bit-Tech

Vol.1, No.1, August 2018
Available online at: http://jurnal.kdi.or.id/index.php/bt

# Analysis of SQL Injection Attacks on Website Service

**Gregorius Hendita Artha Kusuma[1)*]**

[1)2)] *Pancasila University, Faculty of Engineering, Information Technology, Jakarta - Indonesia*
*Jl, Srengseng Sawah, Jagakarsa - Jakarta*

[1)]gregorius@univpancasila.ac.id

*Article history:*

Received 10 August 2018;
Revised 16 August 2018;
Accepted 25 August 2018;
Available online 19 September 2018

*Keywords:*

*Website*
*Web Security*
*SQL Injection*

*Abstract*

*Among the various types of software vulnerabilities, command injection is the most common type of threat in web applications. In command injection, SQL injection type of attacks are extremely prevalent, and ranked as the second most common form of attack on web. SQL injection attacks involve the construction of application's input data that will result in the execution of malicious SQL statements. Most of the SQL injection detection techniques involve the code to be written along with the actual scripting code. These techniques do not detect errors in SQL statements. Hence, this paper proposes a mechanism to identify invalid SQL statements, to analyze the query for invalid non SQL key words, and to customize the captured errors. This mechanism is different from others by means of separation of the main scripting code and SQL injection code.*

## I. PREFACE

The rapid rise in fraud perpetrated over the internet has brought about the classification of nine types of frauds, developed from the data reported by Internet Crime Complaint centre (IC3). The IC3 website has seen a significant increase in frauds involving the exploitation of valid online banking credentials belonging to small and medium sized businesses. Most of the Internet threats are from software application vulnerabilities and flaw in the design of software system. Vulnerabilities in software may allow a third party or program to gain unauthorized access to some resource. Software vulnerability control is one of the most important parts of computer and network security. Virus program use vulnerabilities in operating system and application software to gain unauthorized access. Intruders use vulnerabilities in operating system and application software to gain unauthorized access, to attack and damage other systems. Hence, avoiding software vulnerability is a major countermeasure to protect software applications from internet threat. It is difficult to design and build a secure web application until the designer knows the possible threats in application. Hence, threat modeling is recommended to be part of the design stages in web application. The purpose of threat modeling is to analyze the application's architecture and identify the potentially vulnerable areas. Developers must follow secure coding techniques to develop secure, robust, and hack-resilient solutions. The design and development of application layer software must be supported by a secured network and hosting systems. Weak input validation is an example of an application layer vulnerability, which can result in SQL injection attack. SQL injection is a technique for exploiting web applications that uses client-supplied data in SQL queries without stripping potentially harmful characters.

The primary target of malicious attackers may be to obtain data from the databases. However, SQL injection offers more than the data. SQL injection enables the attacker to run arbitrary commands in the database. SQL injection bugs lead to disclosing sensitive information, tampering the data, running SQL commands with an elevated privilege
**Advantage,** benefit analysis is expected to help to prevent the occurrence of attacks to the website, caused by irresponsible people using vulnerabilities that exist on the website
**Scope,** making this paper only limited to sql injection attacks.

## II. RELATED WORKS/LITERATURE REVIEW

SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. This attack is possible when the user input is not filtered by the script and passed into a SQL statement[1]. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL

* Corresponding author

ISSN 2622-2728 (online) 2622-271X (print) 
doi:

commands. A less direct attack injects malicious code into strings that are intended for storage in a table or metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.

A simple example of SQL injection is a basic HTML form where the user has to provide username and password:

```
<form method="post" action="process_login.php"> <input type="text"
 name="username"> <input type="password" name="password"> </form>
```

The easiest (and worst) way for the script "process_login.php" to work would be to build and execute a database query that looks like this:

```
"SELECT id FROM logins WHERE username = '$username' and password =
                          '$password'";
```

Under such circumstances, if the variables "$username" and "$password" are taken directly from the user's input, the login script can easily be cheated that a valid password has been provided. If the string „ or „‟=‟ is the password and "abc" is the username, the variables are interpolated and hence the above query would look like this[2]:

```
"SELECT id FROM logins WHERE username = 'abc' and password = '' or '' = '';
```

This query will return a row because the final clause is *or '' = ''* which always evaluate to true (i.e. empty string is always equal to an empty string).

