Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Efficient Method to Secure Web applications and Databases against SQL Injection Attacks

Ms. Zeinab Raveshi
Pursuing M.Tech (IT),
Bharati vidyapeeth deemed University
College of engineering, Pune, India.

Mrs. Sonali R. Idate
Education: M.E.
Bharati Vidyapeeth Deemed University
College of Engineering , Pune, India.

Abstract— SQL injection is a common technique for attackers using SQL queries to attack on Web-based applications. These attacks reshape SQL queries and thus alter the behavior of the program for the benefit of the hacker. SQL Injection attacks are one of the gravest threats for web applications. Web applications are becoming an important part of our daily life. So attacks against them also increases rapidly. Of these attacks, a major role is held by SQL injection attacks (SQLIA). In this project proposes a new method for preventing SQL injection attacks in JSP web applications. The basic idea is to check before execution, the intended structure of the SQL query. For this we use semantic comparison. Our focus is on stored procedure attack in which query will be formed within the database itself and so difficult to extract that query structure for validation. Also this attack is less considered in the literature.In addition to these schemes we are using the new approach in order to provide security in depth such as least Privilege and white List Input Validation. To minimize the potential damage of a successful SQL injection attack, one should minimize the privileges assigned to every database account in their environment. Do not assign DBA or admin type access rights to one's application accounts. It is always recommended to prevent attacks before the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is passed to the SQL query.

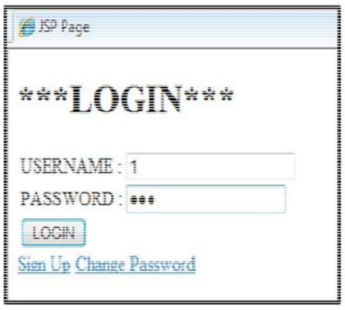
Keywords— SQL, Web Applications, SQLIA, Query, hacker, vulnerability identification, attack prevention.

I. INTRODUCTION

Of many kinds of attacks against web applications, SQL Injection Attack (SQLIA) is one of the top most threats against them [12]. So it is highly requires in the current scenario to have a good solution to prevent such attack to secure the information. This is the motivation behind this work. SQL Injection targets the web applications that use a back end database. Working of a typical web application is as follows: User is giving request through web browsers, which may be some parameters like username, password, account number etc. These are then passed to the web application program where some dynamic SQL queries are generated to retrieve required data from the back end database. SQL Injection attack is launched through specially crafted user inputs. That is attackers are allowed to give requests as normal users. Then they intentionally create some bad input patterns which are passed to the web application code. If the application is vulnerable to SQLIA, then this specially created input will change the intended structure of the SQL query that is being executed on the back end database and will affect the security of information stored in the database. The tendency to change the query structure is the most characteristics feature of SQLIA which is being used for its prevention also. For example, figure 1 showing the normal case whereas figure 2 shows an attack scenario. That is an attacker wants to log in without correct username and password. Instead of entering valid username if he uses injection string like "hacker' OR '1'='1'—" as username and "something" as password, the query formed will be like this:

*Select * from login where user='hacker' or '1'='1' –' and pass='something'*

When this query is executed in the database, it will always return a true and the authentication will succeed.


JSP Page
LOGIN
USERNAME : 1
PASSWORD : ●●●
LOGIN
Sign Up Change Password

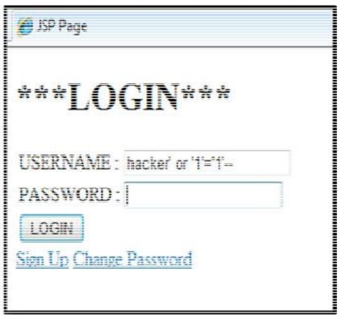

JSP Page
LOGIN
USERNAME : hacker' or '1'='1'--
PASSWORD :
LOGIN
Sign Up Change Password

Figure 1: Example login – Normal case Figure,
Figure 2: Example login – attack case

In this paper we are discussing the previous methods along with present the new method which will efficiently finds out not only various kinds of SQL inject attacks, but also prevent them. Below next section II presents the existing methods and their limitations.

