An Efficient Protective Layer Against SQL Injection Attacks

Bojken Shehu [1] and Aleksander Xhuvani [2]

**[1] Computer Engineering Department, Faculty of Information Technology
Polytechnic University of Tirana
Tirana, Albania**

**[2] Computer Engineering Department, Faculty of Information Technology
Polytechnic University of Tirana
Tirana, Albania**

Abstract

In this paper, we present a detailed discussion on different SQL injection attacks and their prevention technique. In addition, we proposed a new scheme for prevention of SQL injection attack, which consist of three blocks or three tier architecture: the clients, the application server and the database server. Our protective layer works between the clients and application server. Therefore, before sending SQL queries to the database, the protective layer will analyze the query to check the vulnerability. If found any, it reported else it forward the query to database server. The proposed scheme is efficient and overhead is negligible.

Keywords: *SQL Injection, Web Security, Vulnerabilities, Prevention, Database security.*

1. Introduction

In recent years, most of our daily tasks are depend on database driven web applications because of increasing activity, such as banking, booking and shopping. SQL injections are the most common yet critical threats to any website as it is concern to the database, which is the valuable to any organization. These websites helps the users to make personal account there for online transactions [15], or to store their confidential data. As the popularity of internet increases, the use of online and automated processes are also, increases and therefore huge bulks of sensitive and critical data are being handled by the web applications. As the stakes on the information and data stored by the portals become higher, so does the sophistication of hackers. Developers and hackers are racing against each other. Developers try to make the web application secure from the threats and the hacker wish to find the loophole, so that it can steal or damage the application or data. Security threats could be with the intent of stealing confidential information [18], causing deliberate damage, proves capability or simply for the thrill of doing something which most others cannot do.

With SQL injections, cyber-criminals can take complete remote control of the database, with the consequence that they can become able to manipulate the database to do anything they wish, including:
Insert a command to get access to all account details in a system, including user names and retrieve passwords from registry.

- ✓ Shut down a database.
- ✓ Upload files.
- ✓ Through reverse lookup, gather IP addresses and attack those computers with an injection attack.
- ✓ Corrupting, deleting or changing files and interact with the OS, reading and writing files.
- ✓ Online shoplifting e.g. changing the price of a product or service, so that the cost is negligible or free.
- ✓ Insert a bogus name and credit card in to a system to scam it at a later date.
- ✓ Delete the database and its all contents.

There must be some rules that one should be incorporated in every website to make it secure from SQL injections. Many Web applications can be exploited because the user input is being processed in an unsafe manner. All the data provided by a user must be treated as untrustworthy. One of the key requirements for a Web application's security is the proper user input handling, which is not always an easy task. To propose the classification the inputs based on probability and use of character as a vulnerability that helps to identify in SQL detection process. Proper neutralization of such special characters used in an SQL command to avoid the SQL injection.

2. SQL Injection Attack Techniques

The SQL injection attacks can be performed using various techniques [19]. For each attack we identify a pattern of the attack. Some of them are specified as follows:

Tautologies: *The main goal* of tautology-based attack is to inject code in conditional statements so that they are always evaluated as true. Using tautologies, the attacker wishes to either bypass user authentication or insert inject-able parameters or extract data from the database. A typical SQL tautology has the form, where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. Bypassing authentication page and fetching data is the most common example of this kind of attack. In this type of injection, the attacker exploits an inject-able field contained in the WHERE clause of query. He transforms this conditional query into a tautology and hence causes all the rows in the database table targeted by the query to be returned. For example, SELECT * FROM user WHERE id='1' or '1=1'-'AND password='1234'; "or 1=1" is the most commonly known tautology. In this type of attack the injected code will always start with a string terminator (') followed by the conditional OR operator. The OR operator will be followed by a statement that always evaluates to true. The signature for such attacks is the string terminator (') and OR.

