I. J. Computer Network and Information Security, 2013, 2, 1-9
Published Online February 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.02.01

Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries

Jaskanwal Minhas[1] and Raman Kumar[2]
[1,2] Department of Computer Science and Engineering,
[1] Sant Baba Bhag Singh Institute of Engineering and Technology, Jalandhar, Punjab, India.
[2] D A V Institute Engineering and Technology, Jalandhar, Punjab, India.
[1,2] {er.ramankumar}@aol.in

Abstract — Due to internet expansion web applications have now become a part of everyday life. As a result a number of incidents which exploit web application vulnerabilities are increasing. A large number of these incidents are SQL Injection attacks which are a serious security threat to databases which contain sensitive information, the leakage of which cause a large amount of loss. SQL Injection Attacks occur when an intruder changes the query structure by inserting any malicious input. There are a number of methods available to detect and prevent SQL Injection Attacks. But these are too complex to use. This paper proposes a very simple, effective and time saving technique to detect SQLIAs which uses combined static and dynamic analysis and also defines an attack other than existing classification of SQLIAs.

Index Terms —Dynamic and Static query, SQL query, SQLIAs

I. Introduction

Due to the easy access to the internet nowadays most of services consist of web services. But the increasing use of internet for important applications poses a greater risk of attacks on the web. SQL-Injection Attack (SQLIA) is not affecting the systems like other attacks, but the ability of SQLIAs to obtain sensitive information, such as military systems, e-business, and banks, etc results into great security risk. The SQLIA occurs when an intruder changes the structure of the query by inserting any SQL keywords. Any secret information can be easily retrieved from database by using these attacks. Variety of techniques are available to detect SQL injection Attacks. Most preferred are Web Framework, Static Analysis, Dynamic Analysis, Combined Static and Dynamic Analysis and Machine Learning Technique. Web Framework[1] provides filters to filter special characters but other attacks are not detected. Static Analysis[2],[3] method checks the input parameter type, but fails to detect attacks with correct input type. Dynamic Analysis[4],[5] technique is capable of scanning vulnerabilities of Web application but is not able to detect all types of SQLIAs. Combined Static and Dynamic Analysis[6]-[9] includes the benefits of both but this method is very complex. Machine learning method[10] can detect all types of attacks but results in number of false positives and negatives. This paper proposes a very simple and effective method to detect SQL Injection Attacks which uses Combined Static and Dynamic Analysis technique. The method is a combination of SQL Query attribute values removal and combined Static and Dynamic Analysis technique. The rest of the paper is organized in the form of different sections. Section 2 describes the categories of SQLIAs. Section 3 discusses the related work. Section 4 explains the proposed method to detect SQLIAs. Section 5 describes the experimental results. Section 6 concludes this paper.

II. Categories of SQL injection attacks

There are a number of different methods of SQL Injection Attacks which are serious threat for any database. For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query. Now the following classification of SQLIAs in accordance to the Halfond, Viegas and Orso[11] researches is presented.

A. Tautologies

This type of attack injects SQL tokens to the conditional query statement which always evaluates true. This type of attack is used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause.

"Select * from loan where loan_no ='L11' and branch ='aaa' OR '1'='1' "

As the tautology statement (l = 1) has been added to the query statement so it is always true.

B. IlIegal/Logically Incorrect Queries

When any query is rejected, an error message is returned by SQL Oracle engine. This error messages help attacker to find vulnerable parameters in the Database.

C. Union Query

By using this technique, attackers join SQLIA to safe query by using word UNION and then can get data about other tables from the application. Consider the following example

Select loan_no,branch from loan where loan_no=$loan_no

By injecting the following loan_no value:

$loan_no=L11 UNION select cust_name from customers We will have the following query:

Select loan_no,branch from loan where loan_no=L11 UNION select cust_name from customers which will join the result of the original query with all the customer names.

D. Piggy-backed Queries

In this type of attack, intruders exploit database by using query delimiter, such as ";" by appending extra query to the original query. In this attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate. So attacker can inject any command related to SQL to the database. In the following example, attacker inject " PO; drop table branch" into the designation input field instead of logical value. Then the application would produce the query:

Select salary from employee where emp_code='23467' AND designation='PO'; drop table branch

Because of ";" character, database accepts both queries and executes them. The second query is not legitimate and can drop branch table from the database.