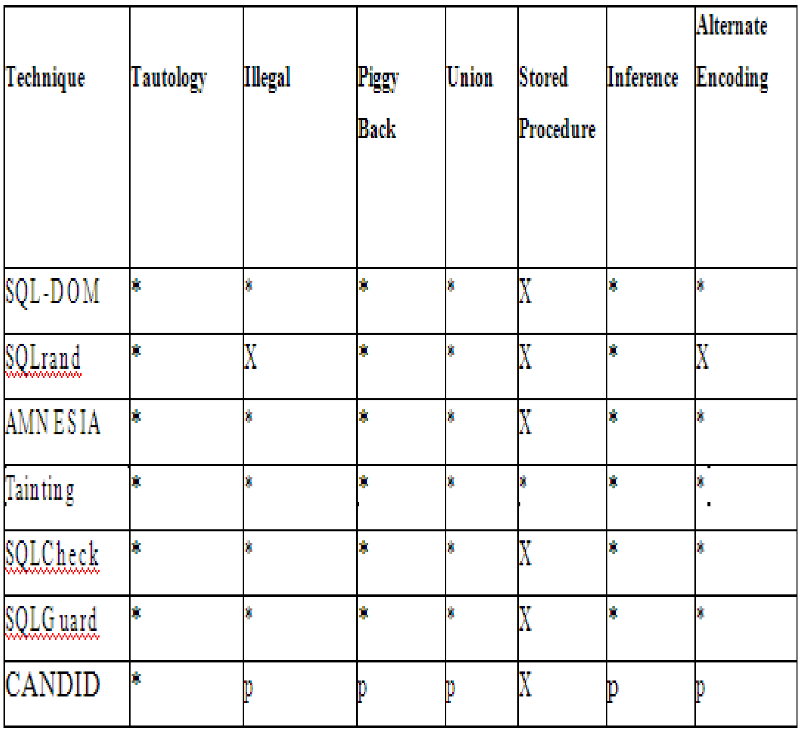
II. EXISTING METHODS AND LIMITATIONS

Research on SQL injection attacks can be broadly classified into two basic categories: *vulnerability identification* approaches and *attack prevention* approaches. The former category consists of techniques that identify vulnerable locations in a Web application that may lead to SQL injection attacks. In order to avoid SQL injection attacks, a programmer often subjects all inputs to input validation and filtering routines that detects attempts to inject SQL commands. The techniques presented in [3, 4, 13] represent the prominent static analysis techniques for vulnerability identification, where code is analyzed to ensure that every piece of input is subject to an input validation check before being incorporated into a query (blocks of code that validate input are manually annotated by the user). While these static analysis approaches scale well and detect vulnerabilities, their use in addressing the SQL injection problem is limited to merely identifying potentially unvalidated inputs. The tools do not provide any way to check the correctness of the input validation routines, and programs using incomplete input validation routines may indeed pass these checks and cause SQL injection attacks. Another approach to solve the problem is provided by the class of attack prevention techniques that *retrofit* programs to shield them against SQL injection attacks [5, 6, 7, 8, 9, 10, 11]. These techniques often require little manual annotation, and instead of detecting vulnerabilities in programs, they offer preventive mechanisms that solve the problem of defending the Web application against SQL injection attacks. Relying on input validation routines as the sole mechanism for SQL injection defense is problematic. Although they can serve as a first level of defense, they cannot defend against sophisticated attack techniques (e.g., those that use alternate encodings and database commands to dynamically construct strings) that inject malicious inputs into SQL queries. A more fundamental technique to solve the problem of preventing SQL injection comes from the commercial database world in the form of PREPARE statements. These statements, originally created for the purpose of making SQL queries more efficient, have an important security benefit. They allow a programmer to declare (and finalize) the structure of every SQL query in the application. Once issued, these statements do not allow malformed inputs to influence the SQL query structure, thereby avoiding SQL injection vulnerabilities altogether. The following statement:

SELECT * FROM phonebook WHERE username = ? AND password = ?

is an example of a PREPARE statement. The question marks in the statement are used as "place-holders" for user inputs during query parsing and, therefore, ensure that these possibly malicious inputs are prevented from influencing the structure of the SQL statement. Thus, PREPARE statements allow a programmer to easily isolate and confine the "data" portions of the SQL query from its "code." Thus, PREPARE statements are in fact a robust and effective mechanism to defend against SQL injection attacks. However, retrofitting an application to make use of PREPARE statements requires manual effort in specifying the intended query at every query point, and the effort required is proportional to the complexity of the Web application.

