IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 14, Issue 6 (Sep. - Oct. 2013), PP 60-68
www.iosrjournals.org

Prevention of SQL Injection Attacks having XML Database

Preshika Tiwari[1] , Ashish Kumar Srivastava[2]
[1] *(PG Scholar CSE, N.I.I.S.T Bhopal , India)*
[2]*(Associate professor CSE, N.I.I.S.T Bhopal , India*)

Abstract: *XML-based Web applications are broadly utilized in computer world, whose main applications are remote operation performance and bring arbitrary data. It is recently used in cloud interfaces, E-Government, Service Oriented Architectures etc. Due to abundant acceptance of this technology large attacks are raised like Denial of Service attacks, attacks on XML Encryption, and XML Signature Wrapping attacks. To stop these types of attacks different techniques were proposed however they're not enough to stop all varieties of attacks. The existing SQLIA prevention techniques can validate the client side data, one by one. It complicates the developer's task to write different validation codes for each data receiving page in the server. This paper proposed an idea of XML based SQLIA prevention technique which can validate the entire client side data by one single call of the dedicated validation function. In this process, the client will submit data in XML format and the server will verify the entire incoming XML file, based on some pr-decided rules called data-rules.*
Keywords: *SQL Injection, XML, Stored Procedure, Data Validation.*

I. Introduction

The widespread adoption of the online as rapid data dissemination and numerous different communications, as well as those having monetary consequences, has basically created it a key element of today's internet organization. These solicitations and their fundamental information bases usually store confidential or perhaps sensitive data. The potential period and damages might have simply quantity to many greenbacks and have additionally prohibited several missions important applications from logging on, that may greatly profit the users. Hence, it's precarious to shield these applications from targeted attacks.

Nowadays many activities are done by dynamic web application. It is clear that private information of people must be kept secret and confidentiality and integrity of them must be provided by developer of web application but unfortunately there is no any guarantee for preserving the underlying databases from current attacks. Web applications are often vulnerable to attacks, which attackers intrude easily to the application's underlying database.

Structural Query Language Injection (SQLI) attack occurs when any attacker tends to manipulate the logic and semantics or syntax of a SQL query by inserting a new SQL keywords or operators. SQL Injection Attack could be a category of code injection attacks that happens once there's no input validation. In fact, attackers will form their illegitimate input as components of ultimate question string that operate by databases. Economic internet applications or secret info systems might be the victims of this vulnerability as a result of attackers by abusing this vulnerability will threat their authority, integrity and confidentiality. So, developers self-addressed some defensive writing practices to eliminate this vulnerability however they're not spare. SQLIAs may escape ancient tools like firewalls and Intrusion Detection Systems (IDSs) as a result of they performed through ports used for normal internet traffic that typically square measure open in firewalls. On the opposite hand, most IDSs specialize in the network and IP layers whereas SQLIAs work application layer.

Researchers have suggested a variety of methods and tools to help inventors to reimburse the fault of the defensive coding [4], [5].

SQL-Injection Attack (SQLIA) constitutes a crucial category of attacks against net applications. By leveraging poor input validation, an assaulter may acquire direct access to the info underlying an application. Any internet application exposed on the web or maybe inside a company computer network may thus be susceptible to SQLIAs. Though the vulnerabilities that cause SQLIAs area unit well understood, they persist owing to lack of effective techniques for detection and preventing them. Defensive programming may, to an extent, shield against bound threats of SQLIAs. However it's impractical to endure this complete method for safeguarding gift systems. Many solutions are proposed to prevent SQLIAs within the application layer that mixes static examination of application level programs and runtime validation of dynamically produced SQL-queries with annexation of user inputs. though these solutions stop SQLIAs at the applying layer, little stress is set on securing objects residing within the info layer like keep procedures that are doubtless susceptible to SQL Injection Attacks. Keep procedures area unit a vital part of a contemporary on-line database. They add an additional layer of abstraction into the planning of software, which implies that, it is long because the interface on the procedure stays an equivalent, the underlying table structure could change with zero noticeable consequence to the application that is using the database. This further layer, to some extent, hides some style secrets from the doubtless malicious users, like definitions of tables. By victimization procedures, one may confirm everyone that the information is usually contained within the info and is rarely exposed. In these databases, the developer is allowed to create dynamic SQL queries i.e., SQL statements square measure designed at runtime in keeping with the various user inputs. For instance, in SQL Server, EXEC (varchar (n) @SQL) may execute discretional SQL statements. This feature offers flexibility to construct SQL statements in keeping with completely different necessities, however faces a possible threat from SQL Injection Attacks