E. Stored Procedure

Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. By using stored procedure a user can store its own function according to the need. In stored procedure, a collection of SQL queries are included. As stored procedure could be coded by programmer, so, this is also one of the causes of SQLIA. Depending on specific stored procedure in the database there are number of different ways to attack.

Create procedure dbo @emp_code varchar2, @desi varchar2 AS

Exec("select * from employee where emp_code ='"+@emp_code+"' and designation='"+ desi"+"');

GO

If the intruder adds one more query after the legitimate query, then the normal query is converted into piggy backed query which is a type of SQLIA.

Select * from employee where emp_code='23451' and designation='PO';Shutdown;

After the execution of original query the second query which is illegitimate is executed and causes database shut down**.**

F. Alternate Encoding

In this type of attack the regular strings and characters are converted into hexadecimal, ASCII and Unicode. Because of this the input query is escaped from filter which scans the query for some bad characters which results in SQLIAs i.e. the converted SQLIA is considered as normal query.

G. 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. In this situation attacker face to a generic page provided by developer, instead of an error message. 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. Consider two possible injections into the login field:

SELECT accounts FROM users WHERE user_name='abc' and 1 =0 -- AND password= AND pin=0

SELECT accounts FROM users WHERE user_name='abc' and 1 = 1 -- AND password = AND pin=0

If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the user_name field vulnerable to injection.

• Timing Attacks: A timing attack lets attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in this attack the following query is inserted into user_name field:

''legalUser' and ASCII (SUBSTRING ((select top 1 name from sysobjects),1,1)) > *X* WAITFOR 5 --''

Following query is produced from this-

SELECT accounts FROM users WHERE user_name ='legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > *X* WAITFOR 5 -- ' AND password='' AND pin=0

This query is used to extract a table name from the database. In this attack substring function is used to extract first character of first table's name. If the ASCII value of character is greater than X, the attacker can get the character by using 5 seconds time delay in the response of database.

III. Related work

Order to detect and prevent SQLIAs a number of detection methods are available. This section explains the related work.

A. Web Framework

A Web Framework is a software framework that is designed to support the development of dynamic websites, web applications and web services. Some Web Frameworks offer SQL Injection Attack prevention methods. PHP provides magic quotes[1]. Magic quotes is a special feature of PHP language in which special characters ',",/,NULL are pre pended with a backslash

before being passed on to detect SQLIAs. But magic quotes support only four special symbols. SQLIAs with other symbols are not detected.

B. Static Analysis

Static Analysis method analyzes the SQL query sentences to detect SQLIAs. This method verifies user's input type to reduce SQLIAs. JDBC checker[2] validates user input to prevent SQLIAs by using JSA. But if the malicious query has correct syntax and type, attack cannot be detected. Combined Automated Reasoning and Static Analysis method by Wasserman[3] uses FSA to detect SQLIAs. The use of FSA under approximation of the SQL grammar makes this technique too restrictive to remove some possible malicious queries from the represented set. The main problem with all static analysis techniques is that these require source code modification and most of the techniques are just used for web applications written in java.

C. Dynamic Analysis

Dynamic Analysis can locate vulnerabilities of SQLIAs without any source code modification. Paros[4] is a tool written in java used to locate vulnerabilities in web applications. Through its proxy nature all HTTP and HTTPS data between server and client, including cookies and form fields, can be intercepted and modified. Paros is not perfect because it uses predetermined attack codes to scan and uses HTTP response to the success-rate of the attack. Sania[5] finds and collects all the SQL Injection vulnerabilities between web application and database and automatically generate elaborate attacks according to the syntax and semantics. Then the Sania compares the parse tree of intended SQL query and those resulting after attack to assess the safety of vulnerable spots. Due to the use of parse tree Sania is more accurate to detect SQLIAs than any other dynamic analysis technique. Dynamic Analysis methods are useful because no source modification is required. But the vulnerabilities found in web application must be manually fixed by developer and not all of them can be found before predefined attack.