From this comparison showing in table 1 next, it is clear that stored procedure attacks are less considered in the literature. Hence in this project we are focusing over particular kind of attacks along with general prevention.



Technique	Tautology	Illegal	Piggy Back	Union	Stored Procedure	Inference	Alternate Encoding
SQL-DOM	*	*	*	*	X	*	*
SQLrand	*	X	*	*	X	*	X
AMNESIA	*	*	*	*	X	*	*
Tainting	*	*	*	*	*	*	*
SQLCheck	*	*	*	*	X	*	*
SQLGuard	*	*	*	*	X	*	*
CANDID	*	p	p	p	X	p	p

Table 1: Comparative Analysis of Existing Methods

*-Prevention
P-Partial prevention X-Prevention not possible

III. RELATED WORK

Various techniques have been proposed for preventing SQL injection attacks. In [18], SQLrand was proposed which uses instruction set randomization of SQL statements to check SQL injection attacks. It uses a proxy to append the key to SQL keywords. A derandomizing proxy then converts the randomized query into appropriate SQL queries for the database. The key is not known to the attacker, so the code injected by attacker is treated as undefined keywords and expressions which cause runtime exceptions, and the query is not sent to the database. The disadvantage of this system is its complex configuration and the security of the key. If the key is exposed, the attacker can formulate queries for a successful attack.

Halfond and Orso in [19] developed AMNESIA, which is a model-based technique that combines static and dynamic analysis. The tool first identifies hotspots where SQL queries are issued to database engines. At each hotspot, a query model is developed by using Non-Deterministic Finite Automata (NDFA). The hotspot is instrumented with monitor code, which matches the dynamically generated query against the query model.

If a generated query is not consumed by NDFA, then it is an attack. Su and Wassermann in [20] based their work on a formal definition of SQL injection attack. In their definition, SQL injection occurs when the intended syntactic structure of SQL queries is changed by tainted input. In order to check whether this policy is violated by a program, they track tainted input dynamically by enclosing it within randomly generated markers. When the program issues an SQL query, the markers indicate the points of the query that contain potentially malicious values.

Cova, Balzarotti et al. in [21] proposed an anomaly based approach for the detection of volition of web applications. They used “Swaddler” for the analysis of the internal state of web applications and to find the relationship between the critical points and internal state. Swaddler identified attacks that attempted to bring violation of the intended workflow of a web application.

Cristian Pinzón in [22] designed an agent for the detection and prevention of SQL injection queries at the database layer of an application. The agent incorporates a case-based reasoning mechanism whose main characteristic involves a mixture of neural networks that carry out the task of filtering attacks. MeiJunjin in [23] proposed an approach for the detection of SQL injection loopholes. The author adopted static, dynamic and automatic testing methods for the detection of SQL injection loopholes. Their proposed approach traces user queries to vulnerable location.

Angelo Ciampa in [17] proposed an approach and a tool- named V1p3R (“viper”) for web-application penetration testing. The working of this approach is based on pattern matching of error message and on outputs produced by the application under testing; it relies upon an extensible knowledge-base consisting of a large set of templates.

IV. PROPOSED WORK

In this paper we are presenting the new approach for SQL Injection detection and prevention. This paper offers a technique, *dynamic query structure validation*, which automatically (and dynamically) mines programmer-intended query structures at each SQL query location, thus providing a robust solution to the retrofitting problem.