Problem is that some new and current techniques or tools are impractical in reality as they could not address all types of attack or yet not have been implemented. Also some of them need to modify web application code or additional infrastructures. However, the main aim of this paper is to introduce all types of SQL injection attacks and evaluate current approaches which can detect or prevent them.

The rest of the paper is organized as follows. Section II discussed about various possible SQL injection attacks and explains some existing SQL prevention techniques. The motivation of our work is explained in section III. Our proposed XML-SQL is discussed in section IV. Finally section V gives conclusion and future work.

II. Sql Injection Detection And Prevention Techniques

Although inventors deploy protective coding or OS desensitization but they are not enough to stop SQLIAs to web applications so researchers have proposed some of techniques to assist inventors. Huang and colleagues [6] proposed WAVES, a black box testing tool technique for testing web applications for SQL injection vulnerabilities. The tool identifies all points of a web application that can be used to inject SQLIAs. It has builded the attacks that targeted these points and monitors the application how to response to the attacks by utilizing machine learning.

IDBC-Checker [7] was not developed with the determined of detecting and preventing common SQLIAs, but can be used to prevent attacks that yield benefit of type mismatches in a dynamically-generated query string. Most of the SQLIAs comprises of syntactically as well as type correct queries so that the system could not catch more general forms of these attacks. Wassermann and Su propose Tautology Checker [8] that uses static analysis to stop tautology attack. The most important limitation of this technique is that the scope of technique is limited to the tautology and cannot detect or prevent other type of attacks.

Xiang Fu and Kai Qian [9] have proposed a design of static analysis framework, known as SAFELI for defining various SQLIA vulnerabilities at compile time. SAFELI statically monitors the MSIL (Microsoft Symbolic intermediate language) byte code of an ASP.NET Web application, using symbolic execution. SAFELI can analyze the source code and will be able to identify delicate vulnerabilities that cannot be discovered by black-box vulnerability scanners. The main drawback of the technique is that it can discover the SQL injection attacks only on the Microsoft based product.

CANDID [10], [4] modifies web applications written in Java through a program transformation. These tools dynamically extract the programmer-proposed query structure on any one input and detect attacks by comparing it adjacent to the structure of the actual query issued. CANDID's natural and simple approach to be very powerful for detection of SQL injection attacks.

In SQL Guard [11] and SQL Check [12] queries are tested at runtime based on a model which is stated as a grammar that only admits legal queries. SQL Guard checks the structure of the query before and after the adding of user-input based on the model. In SQL Check, the model is identified independently by the developer. Both approaches uses a secret key to restrict user input during parsing by the runtime checker, so the security of the system is dependent on attackers not being capable to discover the key. In two approaches developer should change the 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 [13], [14]. In static phase, it make models of various types of queries, which any application can be lawfully generated at each and every point of access to the database system. Queries are interrupted before, they are sent to the database and are checked not in favor of the statically built models, in dynamic phase. Queries that break the model are prevented from accessing through the database. The main drawback of this tool is that its success is dependent on the correctness of its static analysis for building query models.

J ava Dynamic Tainting [16] is another tool that was implemented for java. Despite of other tool, look for string instead of character for taint information and it tries to sanitize the query strings that have been generated using infected input but unfortunately injection in numeric fields cannot be stoped by this approach. Difficulty of identifying all types of sources of user input is the main limitation of this approach.