D. Combined Static and Dynamic Analysis

Combined Static and Dynamic Analysis includes the advantages of both Static and Dynamic Analysis techniques. SQLCheck[7] presents the definition of command injection attacks and gives a sound and complete algorithm for preventing SQLIAs based on context-free grammars and compiler parsing techniques. AMNESIA[6] is a model based approach to detect SQLIAs. This technique builds a model of all legitimate queries. Then each dynamic query is compared with this model to detect SQLIAs. Parse Tree[8] is used to compare static SQL query with dynamic SQL query to detect SQLIAs. Wei[9] proposed a technique to detect attacks in stored procedure using Control flow graph.

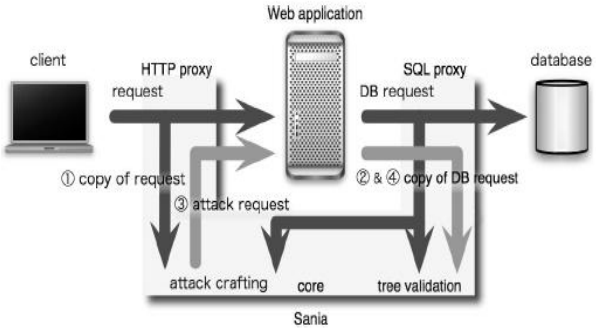

Fig.1. Structure of Sania

E. Instruction-Set Randomization

In Instruction set Randomization SQL query is encoded by inserting a random value in the query. SQLrand[12] uses this technique to protect the query from SQLIAs. But this method is not effective if the random key is predicted.

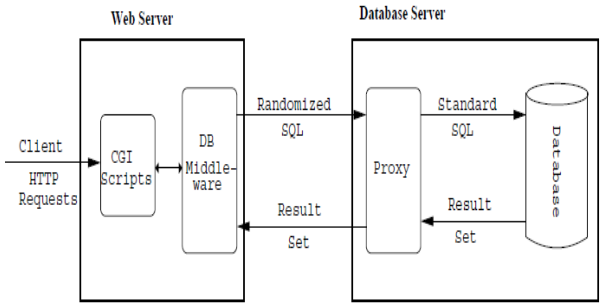

Fig.2. Sqlrand System Architecture

F. Machine Learning

Machine Learning is a technique in which training set i.e. a model is prepared which contain all the legitimate queries belonging to any web application. At run time all the requested queries are compared with queries in the training set to detect SQLIAs. Intrusion Detection System (IDS) by Valeur and colleagues[10] is based on this technique. But this technique results in number of false positives and negatives if the poor training set is used.

G. Prepare Statements

Prepare statements[13] are used in SQL to separate the values in query from the structure of query. In this the skeleton of SQL query is defined and then the values are filled at run time. SQLIA is detected if there is any change in the structure of query. But the main limitation of this method is that the whole web application needs to be modified in order to apply this method.

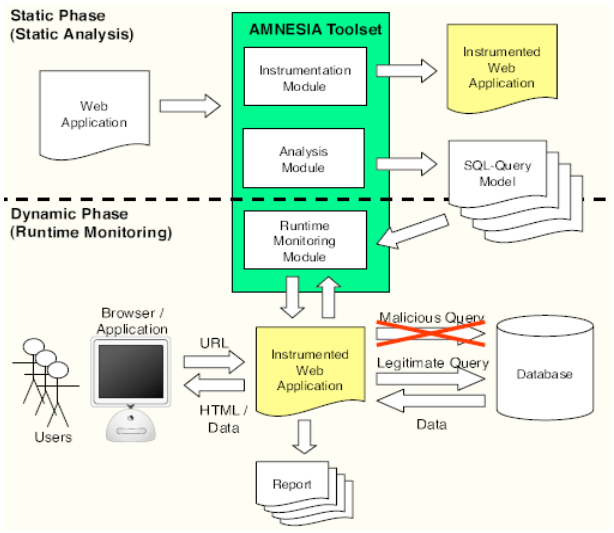

Fig.3. Overview of AMNESIA

H. Taint Based Technique

Java Dynamic Tainting[14] is a tool implemented in java. This tool chases string for taint information instead of character and then sanitizes query strings which are generated using tainted input. But the primary limitation of this tool is that this is not able to detect SQLIAs in numeric fields.

IV. Proposed Method

In this research work combined static and dynamic analysis technique is used to reduce false positives and false negatives. The static query structure is compared with dynamic query. In this database is maintained to store the valid query structure. These valid queries are also known as static queries. The attribute values of dynamic queries are removed during run time and compared with previously stored static queries having same number of tokens as in dynamic query. The symbols needed in the proposed algorithm are given in Table I.