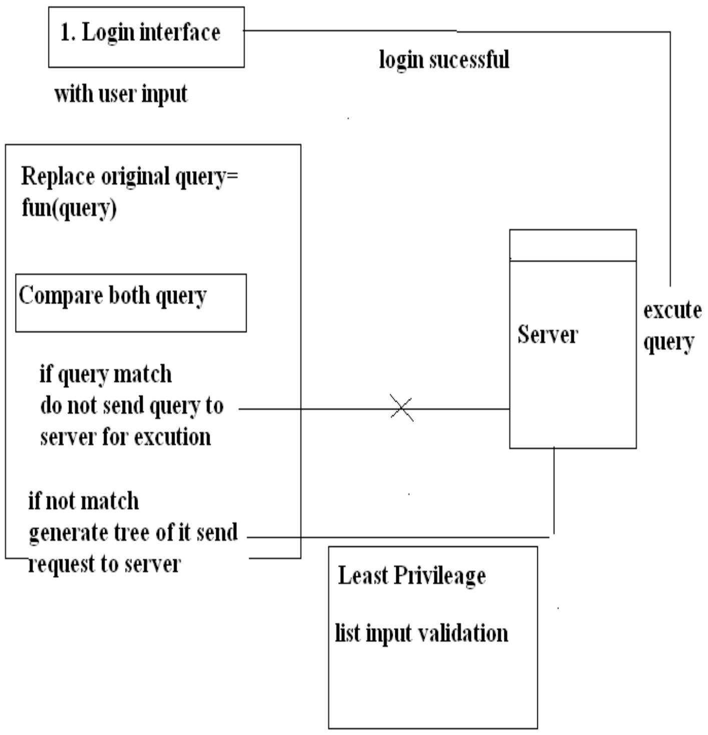


Figure 3: Proposed Framework

The idea is that the process of generation of queries in a dynamic web application can be represented as a function of user‘s inputs [2]. In this context, SQL injection is any situation in which the user‘s input is inducing an unexpected change in the output generated by the function. Two parameters can be defined:

Original_Query = Fun(input_i) i = 1 to n
input_i = input from user
Fun() = Function represented by web application
Benign_Query =Fun(input_benign_i) I = 1 to n
input_benign _i = "qqq" or any evidentlynon-attacking input

The idea requires that the application will not allow the user to enter any part of SQL query directly. Two statements are said to be semantically equivalent, if they perform similar activities, once they are executed on the database server. So if it can be determined that both Original_Query and Benign_Query are semantically equivalent, then there is no possibility of SQL injection. This paper uses this semantic comparison to detect SQL injection. The semantic comparison is done by parsing each of the statements and comparing the syntax tree structure. If the syntax trees of both the queries are equivalent, then the queries are inducing equivalent semantic actions on the database server, since the semantic actions are determined by the structure of the Original_Query.

Steps include:
1. Generate a *Benign_Query* from the *Original_Query* generated by the application. This is done by replacing user inputs to the query with benign inputs.
2. Check the syntax of the *Benign_Query* to ensure its validity while doing the replacement.
3. Get the count of stacked queries in both original SQL query and generated *Benign_Query*.
4. Compare the count of stacked queries. If both counts are different, then we can directly report SQL injection attack and prevent that query from execution without going for semantic checking.
5. Now construct a syntax tree of both *Original_Query* and *Benign_Query* and compare them. Here, syntax trees are created using java *ArrayList* structure.
6. Compare the syntax trees. If they are equal, the queries are valid and allow its execution. Otherwise, report injection and block the query.

Extension to Prevent Stored Procedure Attack

Stored procedures are an important part of relational databases. They add an extra layer of abstraction into the design of a software system. This extra layer hides some design secrets from the potentially malicious users, such as definitions of tables. By using stored procedures, one could make sure that all the data is always contained in the database and is never exposed. In these databases, the developer is allowed to build dynamic SQL queries ie. SQL statements are built at runtime according to the different user inputs. For example, in SQL Server, EXEC (varchar(n) @SQL) could execute arbitrary SQL statements. This feature offers flexibility to construct SQL statements according to different requirements, but faces a potential threat from SQL Injection Attacks.

Filter Techniques

We are using a filter (see Figure 3) in between the Web application server and database server to filter out the abnormal or bad SQL injection queries. If the username and password are making an SQL injection query then the filter will not pass the SQL injection query to the database server, and client side Web page will show that the username and password are invalid.

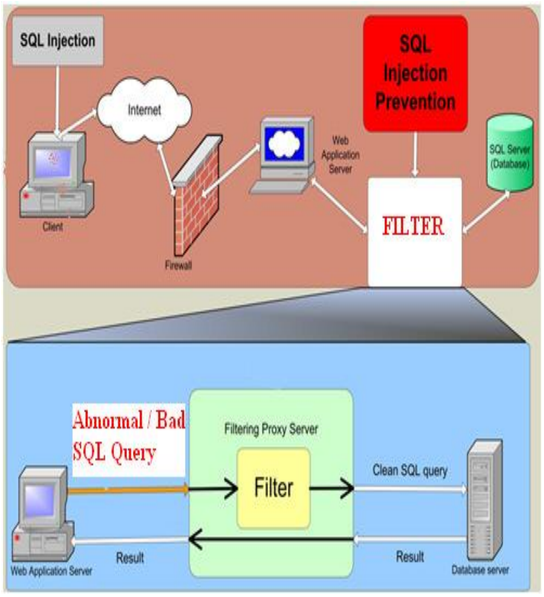


Figure 4: Filter Technique

If operators with equal, special characters or specific characters become SQL injection queries, then the filter will discard the username and password. A single operator cannot make an SQL injection query. In this case, the filter will allow only the single operator to become the username and password, and not the special and specific characters.