Two similar approaches by Nguyen-Tuong [17] and Pietraszek [18], modify a PHP interpreter to track precise per-character taint information. A context sensitive analysis is only used to detect and discard queries if a certain types of SQL tokens has been constructed by a illegitimate input. Drawback of these two approaches is that they require rewriting code.

SQL DOM [19] uses database queries encapsulation for trustable access to databases. They apply a type-checked API which shows that the query building process is systematic. as a result by API they apply coding best practices such as strict user input type checking and input filtering. The major drawback of this approach is that developer will have to learn new programming paradigm or query-development process.

III. Motivation And Problem Defination

All the above SQL injection prevention techniques checks validity of each data item separately. For example, if any user submits a login form than the server will verify the username and password separately. As the growing demand of web application and the complicated data requirements the data submitted by the user is also become complicated .There is a lot of complex HTML form that user has to fill-up and submit. For example, medical store needs to enter multiple product details into one form. Hence, the complexity of data validation into server side can be easily understood. In a single web application different page has different types of HTML form which is essential because each form is dedicated for different task, e.g., user registration, product creation, product purchase etc. Now existing SQL injection prevention techniques handles each data separately. That makes the developer job complicated. The developers have to write code for the validity checking of each html-form separately. Since each form has different types of complexity it is currently not possible to make the validating process as generic for all the forms. This also makes the maintenance job difficult as the data validation policy is different in each from submitting server code.

Different web applications manage the validation rules differently. Also some framework available which allows automated server-side validation and the developer can do very less work to manage it. But complex validation requirements cannot be satisfied by all those automatic validating systems. Also for most of these validating systems, developers have to enter validation rules separately. Sometimes there is a need to send multiple information to server. For example, online customer can purchase 20 items at a time. In that case 20 insertions required in the corresponding database table. Now instead of sending each information separately make one XML file before sending it to the server. On receiving the XML file server will parse the XML file and extract data from it. This is better way to send large and complex information to the server. Also the middle-man attacker cannot easily guess the structure of data. This XML based data passing is not a new concept and is already supporting by almost all server side scripting languages. Also most of the databases have very powerful tool for parsing XML files. So user can use the databases stored procedures to receive XML file and parse it to get all information's. The stored procedure then processes the extracted data one by one as required.

IV. Our Proposal

The proposed idea is one SQL injection prevention mechanism for the applications having XML based data submissions. But our proposed technique will be equally effective for both simple and complex data structure. The XML based SQL injection prevention technique (XML-SQL) is simple and very easy to implement. In this technique all the data validations rules will be stored in a secure place. Whenever server receives any XML based input from client the server will verify the entire XML file based on the verification rules already written in the server.

The main difference between XML-SQL and the other SQL injection prevention technique is that our technique validates the entire incoming data set, at a time. Also the data-rules will be written and stored separately. That makes the developer task easier, what they need to do is just to call the validation function for validating any data. There will be only one validation function written for an application. The validation function will verify the incoming XML files based on the rules written in the XML-rules. For each application the incoming XML files will have registered format for different user requests. Data-rules will be written separately for all types of incoming XML files and the incoming xml file must succeed the validation process.

This process will mainly separate the data validation part of a web application from the application development part. The developer will now no need to worry about the data validity and SQL injection attacks. The data validation part will be maintained by a separate group which will manage the data-rules. This is also beneficial because the normal web developer will be completely unaware about the security rules of the application

Figure 1 explains the basic flow of XML-SQL. When user will submit any query then instead of submitting simple data, it will submit all the data in XML format. Basically it will send one XML file to the server. The server will receive the XML file, parse it and get the original data.

Briefly, the advantage of our proposed technique can be explained as follows:

- No need send multiple requests to the server, all the information can be send in one xml file.
- Server side has rules to verify each XML format. The rules are called data-rules. The rules will be written and stored in secure place in the server. The rules will be written in regular expression.