An SQL Injection attack has a set of properties, such as assets under threat, vulnerabilities being exploited and attack techniques utilized by threat agents. Attack techniques are the specific means by which a threat agent carries out attacks using malicious code. Threat agents may use many different methods to achieve their goals, often combining several of these sequentially or employing them in different varieties like tautology, end of line Comment, illegal / Logically incorrect query, union query, piggy-backed query, system stored procedure, blind injection and *OPENROWSET* result retrieval. All these threats can be prevented from the web applications by sanitizing the user inputs and validating the query structure. The existing solutions focus on web application attacks in general and are dependent on the applications. All the solutions are not placed as a layered approach and are specific to back-end databases and specific to platforms.

This paper presents a different methodology to detect and protect SQL Injection in web applications through independent web services in a layered approach. This paper focuses on the identification of invalid SQL statements, analyzing the query for invalid non-SQL keywords which are not present in database, capturing errors, generalizing the error of illegal and logically incorrect queries, printing outputs, detecting SQL injections and maintaining file system for further references. The model proposed in this paper uses a layered approach by which the main scripting code and SQL injection detection code are separated. Also, SQL validation and detection code is implemented as independent web services.

## III. METHODS

Many researchers have contributed in the area of SQL injection. David Morgan describes the concepts of SQL injection, explores the attack vectors, and cites examples for preventing them. Welty describes the nature of SQL errors, various frequencies in which they occur and methods of detecting and correcting them. Johannes and Lam discusses how a website can be defaced through SQL Injection. Prithvi et al. describe the prevention of SQL Injection attacks using the technique of CANDID queries. They initially used the CANDID inputs, then the original inputs and then tested whether the path taken by both queries are the same or any variations do exist. If exist, they detect the SQL Injection. Recent SQL Injection attacks on hundreds of thousands of web sites were hit, including those operated by the UN and UK government. Martin et al. Discussed the syntax to prevent injection vulnerabilities in a language-independent way. In their method, a SQL expression is spliced into the final SQL query. Then this query is composed with inserted values at runtime and L- R parsing is done on the entire query for validation. Dimitris and Diomidis implemented a system, which lies in between web application and

connectivity driver. In order to secure the application from SQLIAs, the driver will go through a training phase. This involves executing all the SQL queries of the application so that the driver can identify them. Then, the driver‟s operations can shift into production mode, where the driver takes into account all the trained legitimate queries to prevent SQLIAs by detecting and blocking them. However, it would be appropriate to have SQL validation and injection code independent of the application and environment. XML based web services provides solutions for such systems. Hence, this paper discusses a layered approach based on web services.

## IV. RESULTS AND DISCUSSION

### System Architecture

This proposed system architecture consists of two layers namely syntactic verification and customize error generation as shown in figure 1. The First layer consists of modules such as query format engine, XML file generation, query parser, query structure analyzer, SQL keyword verifier and XML Schema. The second layer consists of SQL verb verifier, query processor, error logger and error customizer. Both layers are used to analyze user input and protect web applications from SQL injections.

When the user provides the required inputs into the web forms, they are placed into the SQL string in an appropriate place. Finally, the complete SQL string is generated for processing data transaction. The framed SQL string may cause SQL injection in a web application