Filters can also check a single line comment, multiple line comments, and concatenate (double pipe) the fields of username and password. If any comment –among those that are discussed above – is present in the username and password, then the filter discards the entered values.

- **--** or # single-line comment
- /*…*/ multiple-line comment
- || concatenate (double pipe)

The Three Components

The three components in the architecture of our proposed technique are as follows:

a) User login interface
b) SQL injection protector for authentication
c) User account table

The user account table is used to store the users' account data. SQL Injection Protector for Authentication (SQLIPA) is the arc component of architecture (see Figure 4).

Customized Error Message

Error messages here refer to the blind SQL injection attack. The binge of informative error messages may accommodate the knowledge to access the database to the user. But it is a difficult task for debugging if we try to remove error messages completely. Customized error messages hinder the reconnaissance progress of threat agents, particularly in deducing specific details such as injectable parameters etc.

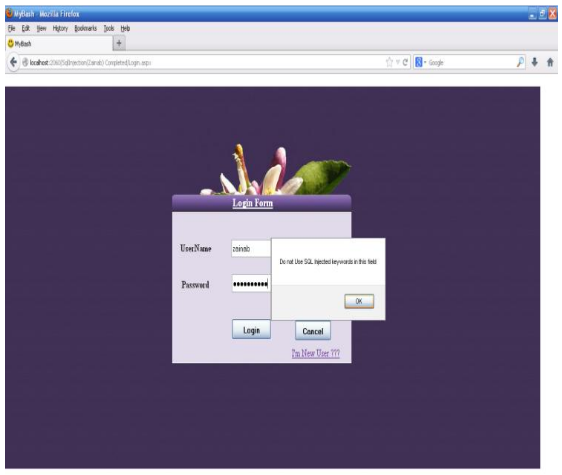
POST Method

For sending data to the server, the POST method is used. In this method, along the request object, the query string is appended, but not in URL. That is why transferable parameters are in the hidden form.

V. Implementation Results

Here we will show some results which will detect and prevent the SQL injection attacks.

1. Prevent attack on website



MyBash - Mozilla Firefox

localhost:2060/SqlInjection(Zainab) Completed/Login.aspx

Login Form

UserName: zainab

Password: ••••••••••

Do not Use SQL Injected keywords in this field

Figure 5: Preventing Attack From Website

2. Normal Login

Figure 6: Normal User Login

3. Perform System Operation

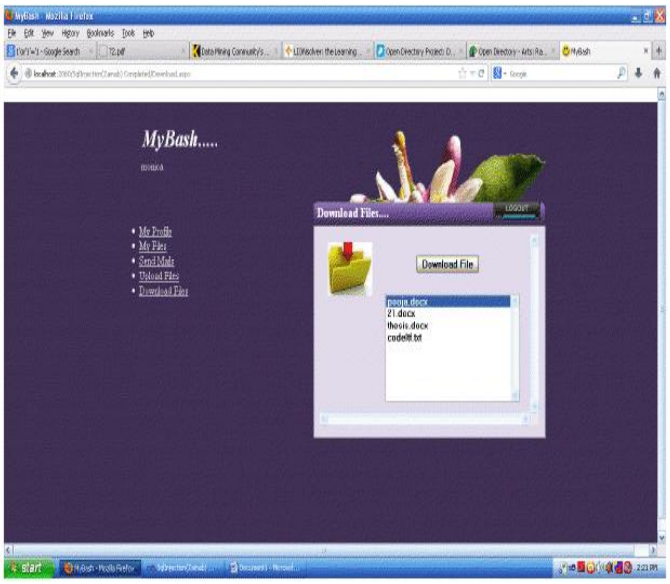


Figure 7: Operations of System

VI. Conclusions

SQL injection vulnerability is one of the top vulnerabilities present in the web applications. In this paper we proposed an efficient approach to prevent this vulnerability. Our solution is based on the principle of dynamic query structure validation which is done through checking query's semantics. It detects SQL injection by generating a benign query from the final SQL query generated by the application and the inputs from the users and then comparing the semantics of safe query and the SQL query. The main focus is on stored procedure attacks in which getting query structure before actual execution is difficult