- Whenever a server receives one user request in XML format, it will parse the input XML and verify it against the rule written in data-rules. This will make the data validation system generic and no need to write code for data validation in each and every page.

Similar to SQL Injection, XPath Injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the XML data is being used for authentication (such as an XML based user file).Querying XML is done with XPath, a type of simple descriptive statement that allows the XML query to locate a piece of information. Like SQL, you can specify certain attributes to find, and patterns to match. When using XML for a web site it is common to accept some form of input on the query string to identify the content to locate and display on the page. This input must be sanitized to verify that it doesn't mess up the X-Path query and return the wrong data. Our proposed technique will be handle such issues securely and confirms that such attacks cannot change the XML-rules written in the server.

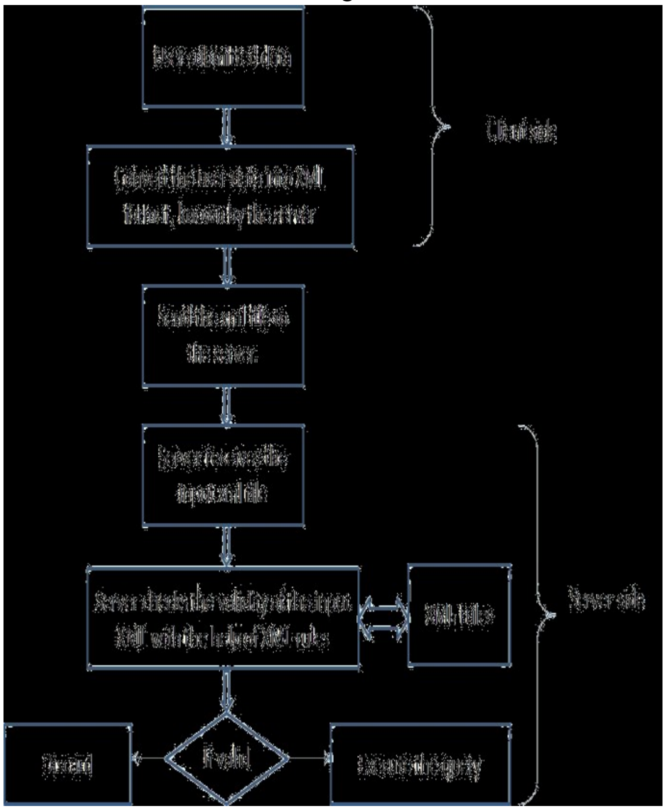

Figure 1 A flow diagram explaining the concept of proposed XML-SQL

1. Technical Details of XML-SQL

This section discuss about the important modules of XML-SQL. The understanding of the technical details of XML-SQL lies on the understanding of such modules. Next part will explain them one-by-one.

1.1 How it works**:** As explained earlier, in XML-SQL data validation happens inside the database server and not in the web application server. The database, have created a stored procedure which accepts all the input data together as a XML file and validate them against the rules saved inside the data base. Hence forth, will call this stored procedure as V-SP. Whenever application server needs to verify any data, it will make an XML file (in proper format) for all such data and passes it to the stored procedure. The V-SP in database will verify each data values against the corresponding rules.

1.2 Input XML format: There are two XML formats used: One is used for user data input into the V-SP, another is used by V-SP to return error message to the application server. Discuss the first one here, the second XML format is explained on section IV-A7. The structure of the input XML file that carries all the input data together to the database for validation is given bellow:

```
<sql-verification> <data-type>T1</data-type> <data-value>V1</data-value>
    <data-type>Tn</data-type> <datavalue>Vn</data-value>
```

</sql-verification>

In this XML format, for each input data there are two elements: i) <data-value> and ii) <data-type>. The <data-value> contains the data value entered by the user while <data-type> contains the name of data rule which must match the data value during the verification process. <data-type> contains only the name of the data rule, the actual rule is written in regular language and stored inside the database. For example, the above XML contains two data: first one (V1) is of type T1 and the second one (Vn) is of type Tn.