Logically incorrect query attacks: *The main goal* of the Illegal/Logically Incorrect Queries based SQL Attacks is to gather the information about the back end database of the Web Application. When a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical error by purpose. In this example attacker makes a type mismatch error by injecting the following text into the input field: 1) Original URL:*http://www.toolsmarket-al.com/veglat/?id_nav=2234* 2) SQL Injection: *http://www.toolsmarket-al/veglat/?id_nav=2234'* 3*) Error message showed: SELECT name FROM Employee WHERE id=2234\'. From the message error we can find out name of table and fields: name; Employee; id*. By the gained information attacker can organize more strict attacks. The Illegal/Logically Incorrect Queries based SQL attack is considered as the basis step for all the other techniques. In this type of attack there are several ways to perform illegal or incorrect queries like incorrectly terminating the string ('), using AND operator to perform incorrect logics, using order by, etc.

Union Query: *The main goal* of the Union Query is to trick the database to return the results from a table different to the one intended. By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application. This technique is mainly used to bypass authentication and extract data. *For example the query executed from the server is the following: SELECT Name, Phone FROM Users WHERE Id=$id. By injecting the following Id value: $id =1 UNION ALL SELECT credit Card Number, 1 FROM Credit sys Table. We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT credit card Number, 1 FROM Credit sys Table.* This will join the result of the original query with all the credit card users. The signature of this attack is UNION character of SQL.

Stored Procedures: *The main goal* of the Stored Procedures SQL attack is to perform privilege escalation and try to execute the SQL procedures. SQL injection attacks of this type try to execute the SQL procedures. Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. As stored procedure could be coded by programmer, so this part is as inject-able as web application forms. Depend on specific stored procedure on the database there are different ways to attack. In the following example [4], attacker exploits parameterized stored procedure. *CREATE PROCEDURE DBO. is Authenticated @user Name varchar2, @pass varchar2, @pin int ASEXEC("SELECT accounts FROM users WHERE login='" +@user Name+ "' and pass='"+@password+"'and pin="+@pin); GO For authorized/unauthorized user the stored procedure returns true/false. As an SQL injection attack, intruder input "'; SHUTDOWN; - -"for username or password. Then the stored procedure generates the following query: SELECT accounts FROM users WHERE login='boni' AND pass=''; SHUTDOWN; -- AND pin=* . After that, this type of attack works as piggy-back attack. The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down. So, it is considerable that stored procedures are as vulnerable as web application code. The signature of this attack will be the same as that of piggy-backed queries.

Piggy-Backed Queries: *The main goal* of the Piggy-Backed Query is to execute remote commands or add or modify data. In this attack type, an attacker tries to inject additional queries along with the original query, which are said to "piggy-back" onto the original query. As a result, the database receives multiple SQL queries for execution. Vulnerability of this kind of attack is dependent of the kind of database [5]. *For example, if the attacker inputs [';drop table users--] into the password field, the application generates the query: SELECT Login_ID FROM users_ID WHERE login_ID='john' and password=''; DROP TABLE users-' AND ID=2345* After executing the first query, the database encounters the query delimiters (;) and execute the second query. The result of executing second query would result into dropping the table users, which would likely destroy valuable information. The signature of this attack is (;), the database line terminator.

Inference: *The main goal* of the inference is to change the behavior of a database or application. There are two well-known attack techniques that are based on inference: blind injection and timing attacks.

Blind Injection**:** Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field: *For example, SELECT accounts FROM users WHERE id= '1111' and 1 =0 -- AND pass = AND pin=0 SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=0* If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submits the first query and receives an error message because of "1=0 ". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection. The possible guessing start with the AND operator and some time attacker also uses conditional operators.

Timing Attacks**:** A timing attack lets an attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. This technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. *For example, declare @ varchar (8000) select @s = db_name () if (ascii (substring (@s, 1, 1)) & (power (2, 0))) > 0 waitfor delay '0:0:5'* Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter. WAITFOR is a function used for delaying the response from the database. In this type of attack the IF ELSE statement is used for injecting queries. So the possible signatures of this attack are WAITFOR, IF, ELSE.

