS*, 2007, vol. 4298, pages: 74-82.
9. Z. Su, G. Wassermann, The essence of command injection attacks in web applications, *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Charleston, SC, USA, 2006, pages: 372–382.
10. W.G.J. Halfond, A. Orso, P. Manolios, WASP: protecting web applications using positive tainting and syntax-aware evaluation, *IEEE Transactions on Software Engineering,* 2008, vol. 34 (1), pages: 65–81.
11. MeiJunjin, An approach for SQL injection vulnerability detection, *IEEE Sixth International Conference on Information Technology: New Generations*, pages: 1411-1414, 2009.
12. Lijiu Zhang, Qing Gu, Shushen Peng, Xiang Chen, Haigang Zhao, Daoxu Chen, "D-WAV: A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms", *IEEE Fifth International Conference on Software Engineering Advances*, pages: 501-507, 2010.