Consider an example to explain the symbols used in Table I.

SQ: select * from users where user_id=? and password=? DQ_t: select * from users where user_id='admin' and password='123456'

DQ_f: select * from users where user_id='' or '1'='1'— and password='123456'

The detection method proposed in this paper will utilize the function *f* given in equation 1, which has the capability to remove the attribute values and replace the attribute values with '?' in SQL queries.

$$DDQ = f(DQ) \quad (1)$$

Another function used in this paper *fn* given in equation 2 calculates the total number of tokens in static and dynamic queries.

$$SQT = fn(SQ), \qquad DDQt = fn(DDQ) \quad (2)$$

In algorithm 1, the function *f*, replaces the values within quotes ('') with symbol '?'.

TABLE I SQLIAs Detection Symbols

Symbol	Description
$I_{\{t,f\}}$	User input data { *t* : normal input , *f* :abnormal input }
f	SQL attribute value removing function
fn	Function to calculate total number of tokens in static and dynamic query
SQ	Static SQL query
$DQ_{\{t,f\}}$	Generated dynamic SQL query from user Input { *t* : normal query , *f* :abnormal query }
$DDQ_{\{t,f\}}$	Removed attribute values of dynamic SQL Query { *t* : normal query , *f* :abnormal query }
SQT	Total number of tokens in static query
$DDQT_{\{t,f\}}$	Total number of tokens in dynamic quer { *t* : normal query , *f* :abnormal query }

In algorithm 2 and algorithm 3, the function *fn*, calculates the total number of tokens in static and dynamic queries.

```
Algorithm f(One SQL query)
Enumerate Quotation_Status= { Quot_Start, Quot_End}
Input String=One SQL query;
Output_String=Null;
Do while( not empty of Input String)
{
Char=Get_Token(Input_String);
If Char is a quotation character and if Char is in between Quot_Start and Quot_End
{
Replace Char with '?'
}
Else {
Add Char to Output_String;
}}
Return Output_String;
```

Algorithm 1: Algorithm which removes attributes

```
Algorithm fn(SQ,Total_Token_S)
//SQ is static query
//Total_Token_S returns the total numbers of tokens in a static query
Input String=SQ;
Total_Token_S=0;
Do while( not empty of Input String) {
Char=Get_Char(Input_String);
If Char is a space character
{
Total_Token_S++;
}
```

Algorithm 2: Proposed Algorithm for token calculation in static query

Algorithm fn(DDQ,Total_Token_D)
//DDQ is a dynamic query generated after removing attribute values
//Total_Token_D returns the total numbers of tokens in a dynamic query
Input String=DDQ;
Total_Token_D=0;
Do while(not empty of Input String)
{
Char=Get_Char(Input_String);
If Char is a space character
{
Total_Token_D++;
}
Algorithm 3: Proposed Algorithm for token calculation in dynamic query

The following example shows the result of functions f and fn. DQ_1 is a normal dynamic query and DQ_2 is an abnormal dynamic query.

SQ = select * from users where user_id=? and password=?

DQ_1 = select * from users where user_id =’admin’ and password=’123456’

DDQ_1=f(DQ_1)=select * from users where user_id=? and password=?

Total number of tokens in dynamic query $DDQT_1$ = fn(DDQ_1) = 8

DQ_2 = select * from users where user_id=’’ or ‘1’=’1’—and password=’abc’

DDQ_2=f(DQ_2)=select * from users where user_id=’??=?—and designation=?

Total number of tokens in dynamic query $DDQT_2$ = fn(DDQ_2) = 7

Following formula is applied regardless of whether a query is normal or abnormal.