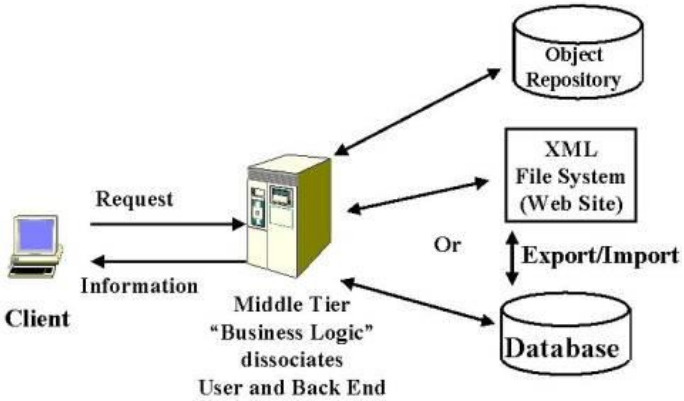



Fig 1. System Architecture

The syntactic verification layer in our approach is used to detect SQL injection. In this the framed SQL string is converted into a XML structure so that, the semantic of the SQL string can be verified through our meta definition. When the SQL string is converted to XML structure, it is subjected to some standard rules which are defined. The root tag of any query is treated as <sql> tag. This root tag becomes root element of XML structure of the given SQL query. The root element must specify the type of the SQL query. To specify the type of the query, SQL queries are broadly classified into five major types such as select, insert update, delete and create statements. Thus the tag following the <sql> tag must indicate what type of statement it is. Hence, select has <select> tag, which means *select* statement. Similarly,

### SQL Statement Classification Schema

Other SQL statements have their own identification tags like <inserts> <updates > <deletes> and <creates>

The words in SQL query is divided into types such as SQL key words and non - SQL words based on the SQL grammar[3]. The SQL keywords form the names of the tags and the non - SQL keywords form the values inside the tag. Once the XML structure is generated using the above rules, the XML so formed from the given SQL statement is validated against the meta definition to find whether the SQL is valid or invalid. Another module in our layered approach creates the meta definition as a meta XML Schema as shown in figure 3. The XML Schema structure is developed for validating the XML file which is generated from the SQL query. This is created in such a way that it validates almost all type of SQL statement. The basic structure of XML Schema is divided into five types of statements as select, insert, update, delete and create. The schema developed must consider all the aspect such as the child elements pertaining to a root tag. For Instance a select statement must have the tags <select>, <from>, <where>, <and>, <or> and so on. For Instance most of the select statements start with the keyword select followed by from.

Thus before any <from> tag, a <select> tag must appear in a select statement and the number of times a particular tag can appear inside a given parent tag must be specified.

**Internal SQL Statement Schema Structure**

This is usually done by specifying the property "minOccurs" in XML Schema. For instance a select statement can appear within another select statement such as "select rollno from student where name in (select name from pupil). The XML generation module, generates the XML structure from the SQL query. The input SQL query is converted into a XML structure and stored as a XML file. The XML file is taken as input file for validation against XML schema. The query is scanned for the list of SQL keywords and tokens. Two arrays, one containing SQL keywords and the other containing the non-SQL keywords are formed. Using the keywords the XML element nodes are created with the SQL keywords as the names of the element and their corresponding non- SQL text as their value. The following example describes this feature

SQL String: *Select roll_no, grade, addr from student*

For the given SQL string <sql> is the root tag as per our definition. Here the verb of the SQL statement is under the classification select. So, the tag is <selects>. The SQL query returns the value of roll_no, grade and address. These are non-SQL keywords. So, all the non SQL keywords are to be placed as value for the element.

*<select> roll_no </select> <select> grade </select> <select> addr </select>*

Further, another SQL keyword is „from' and the non-SQL keyword is „student'. So, the tag is

*<from> student </from>.*

Hence, the XML file will be as follows

*<?xml version ="1.0"> <sql> <selects> <select> roll_no </select> <select> grade </select> <select> addr </select> <from> student </from> </selects> </sql>*

The non-SQL keywords validation module involves the validation of the non-SQL keywords in the input SQL query. This mainly includes the table names, column names and values. An array which consist of non-SQL keywords is taken for this function. In a SQL query, a comment may cause serious threat in web application through SQL injection. To analyze the comment, the comment analyzer module scans the entire SQL query for any comments. If any comments are found in the SQL query, then it is rejected and error is returned from the error generated service layer. However, through non-SQL Keywords tautology based SQL injection causes major threat to web applications. The tautology analyzer performs the protection of verifying the tautology in the SQL String. From the non-SQL keywords the column names and table names are identified. This information is then used to check whether the columns belongs to their respective database table as mentioned in the query. If a mismatch is found, a generalized error is returned to the web client.

The Query structure evaluation module in the first syntactic verification layer validates the structure of the input SQL query using the XML generated by validating it against the XML schema. It takes the XML file generated and schema as input and does XML schema validation. If the validation passes, the query structure is considered as correct and it returns true.