Alternate Encodings: *The main goal* of the Alternate Encodings is to avoid being identified by secure defensive coding and automated prevention mechanisms. Hence it helps the attackers to evade detection. It is usually combined with other attack techniques. In this technique, attackers modify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode. Because by this way they can escape from developer's filter which scan input queries for special known "bad character". By this technique, different attacks could be hidden in alternate encodings successfully. *In the following example the pin field is injected with this string: "0; exec (0x73587574 64 5f177 6e), " and the result query is: SELECT accounts FROM users WHERE login=" AND pin=0; exec (char (0x73687574646j776e))* This example use the char () function and ASCII hexadecimal encoding. The char () function takes hexadecimal encoding of character(s) and returns the actual character(s). The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the attack string. This encoded string is translated into the shutdown command by database when it is executed.

3. Literature Review

After studying many researches on the SQL injection detection and prevention, it is found that not a single technique is strong enough to handle all the problems and vulnerabilities to websites due to SQL injections. Some of the techniques proposed the changes at the development stages. As in [1], the proposed technique works for defending the SQL injections against the stored procedure. Similarly, various researches provide defending techniques against SQL injection.

Authors in article [2], proposed a Swaddler which, analyses the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state.

Authors in article [3] proposed a string tokenizer which, creates tokens of original query and SQL-injected query, and creates an array of tokens of both the original and injected query, if the length of arrays of both query is found equal, that means no SQL-injection., Otherwise there is injection.

Authors in articles [4], proposed a proxy filter which can be effective against SQLIA; they used a proxy to filter input data and output data streams for a web application, although correctly specify filtering rules for each application is required by the developers to input.

Authors in article [5] proposed a mechanism which filters the SQL Injection in a static manner. The SQL statements by comparing the parse tree of a SQL statement before and after input and only allowing to SQL statements to execute if the parse trees match.

Authors in article [6] deals with an application specific randomized encryption algorithm to detect and prevent it further its effectiveness was compared with other existing techniques and its performance was quantified. Hence we took up this web security vulnerability and analysed its attack types. The security threat posed SQLIA is really high and it is very necessary to protect users‟ data in a web application, since it is very confidential and sensitive.

Authors in article [7] uses a methodology which make use of an independent web service which is intended to generalize the syntactic structure of the SQL query and validate user inputs. The SQL query inputs submitted by the user are parsed through an independent service and the correctness of the syntactic structure of the query are checked. The main advantage of this paper is that the error message generated does not contain any Meta data information about the database which could help the attacker. Since the web service is not integrated with the web application, any modification that should be done to the system should be done in such a way that it should be supported by the web service.

Authors in article [8] proposed a translation and validation (TransSQL) based approach for detecting and preventing SQL Injection attacks. The basic idea of this approach relies on how different databases interpret SQL queries and those SQL queries with an injection. After detailed analysis on how different databases interpret SQL queries, Kai-Xiang Zhang, et.al proposed an effective solution TransSQL, using which the SQL requests are executed in two different databases to evaluate the responses generated.

Authors in article [1] proposed a technique to defend attacks against the stored procedures. This technique combines a static application code analysis with a runtime validation to eliminate injection attacks. In the static part, a stored procedure parser is designed, and for any SQL statement that depends on user inputs, and use this parser to instrument the necessary statements in order to compare the original SQL statement structure to that including user input. The underlying idea of this technique is that any SQLIA will alter the structure of the original SQL statement and by detecting the difference in the structures, a SQLIA can be identified.

Authors in article [9] proposed a combinatorial approach for shielding web applications against SQL injection attacks. This combined approach incorporates signature based method, used to address security problems related to input validation and auditing based method which analyze the transactions to find out the malicious access. This approach requires no modification of the runtime system, and imposes a low execution overhead. It can be inferred from this approach that the public interface exposed by an application becomes the only source of attack.

Authors in article [10] used SVM (Support Vector Machine) for classification and prediction of SQL-Injection attack. In proposed algorithm, SQL-Injection attack detection accuracy is (96.47% and which is the highest among the existing SQL-Injection detection techniques.