If (DDQT=SQT) then// if tokens are same
If (DDQ=SQ) // if queries are same
Then
Result=Normal
Else
Result=Abnormal

N: Total number of fixed SQL queries
SQi: i ’th static SQL query
DQi: Dynamic SQL query generated from SQi
f : Function to delete value of attribute in SQL query
SQ = {SQ_1, . . . , SQ_n},
// Static analysis
1. For i=1 to N
2. Get SQ_i
3. SQT_i=fn(SQ_i) //returns total number of tokens in i^{th} static query and store numeric value in variable SQT_i
4. End {For}

// Dynamic analysis (running time)
6. While(Normal & $\forall k \in N$)
7. Get DQ_k with input values
8. DDQ_k = f (DQ_k) //Remove attribute values
9.$DDQT_k$=fn(DDQ_K) //returns total number of tokens in k^{th} dynamic query and store numeric value in variable $DDQT_k$
10.If ($DDQT_k$=SQT_k)then //if number of tokens in static and dynamic queries are same
If (DDQ_k=SQ_k) //if queries are same
11. Result = Normal
12. Else
13. Result = Abnormal
14. End {If}
15. End {While}
Algorithm 4: Proposed Algorithm for comparison of queries

After removing attribute values this method locates for static queries having same number of tokens as in dynamic query. Then the dynamic query is compared character by character only with that static queries having same number of tokens. The technique of comparing dynamic queries only with that static queries having same number of tokens improves response time. If match is found requested dynamic query is valid query otherwise it is SQL Injection Attack.

V. Experiment and Evaluation

A. Experimental Result Analysis

This section shows the results of experiment. The main advantage of proposed method is its simplicity. The complexity of the algorithm is divided into two parts-first token calculation and second searching for dynamic query. In token calculation complexity depends upon the database we are using because each database has its own syntax but this token calculation technique is same for all databases. In searching for a query best case occurs when the algorithm finds the query in first comparison and the worst case occurs when the query is not in the list. The complexity for searching is O(n). But the best part of this method is that the whole list of static queries is divided according to the number of tokens. The dynamic query is

TABLE II Sample Input SQL Database

S.No.	SQL Statements
1	Select * from branch
2	Select emp_code from employee union select emp_code from branch
3	Select branch_name from branch where emp_code='20496'
4	Select emp_code,branch_name,address from branch
5	Select * from branch where branch_name like 'S%'
6	Select * from employee where designation='PO'
7	Select designation,department,salary from employee where emp_code='19961'
8	Select address from branch where branch_name='STATE BANK OF PATIALA KAPURTHALA'
9	Select * from employee where salary between '20000' and '50000'
10	Delete from employee where designation='APRO'
11	Update employee set designation='PO' where designation='APRO'
12	Update branch set branch_name='STATE BANK OF PATIALA' where emp-code='20496'
13	Create or replace procedure abc(Vdesignation in char)is emp_no number;begin select emp_code into emp_no from employee where designation=Vdesignation;insert into branch values(emp_no,'OFFICE OF THE GENERAL MANAGER RAIL COACH FACTORY','RAIL COACH FACTORY KAPURTHALA');end;
14	Select emp_code,branch_name,address from branch where emp_code in(select emp_code from employee where designation='PO')
15	Select E.emp_code,designation,department,salary,B.emp_code,branch_name, address from employee E inner join branch B on E.emp_code=B.emp_code where designation='PO'
16	delete from branch where emp_code='17589'
17	delete from employee where department='DEPARTMENT OF POST'
18	Select branch_name from branch where emp_code='20496'
19	Select E.emp_code,designation,department,salary,B.emp_code,branch_name, address from employee E right join branch B on E.emp_code=B.emp_code where designation='PO'
20	select emp_code,salary from employee where designation like '_P%'
21	select avg(salary) from employee group by department having department='THE MALL KAPURTHALA'
22	select designation,salary from employee union select branch_name,address from branch
23	select max(salary) from employee group by designation having designation='PO'
24	Select designation from employee where salary='9200'
25	Select designation,count(designation) "no of designations" from employee group by designation

TABLE IV Experiment Results

Total no. of Static queries	No. of inserted SQL queries	Valid SQL queries	Detected SQL Injection attacks	Total time taken (milliseconds)	Average time per query (milliseconds)
255	109	48	61	8580	78.7

TABLE V Accuracy Results

Total no. of inserted queries	Total false positives	Total false negatives
48(Valid queries)+61(SQLIAs)	0	0

TABLE VI Performance Analysis

Total no. of inserted queries	Total time taken by base paper method (Milliseconds)	Total time taken by proposed method (Milliseconds)
109	25318	8580

TABLE VII DETECTION AND PREVENTION METHODS OF VARIOUS SQLIAS

Symbols: '●' defines that detection and prevention is possible '○' defines that detection and prevention is partially possible 'N/A' Not Applicable '✕' defines that detection and prevention is impossible 'ND' defines that attack is not defined