1.3 Data Type**:** Data type does not means the generic data types of programming languages. Here data types mean some predefined rules based on which user data should be verified. The data types are stored in database. The table “certificate”, in database stores all the data types. The columns of the table are:

- Name: a unique name of the data type.
- Rule: a rule for the data type written in regular expression.
- Error message: the error message that has to be sending to application server if the validation fails.

A tuple like “< INT, ^[0-9]+$, Only integer accepted >” in table “certificate” is an example of valid data type. Here, INT means the name of the data type, ^[0-9]+$ is the rule the data type written in regular expression and the remaining part is the error message. For complete understanding about how the rules are written, one has to know the basics of regular expressions. The syntax of regular expression slightly varies from, language to language. Since XML-SQL uses such rules inside My SQL database, the regular expression syntax followed are for My SQL. For a brief understanding of regular expression in MYSQL user can go to the link http://dev.mysql.com/doc/refm.

The data types which are default available in database (in the table “certificate”) are as follows:

- INT: Integer number.
- NUM: Any number including decimal point.
- STRING: Any string contains English alphabet, number and spaces.
- PINCODE: Postal pin code format. e.g., 781039 ALPHA: A single English word.

The rule for this data types are written in database. See the table “certificate” in database. Any new data type can be added into the database but the rule must have to write correctly. To add a new data type the person must have some basic understanding of the syntax of MySQL based regular expressions.

1.4 XML generator in Application Server : This module takes all the input data and data type name as input and generates an XML (input XML) from these inputs. The structure of the XML format is explained above. After generating the XML, it calls the stored procedure (V-SP) by passing the generated XML as a parameter. The entire module is written in a function and the function is given below:

```
 function erify_with_xml() { $result="NONE";

    $numargs = func_num_args(); if($numargs%2!=0) {

        $result="INVALID_NUM_ARGS"; return $result;
    }

    $xml='<sql-verification>'; $arg_list = func_get_args();

    for ($i = 0; $i < $numargs; $i=$i+2) { $xml=$xml."<data-
    type>".$arg_list[$i].

            "</data-type><data-value>".$arg_list[$i+1]. "</data-value>";
    }
    $xml=$xml."</sql-verification>";
```

The stored procedure (V-SP) returns three parameters (not directly and will be discussed in next section).

Result: that means the entire data verification succeeded or not.

UID: If the verification succeeded then it will also return one unique id (UID) which can be used for further secure communication with the database. The purpose of UID will be discussed later.

Error Message: If the data verification fails for some data, then the errors messages of all such data types will be returned together as an XML file. The details will be explained in next section.

1.5 Stored Procedure (V-SP): This is the stored procedure written for MySQL (can also be written for other databases). Its takes an XML formatted input from the user and perform data validation for the data contains in the XML. The complete structure of the XML file is already explained in section IV-A2. The task of V-SP is explained step-by-step below:

- Parse the XML.
- Verify each data into the XML: As explained earlier, each data in the input XML has two elements: <data-value> and <data-type>. <data-type> is used to fetch the rules written in the database for that particular data type.
- During verification, if any data fails to satisfy the corresponding rules then the corresponding error message for that rule will be added to the error message. The error message is also written in XML format and will be discussed in the next section.
- If verification succeeded then a 128 bit unique id (UID) will created and saved into the database. The purpose of this UID will be explained later.

The stored procedure does not return anything, in fact it runs a select query just before terminating. This select query creates a result set (which will contain only one row) for 'result', 'result xml' and 'unique id'. Here 'result' return either 0 or 1 depending on whether the data validation processes succeeded completely or not. 'result xml' is the error message and will be empty if 'result' contains 1. 'unique id' contains the generated unique id and will be empty if 'result' contains 0.