The second layer is to protect SQL injection by logically incorrect queries and illegal queries. Framing illegal / logically incorrect query technique is usually used by the threat agent during the information gathering stage of the attack. By injecting illegal / logically incorrect requests, an attacker may gain knowledge about the injectable parameters, data types of columns within the table, name of tables, etc[5]. Although every database management system in the commercial market support ANSI/ISO standard Structured Query Language, each vendor also develops a proprietary SQL language dialect. Almost every SQL injection attack within web application threat is targeted to a specific database. But there is a need for general solution to commonly targeted databases like MS SQL Server, MySQL, Oracle, DB2, Sybase, Informix and MS Access. Most of the available solutions are specific to a commercial database software. Through our customized error generation layer, the threat agent cannot deduce specific details such as injectable parameters.

There are four modules in the customize error generation layer. The SQL verb verifier module reads the SQL string and verifies the verb of the string. If the prime verb is not matched with the dataset of SQL verb, then the SQL string is rejected and not send for processing the query. If the SQL string passes the verification, then the string is passed to the error processing module, where the query is executed. After the execution of the SQL string, the query processing module returns the result as dataset to the web server. If an attacker tries to know the schema of the table or any details of the table with illegal query, then the illegal SQL string returns the exact error messages to the web server. In our layered approach, if the query returns valuable result set, then the result set is returned as a dataset to the client via web server as shown in figure 4. If the processed query returns the specific error messages, then the error message is customized by a error customizer module as SQL error and returned to the web server for client information.

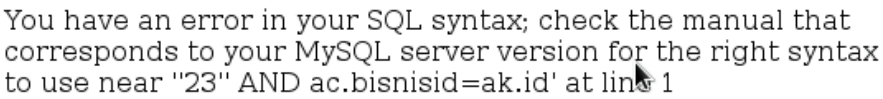
**Valuable Result set**

This test have been conducted manually for every sample ofwebsite using URI disturbance to get the information whetherthe web application have any possibility of the SQL injectionsecurity hole. One of the way to do this testing is to write input manually with a single quotation coma (‘) at the end of URI space. In Fig. 6 is shown the local website before the SQL injection method has been applied.


Fig 2. Website before SQL Injection is performed

The lack of this web application is causing by the absent of filter at the input stage, as so let the manual input of a single quotation coma (‘) may caused the error [4-6]. The error is shown in Fig. 3. (a) and Fig. 3. (b)


(a)




(b)

From the error revealed in Fig. 3 (a) and 3 (b), it is shown that the above website have security flaw over the SQL injection. Furthermore, by using some other sophisticated method, a hacker can directly digging any important information from those website. An example of the hijacked information is shown in Fig. 4.


Fig 4. Information Arise from SQL Injection

**Testing Using W3af**

After evaluating manual test, this research continues the security hole testing by using *w3af* application for any sample of website. Firstly, this application will perform discovery process by *w3af* where it will searching the server header from the website (as shown in Fig. 5) and indexing the website to produce website Universal Resource Identifier (URI) list before it perform the URI mapping (as shown in 10). A moment after finishing URI mapping, *w3af* application will continue doing SQL injection audit for those website by using keyword *d'z"0*. In some website, this keyword can be encoded with URL encode as *d%27z%220*.