Detection/Prevention Methods	Tautologies	Illegal/Incorrect Queries	Union Queries	Piggy Backed Queries	Stored Procedures	Inference	Alternate Encoding	White Space Manipulation Attack
AMNESIA[6]	●	●	●	●	×	●	●	ND
CSSE[15]	●	●	●	●	×	●	×	ND
Java Dynamic Tainting[14]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SQLCheck[7]	●	●	●	●	×	●	●	ND
SQLGuard[8]	●	●	●	●	×	●	●	ND
SQLrand[12]	●	×	●	●	×	●	×	ND
Tautology Checker[3]	●	×	×	×	×	×	×	ND
Web App. Hardening[16]	●	●	●	●	×	●	×	ND
IDS[10]	o	o	o	o	o	o	o	ND
JDBC-Checker[2]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Java Static Tainting[17]	●	●	●	●	●	●	●	ND
Safe Query Objects[18]	●	●	●	●	×	●	●	ND
Security Gateway[19]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SecuriFly[13]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SQL DOM[20]	●	●	●	●	×	●	●	ND
WAVES[21]	o	o	o	o	o	N/A	o	ND
WebSSARI[22]	●	●	●	●	●	●	●	ND
Base paper method[23]	●	●	●	●	●	●	●	ND
Proposed method	●	●	●	●	●	●	●	●

Compared character by character only with that static queries having same number of tokens. This results in less number of comparisons. Table II shows the sample database. Table III shows the sample injected SQL statements. Table IV, Table V and Table VI shows the experimental results.

B. *Comparison of various detection and prevention methods by attack types*

Table VII shows the comparison of various detection and prevention methods according to various attack types. Halfond[11] classified SQL Injection attacks into seven major categories- Tautologies, Incorrect queries, Union queries, Piggy-Backed queries, Stored procedures, Inference and Alternate encodings. As shown in table VII one more category of SQLIAs is defined named White Space Manipulation Attack which is not defined in any other detection and prevention method. In this type of attack an attacker can manipulate white spaces to prevent an attack from detection.

V. Conclusions & Recommendations

This paper proposes a very simple method to detect SQLIAs which compares static SQL queries with dynamic SQL queries after removing attribute values. To minimize the response time incoming queries are compared character by character only with that static queries having same number of tokens. In this one more attack known as white space manipulation attack other than existing classification of SQLIAs is defined and detected by proposed method. Removing of attribute values makes a SQL query independent of the database. So this method is used for any database. Future work will focus on to detect other types of attacks like cross site scripting attacks.

Acknowledgment

The authors also wish to thank many anonymous models.

References

[1] PHP, magic quotes, http: // www.php.net/magic_quotes/ .

[2] C. Gould, Z. Su, P. Devanbu, “JDBC checker: a static analysis tool for SQL/JDBC applications”, In Proceedings of the 26th International Conference on Software Engineering, ICSE, 2004, pp. 697–698.

[3] G. Wassermann, Z. Su, “An analysis framework for security in web applications”, In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, SAVCBS, 2004, pp. 70–78.

[4] Paros. Parosproxy.org. http : // www.Parosproxy.org/.

[5] Yuji Kosuga et al, "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection", In Computer Security Applications Conference, 2007, pp.107-117.

[6] Halfond W. G, Orso. A, "AMNESIA : Analysis and Monitoring
for Neutralizing SQL-Injection Attacks", In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005, pp. 174-183.

[7] Z. Su, G. Wassermann, "The essence of command injection attacks in web applications", In Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2006, pp. 372–382.

[8] Buehrer. G, Weide. B. W, Sivilotti. P A, "Using Parse Tree Validation to Prevent SQL Injection Attacks", In Proceedings of the 5th international Workshop on Software Engineering and Middleware, 2005, pp. 105-113.

[9] Wei. K, Muthuprasanna. M, Kothari. S, "Preventing SQL injection attacks in stored procedures", In Software Engineering Conference 2006. Australian, 2006, pp. 18-21.

[10] F. Valeur, D. Mutz, G. Vigna , "A Learning-Based Approach to the Detection of SQL Attacks", In Proceedings of the Conference of Detection of Intrusions and Malware and Vulnerability Assessment, 2005, pp. 123-140.