The code of V-SP is given bellow. Note that the name V-SP is the generic name of this stored procedure but for the MySQL implementation named it as certify datum v2.

```
DELIMITER $$
CREATE PROCEDURE certify_datum_v2 (IN in_xml VARCHAR(20000),
IN is_detail_required BOOL)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE result_xml VARCHAR(10000) DEFAULT '<result>';
DECLARE unique_id VARCHAR(50) DEFAULT '';
DECLARE data_count INT DEFAULT 1;
DECLARE data_value VARCHAR(500) DEFAULT '';
DECLARE data_type VARCHAR(100) DEFAULT '';
DECLARE error_found INT DEFAULT 0;
DECLARE data_rule VARCHAR(200) DEFAULT '';
DECLARE condition_1 INT DEFAULT 2;
DECLARE error_message VARCHAR(500) DEFAULT '';
 DECLARE temp_message VARCHAR(500) DEFAULT '';
SET data_count=ExtractValue(in_xml, 'count(//sql-verification//data-type)');
WHILE i<= data_count DO
SET data_value=ExtractValue(in_xml, '//sql-verification//data-value[$i]');
SET data_type=ExtractValue(in_xml, '//sql-verification//data-type[$i]');
SELECT rule INTO data_rule FROM certificate
WHERE name=data_type;
SELECT data_value NOT REGEXP data_rule INTO condition_1;
IF condition_1=1 THEN
SET error_found=1;
SELECT error_msg INTO error_message FROM certificate
WHERE name=data_type;
   SET                                    temp_message=CONCAT('<error_message><data_id>',i,
'</data_id><message>',error_message,'</message>
</error_message>');
SET result_xml=CONCAT(result_xml,temp_message);
SET temp_message=''; END IF;
SET i=i+1; END WHILE;
   IF error_found=0 THEN SET unique_id=UID();
INSERT INTO key_checker value(unique_id); END IF;
SELECT error_found AS result, result_xml,unique_id; END$$ DELIMITER ;
```

1.6 What is purpose of UID: The V-SP creates such UID, if the data validation process succeeded. This UID is stored in the database and also returned to the application server. The purpose of such UID is to make the further database communication secure. It may be possible that a hacker break the access of the application server. In that case he or she can remove the XML-SQL based verification process from the application code. If he do this then the data verification process based on XML-SQL will be bypassed and the SQL injection will be entered into the database. To prevent such type of bypass, UID can be used. During the database operation the UID must

also need to pass to the database. In database the UID passed by the user will be compared with the UID stored inside the database and only allow to proceed the operation if both matches. Instead of executing SQL statements directly from application server tt is advisable to use stored procedure for database operations. In this way the UID checking will be easier for the database.
1.7 Error Message XML format: The format of the XML generated by V-SP for error messages is as follows:

```
<result> <error_message>
      <data_id>1</data_id>
      <message>Only integer number accepted.</message> </error_message>
   <error_message> <data_id>2</data_id>
      <message>Only integer number accepted.</message> </error_message>
</result>
```

It is up to the application developer about how to parse and use such error messages.
1.8 How to install XML-SQL: As discussed earlier XML-SQL makes the application developer job easier. They no need to worry about the security rules because the rules will be written by some other person. They just need to use those rules by using the name of the corresponding rule. Also developer cannot see the rules written in the database. In this way the application will be more secure because even developer won't have clear knowledge about the verification rules. Currently it supports only php and MySQL based application but it can be very easily extended for other languages and databases. For installing XML-SQL into your application, just do the following steps:

Verification Database: If your application already has a database then just run the "verify.sql" into you database. After running the sql script it will create two tables (certificate and key checker) and one stored procedure (certify datum v2) in your database. If you don't have any database already then create one database first and then run the verify.sql script on it.

verify.php: Store this php file somewhere in your application so that you can access it. The file contains a function verify with xml which works as XML generator as explained in section IV-A4. Also you need to change the four constants written at the top of the file based on you MySQL database.
That's it your application now completely ready to use XML-SQL. You need a server and MySQL to run such application.