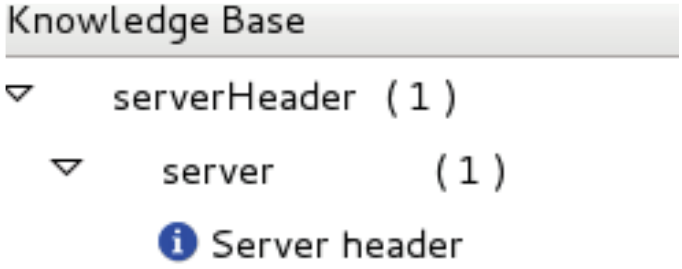



Fig 5. Server header search

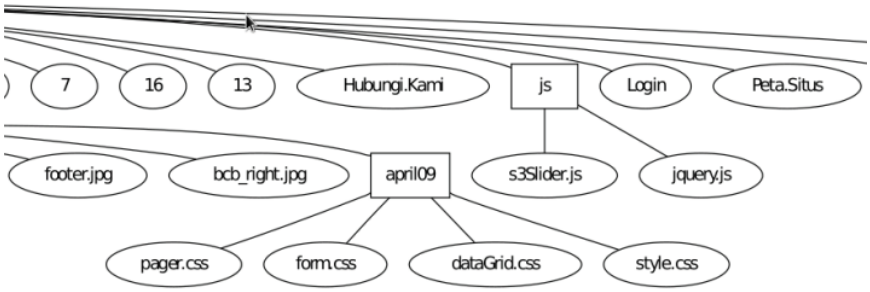



Fig 6. URI mapping

Fig. 7 (a) have shown the sample of website which has SQL injection security hole, as seen in detail those website have some SQL injection security hole in many places inside the website. In addition, it is shown clearly in Fig. 7 (b) that the HTTP header request in *w3af* application revealed the SQL injection security hole is in parameter tbl_kde. Moreover, Fig. 7 (c) have shown that w3af application revealed the information that it have found the SQL injection security hole.

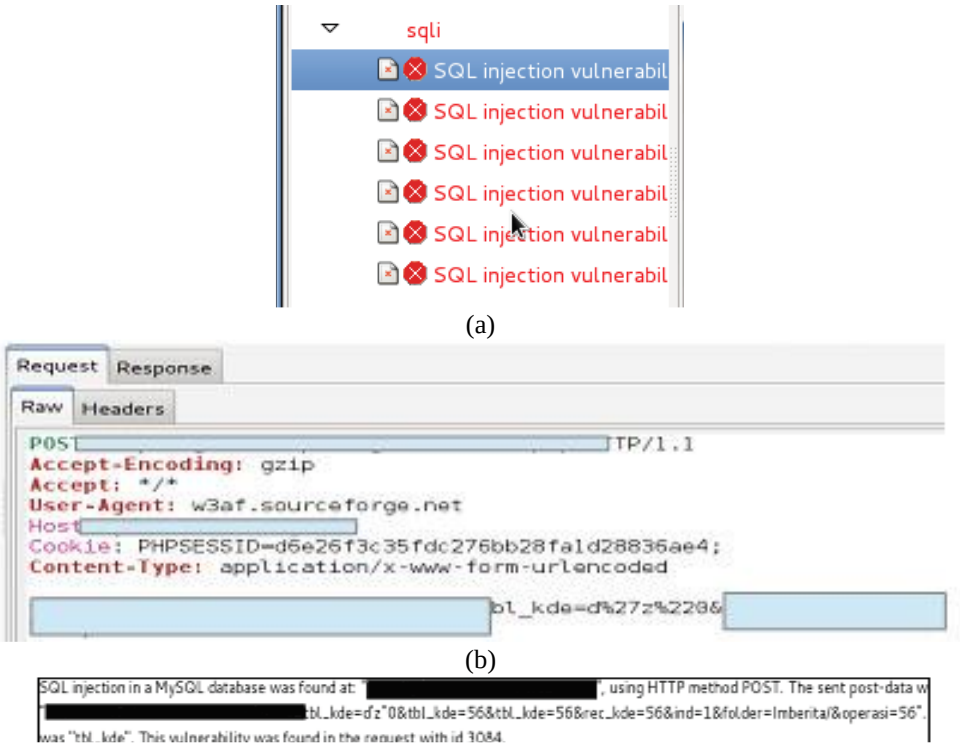



Fig 7. SQL Injection in w3af Application SQL Injection Security Hole (b) HTTP Header Request (c) SQL Injection Information in W3af Application

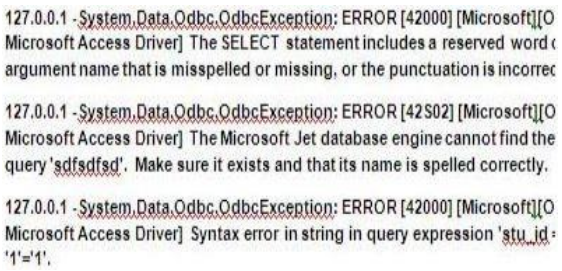



Fig 8. Log File

This module will not reveal the real SQL error message or error code to the client to achieve SQL Injection. From the generalized error message as SQL Error, the attacker will not be able to understand the table schema or any type of database information.

```
1  <diffgr:diffgram xmlns:msdata="urn:schemas-micorosoft-com:xml"
     xmlns:diffgr="urn:schemas-micorosoft-com:xml-diffgram-v1">
2  <NewDataSet xmlns="">
3      <Table1 diffgr:id="Table11" msdata:rowOrder="0"
       diffgr:hasChange>
4          <Error_Message>SQL Error</Error_Message>
5      </Table1>
6      </NewDataSet>
7      </diffgr:diffgram>
```