References

[1] Halfond, W., Viegas, J., & Orso, A. (2006). "Classification of SQLInjection Attacks and Countermeasures." *SSSE 2006.*

[2] Sandeep Nair Narayanan, Alwyn Roshan Pais, & Radhesh Mohandas. Detection and Prevention of SQL Injection Attacks using Semantic Equivalence. Springer 2011

[3] Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM **.**Etienne Janot, Pavol Zavarsky Concordia University College of Alberta, Department of Information Systems Security

[4] Xie, Y., and Aiken, A. Static detection of security vulnerabilities in scripting languages. *In USENIX Security Symposium* (2006).

[5] Boyd, S. W., and Keromytis, A. D. Sqlrand: Preventing sql injection attacks. In *ACNS* (2004), pp. 292–302.

[6] Halfond, W., and Orso, A. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. *In ASE* (2005), pp. 174–183.

[7] Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., and Evans, D. Automatically hardening web applications using precise tainting. *In SEC* (2005), pp. 295–308.

[8] Buehrer, G., Weide, B. W., and Sivilotti, P. A. G. Using parse tree validation to prevent sql injection attacks. In SEM (2005).

[9] Prithvi Bisht, P. Madhusudan, V. N. VENKATAKRISHNAN. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. *ACMTransactions on Information and System Security,*Vol. 13, No. 2, Article 14, Publication date: February 2010.

[10] Ke Wei, M. Muthuprasanna, Suraj Kothari. Preventing SQL Injection Attacks in Stored Procedures. *IEEE Software Engineering Conference, 2006. Australian*.

[11] Pietraszek, T. Berghe, C. V. 2006. Defending against injection attacks through context sensitive string evaluation. In *Proceedings of the Conference on Recent Advances in Intrusion Detection.* Springer, Berlin, 124–145.

[12] Sushila Madan and Supriya Madan, "Security Standards Perspective to Fortify Web Database Applications From Code Injection Attacks," IEEE International Conference on Intelligent Systems, Modeling and Simulation, page 226-230, 2010.

[13] Hossain Shahriar and Mohammad Zulkernine, "Taxonomy and Classification of Automatic Monitoring of Program Security Vulnerability Exploitations," The Journal of Systems and Software, pp. 250-269, 2010 (www.elsevier.com/locate/jss).

[14] H. Shahriar and M. Zulkernine, "MUSIC: Mutation-based SQL Injection Vulnerability Checking," The Eighth International Conference on Quality Software, IEEE Computer Society, 2008.

[15] Angelos D and Keromyns, "Randomized Instruction Sets and Runtime Environments," IEEE Security & Privacy, IEEE Computer Society, 2009.

[16] A.Asmawi, Z.M.Sidek, and S.A.Razak, "System Architecture for SQL Injection and Insider Misuse Detection System," IEEE Conference, 2008.

[17] M.Kiani, A. Clark, and G.Mohay, "Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks," The Third International Conference on Availability, Reliability, and Security, IEEE Computer Society, 2008.

[18] J.C. Lin, J.M. Chen, and C.H. Liu, "An Automatic Mechanism for Sanitizing Malicious Injection," The 9th International Conference for Young Computer Scientists, IEEE Computer Society, 2008.

[19] A. Suliman, M. K. Shankarapani, S.Mukkamala, and A.H. Sung, "RFID Malware Fragmentation Attacks," IEEE Conference, 2008.

[20] Y. Kosuga, K.Kono, M.Hanaoka, M.Hishiyama, and Y. Takahama, "Sania: Syntactic and Semantic Analysis for automated Testing against SQL Injection," 23rd Annual Computer Security Applications Conference, IEEE Computer Society, 2007.

[21] J.C. Lin and J.M. Chen, "The Automatic Defense Mechanism for Malicious Injection Attack," Seventh International Conference on Computer and Information Technology, IEEE Computer Society, 2007.

[22] E. Bertino, A. Kamra, and James P. Early, "Profiling Database Applications to Detect SQL Injection Attacks," IEEE Conference, 2007.

[23] S. W. Boyd and A. D. Keromytis, "SQLRand: Preventing SQL injection attacks", In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, Springer-Verlag, June 2004.