1.9 How to use XML-SQL: Now installation finished and you need to use XML-SQL in your application. Do the following steps: Receive all the user inputs(either by POST or GET method).In PHP normally as like $data1=$_POST ['data-field1']; this part is common and you have to do it to receive the user input. Nothing is new in it. After receiving all the data, confirm the appropriate data type (name of the data type) for each data. If the currently available data types are not exactly meeting your requirements then contact admin to create new data types in XML-SQL database.
Now call the verify function as follows. Verify with xml ($type_1$; $data_1$; $type_2$; $data_2$; : : : ; $type_n$; $data_n$) Where $type_i$ is the data type name of the ith data and $data_l$ is the actual content of ith data. As explained earlier, the function will return result as "ERROR" or "VERIFIED". Also the function saved two values into session: error_ message and uid.Use error_ message or uid appropriately.

V. Results

1. EXPERIMENTAL ANALYSIS

For the experimental analysis of the proposed technique has developed a test bed. The test bed is written in JavaScript and capable to run several queries from different web based applications. Three applications have been used for the analysis purpose. The detail of the application is given in Table I. The applications are developed for the testing purpose of XML-SQL. Each application has multiple html forms through which a user can insert data into the corresponding database. Each html form contains multiple data fields. All the possible SQL injection attacks must have to be entered through these data fields. Proper data validation must have to be done in serve side on these data fields to prevent attacks. Apply XML-SQL technique in server side to prevent attacks through these data fields.

1.1. Input generation

We have generated different types of inputs for each application and for each html form. It contains both safe and malicious inputs. Different types of attacks (as explained in Section I) has been written as input. All the inputs are stored in files . The developed test bed can run multiple inputs and check the number of attacks detected. In test pad the user has to select the application and the html form (from the drop down menus) to run the corresponding inputs.

1.2 Experimental results

After running test bed on all the inputs the results generated are shown in Table II. From the table it shows that XML-SQL detects all the attacks and the detection rate is 100%. The detection procedure of XML-SQL is simple and less expensive as compared to the other equally powerful techniques. Also as mentioned in Section III, it makes the developer's life easy.

TABLE I THE DETAILS OF THE APPLICATIONS USED FOR EXPERIMENTAL ANALYSIS.

Application Name	Description
Employee Management	An PHP based application for managing employees in office
Project Unloader	An academic application for managing students project
Medicine-S	An application for medical shops

TABLE II EXPERIMENTAL RESULTS OF XML-SQL CALCULATED USING THE HELP OF GENERATED TEST BED

Application	Total inputs	Total malicious inputs	Total attacks detected	Rate of detection
Employee Management	200	75	75	100%
Project Unloader	300	157	157	100%
Medicine-S	225	87	87	100%

VI. Conclusion

This paper has presented a new technique for sql injection vulnerabilities and measured different SQL Injection attacks and also the existing SQLIAs prevention techniques exist. Paper have shown that the existing SQLIAs prevention techniques can validate the client side data, one by one. This also complicates the developer's task to write different validation codes for each data receiving page in the server.An idea has been proposed of XML based SQLIAs prevention technique which can validate the entire client side data by one single call of the dedicated validation function. In this process, the client will submit data in XML format and the server will verify the entire incoming XML file, based on some pre-decided rules called data-rules. The future work will equate the tools efficiency and flexibility This tool then would be used as a platform to evaluate the different web application scripts available in the public domain.

References

[1] P. Bisht, P. Madhusudan , and V. N. Venkatakrishnan, "CAN- DID: dynamic candidate evaluations for automatic prevention of SQL injection attacks," ACM Trans. Inf. Syst. Secur., vol. 13, no. 2, pp. 1–39, 2010.

[2] K. Kemalis and T. Tzouramanis, "Sql-ids: a specification- based approach for sql-injection detection," in Proceedings of the 2008 ACM symposium on Applied computing, ser. SAC '08. ACM, 2008, pp. 2153–2158.