Fig 9. Generalized (Customized) Error

To evaluate how well the proposed approach provides solution to protect SQL Injection in web application, we analyze the performance of the web application based on the response time with our layered approach as well as without the layered approach. Each layer‟s response time is evaluated independently and the response time is tabulated. Table1 shows the response time of the syntactic layer compared without the syntactic layer.

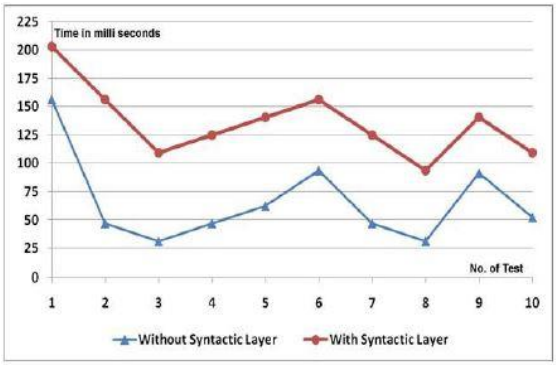



Fig 10. Comparision of response time

The above graph shows the comparison on the response time, between with layer and without layer approach. This time difference is not significant compare to the SQL Injection vulnerability. Here, the time varies in only milli seconds. Moreover, our approach uses the independent web service to detect the SQL injection. So, any change in web application does not affect the web service which protects SQL injection and vice versa.

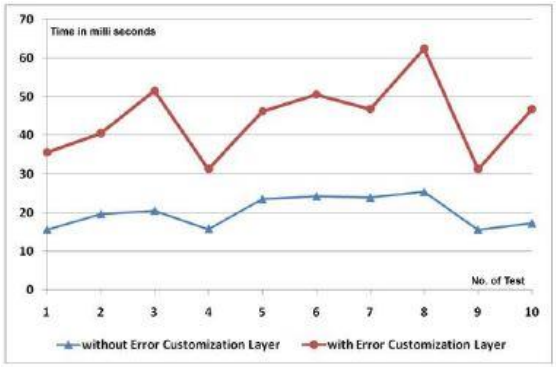



Fig 11. Comparision of response time

Graph shows the comparison on the response time between with error customization layer and without error customization layer approach. This time difference is not significant compared to the SQL Injection vulnerability. Here, the time varies in only nano seconds. Moreover, this approach uses the independent web service to detect the SQL injection. So, any change in web application does not affect the web service which protects SQL injection and vice versa

## V. CONCLUSION

Many web sites in the world has vulnerable, which can be hacked by such SQL injection technique. Hacker can input abnormal string on input form to corrupt a system. In this paper, we proposed a layered web service approach for detecting the SQL Injection by tautology, illegal/ logically incorrect queries and piggy pack. Comparing with previous approaches, the layered web service approach is independent to the platform and work on any type of back end database. We analyzed the web application with the layered web service and found that the response time of the web application. In future, we intend to analyze the input string which is given as input to the web form by a user. The independent analysis of the input string will give the greater performance to protect SQL Injection. If user input is properly analyzed, we can protect SQL injection in a better way

## REFERENCES

[1] R. Ezumalai and G. Aghila, "Combinatorial Approach for Preventing SQL Injection Attacks," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, 2009.

[2] H. Alnabulsi, M. R. Islam, and Q. Mamun, "Detecting SQL injection attacks using SNORT IDS," in *Asia-Pacific World Congress on Computer Science and Engineering, APWC on CSE 2014*, 2014.

[3] J. Clarke, *SQL Injection Attacks and Defense*. 2009.

[4] Sharmin Rashid, Subhra Prosun Paul. 2013. "Proposed Methods of IP Spoofing Detection & Prevention."

[5] Simar Preet Singh, A Raman Maini. 2011. "Spoofing Attacks of Domain Name System Internet"