[11] William G.J. Halfond et al, "A Classification of SQL Injection Attacks and Counter measures", In Proceedings of the Intern. Symposium on Secure Software Engineering, 2006, pp. 101-111.

[12] S. Boyd, A. Keromytis, "SQLrand: preventing SQL injection attacks", In Applied Cryptography and Network Security, In LNCS, vol. 3089, 2004, pp. 74-82.

[13] M. Martin, B. Livshits, and M. S. Lam, "Finding Application Errors and Security Flaws Using PQL: A Program Query Language", In Proceedings of the 20th Annual ACM SIGPLAN conference on Object oriented programming systems languages and applications, 2005.

[14] V. Haldar, D. Chandra, and M. Franz, "Dynamic Taint Propagation for Java", In Proceedings 21st Annual Computer Security Applications Conference, 2005.

[15] T.C. Pietraszek, V. Berghe, "Defending against injection attacks through context–sensitive string evaluation", In Proceeding of Recent Advances in Intrusion Detection, in: LNCS, vol. 3858, 2006, pp. 124–145.

[16] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, D. Evans, "Automatically hardening web application using precise tainting information", In Twentieth IFIP International Information Security Conference, in: LNCS, vol. 181, 2005, pp. 295–307.

[17] V.B. Livshits, M.S. Lam, "Finding security errors in Java programs with static analysis", In Proceedings of the 14th Usenix Security Symposium, 2005, pp. 271–286.

[18] W. R. Cook and S. Rai, "Safe Query Objects: Statically Typed Objects as Remotely Executable Queries", In Proceedings of the 27th Intern. Conf. on Software Engineering, 2005, pp. 97–106.

[19] D. Scott, R. Sharp, "Abstracting application-level web security", In Proceedings of the 11th International Conference on the World Wide Web, 2002, pp. 396–407.

[20] R. McClure and I. Kr¨uger, "SQL DOM: Compile Time Checking of Dynamic SQL Statements", In Proceedings of the 27th Intern. Conf. on Software Engineering , 2005, pp. 88–96.

[21] Y. Huang, S. Huang, T. Lin, C. Tasi, "Web application security assessment by fault injection and behavior monitoring", In Proceedings of the 12th International Conference on World Wide Web, 2003, pp. 148–159.

[22] Y. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, S.Y. Kuo, "Securing web application code by static analysis and runtime protection", In Proceedings of the 12th International World Wide Web Conference ACM, 2004, pp. 40–52.

[23] Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values", In Center for Information Security Technologies, Korea University, 2011, pp. 136-713.

[24] Jeom-Goo Kim , "Injection Attack Detection using the Removal of SQL Query Attribute Values", In IEEE, 2011.

[25] W. G. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks", In Proceedings of the Third Intern. ICSE Workshop on Dynamic Analysis (WODA 2005), 2005, pp. 22–28.

[26] Stephen Thomas, Laurie Williams, "Using Automated Fix Generation to Secure SQL Statements", In Third International Workshop on Software Engineering for Secure Systems, 2007, pp. 287-293.

[27] V. Shanmughaneethi et al, "Securing Web Applications with Service Based SQL Injection Detection", In International Conference on Advances in Computing, Control and Telecommunication Technologies, 2009, pp. 702-704.

Ms. Jaskanwal Minhas is working as an Assistant Professor with the Department of Computer Science and Engineering, S B B S Institute of Engineering and Technology, Jalandhar. Before joining S B B S Institute of Engineering and Technology, Jalandhar she did her Bachelor of Technology *with honours* from R I E T, Phagwara

Mr. Raman Kumar *(er.ramankumar@aol.in)* is working as an Assistant Professor with the Department of

Computer Science and Engineering, D A V Institute of Engineering and Technology, Jalandhar.

Before joining D A V Institute of Engineering and Technology, Jalandhar, He did his Bachelor of Technology *with honours* in Computer Science and Engineering from Guru Nanak Dev University; Amritsar *(A 5 Star NAAC University).* He did his Master of Technology *with honours* Computer Science and Engineering from Guru Nanak Dev University; Amritsar *(A 5 Star NAAC University)*. His major area of research is Cryptography, Security Engineering and Information security. He has published many papers in refereed journals and conference proceedings on his research areas.