[3] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai, "Web application security assessment by fault injection and behavior monitoring," in Proceedings of the 12th international conference on World Wide Web, ser. WWW '03, 2003, pp. 148–159.

[4] G.Wassermann , C.Gould, Z.Su, and P.Devanbu, "Static checking of dynamically generated queries in database appli- cations," ACM Trans. Softw. Eng. Methodol., vol. 16, no. 4, Sep. 2007.

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

[6] Fu and K. Qian, "Safeli: Sql injection scanner using symbolic execution," in Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications, ser. TAV-WEB '08, 2008, pp. 34–39.

[7] S. Bandhakavi, P. Bisht, P. Madhusudan, and V. N. Venkatakr- ishnan, "Candid: preventing sql injection attacks using dy- namic candidate evaluations," in Proceedings of the 14th ACM conference on Computer and communications security, ser. CCS '07, 2007, pp. 12–24.

[8] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent sql injection attacks," in Proceedings of the 5th international workshop on Software engineering and middleware, ser. SEM '05, 2005, pp. 106–113.

[9] Z. Su and G. Wassermann, "The essence of command in- jection attacks in web applications," SIGPLAN Not., vol. 41, no. 1, pp. 372–382, Jan. 2006.

[10] W. G. J. Halfond and A. Orso, "Amnesia: analysis and moni- toring for neutralizing sql-injection attacks," in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ser. ASE '05, 2005, pp. 174–183.

[11] William G.J. Halfond and alessandro Orsa, "Combining static analysis and runtime monitoring to counter sql-injection attacks," SIGSOFT Softw. Eng. Notes, vol. 30, no. 4, pp. 1–7, May 2005.

[12] V. B. Livshits and M. S. Lam, "Finding security vulnerabili- ties in java applications with static analysis," in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14, ser. SSYM'05, 2005, pp. 18–18.

[13] V. Haldar, D. Chandra, and M. Franz, "Dynamic taint propa- gation for java," in Proceedings of the 21st Annual Computer Security Applications Conference, ser. ACSAC '05, 2005, pp. 303–311.

[14] A.Nguyen-tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans, "Automatically hardening web applications using precise tainting," in In 20th IFIP International Information Security Conference, 2005, pp. 372–382.

[15] T. Pietraszek and C. V. Berghe, "Defending against injec- tion attacks through context-sensitive string evaluation," in Proceedings of the 8th international conference on Recent Advances in Intrusion Detection, ser. RAID'05, 2006, pp. 124–145.

[16] R. A. McClure and I. H. Kr¨uger, "Sql dom: compile time checking of dynamic sql statements," in Proceedings of the 27th international conference on Software engineering, ser. ICSE '05, 2005, pp. 88–96.

[17] W. G. J. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter sql injection attacks," in Proceedings of the 14th ACM SIGSOFT inter- national symposium on Foundations of software engineering, ser. SIGSOFT '06/FSE-14, 2006, pp. 175–185.

[18] F. Valeur, D. Mutz, and G. Vigna, "A learning-based ap- proach to the detection of sql attacks," in Proceedings of the Second international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, ser. DIMVA'05. Springer-Verlag, 2005, pp. 123–140.

[19] D. Scott and R. Sharp, "Abstracting application-level web security," in Proceedings of the 11th international conference on World Wide Web, ser. WWW '02, 2002, pp. 396–407.

[20] P.Grazie, "Phd sqlprevent thesis," Ph.D. dissertation, Univer- sity of British Columbia(UBC) Vancouver, Canada, 2008.

[21] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swad- dler: An approach for the anomaly-based detection of state violations in web applications," 2007.

[22] S. W. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in In Proceedings of the 2nd Applied Cryp- tography and Network Security (ACNS) Conference, 2004, pp. 292–302.

[23] W. R. Cook, "Safe query objects: statically typed objects as remotely executable queries," in In Proceedings of the 27th International Conference on Software Engineering (ICSE. ACM Press, 2005, pp. 97–106.