Code Checkers are based on static analysis of web application that can reduce SQL injection vulnerabilities and detect type errors. For instance, JDBC-Checker [11] is a tool used to code check for statically validating the type rightness of dynamically-generated SQL queries. However, researchers have also developed particular packages that can be applied to make SQL query statement safe [12].

Authors in article [13] proposed AMNESIA that combines dynamic and static for preventing and detecting web application vulnerabilities at the runtime. AMNESIA uses static analysis to generate different type of query statements. In the dynamic phase AMNESIA interprets all queries before they are sent to the database and validates each query against the statically built models. AMNESIA stops all queries before they are sent to the database and validates each query statement against the AMNESIA models. However, the primary limitation in AMNESIA according to article [14] is that the technique is dependent on the accuracy of its static analysis for building query models for successful prevention of SQL injection.

Authors in article [16] proposed an Aspect Oriented system for detecting and prevent common attacks in web applications like Cross Site Scripting (XSS) and SQL Injection and evaluate its performance by measuring the overhead introduced into the web application. The results of our tests show that this technique was effective in detecting attacks while maintaining a low performance overhead.

4. Proposed Work

Our proposed work consists of four parts: the clients, our protective layer, the application server and the database server (Fig.1). We proposed a protective layer approach to address the problem of SQL injection attacks. By protective layer we mean a program written in java that runs between client side and application server. Every request coming from the client must pass through the protective layer before being processed by the application server. If the request contains any of the attacks signatures mentioned in the previous section, it is illegitimate access to the database. The goal of this work is to prevent illegitimate access to the web application and database.

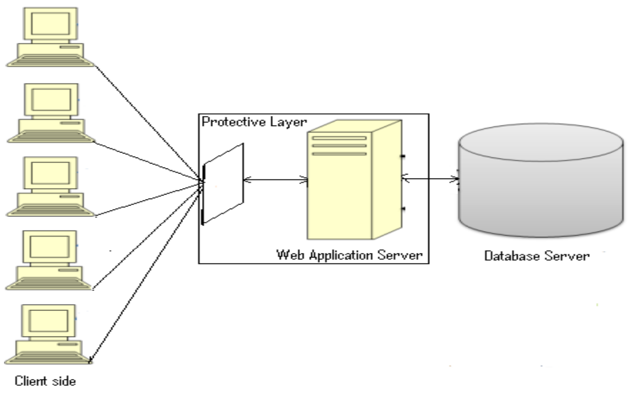

Fig. 1 Proposed architecture

As you can see from our proposed work, the SQL queries are generated by the application server. Our proposed protective layer checks the presence of SQL signatures before the input is processed by the application server. The proposed work uses protective layer to prevent SQL injections, we will implement our protective layer in sixth steps.

Algorithm:

```
1. First, we checked the link request from client.
2. We parse the input.
   A. If parser return TRUE (String Parse successfully), go to step 4.
   B. If parser return FALSE (String Parse un-successful), go to step 3.
3. Check attack pattern, this step check the existing signature to compare with.
   A. If the signature found, generate error message.
   B. If the signature not found, go to step 4.
4. Make transaction, the inputted data does not contain any vulnerable character.
5. Call error page to record every event in a log file
6. Exit.
```

We just need to embed the protective layer between client side and application server of any website, will the help of proposed technique we will able to prevent SQL injection attacks.

5. Implementation

The complete implementation of the proposed work carried out using Visual Studio 2008, SQL server 2008 and a vulnerability scanner tool. Through which we can scan our own developed website for any vulnerability threat and SQL attacks. To discuss the effectiveness of our protective layer approach, we also demonstrate the whole project by different web applications along with the database in SQL server 2008. The demonstrated website works in two modes at every instance. Whether its login or search, inserting or updating, etc.

1. Safe mode: In this mode the website works under the guidance of protective layer, which sanitizes the inputs and always looking for any intrusions or malicious code in the input. In this mode the protective layer works by checking the presence of these attack signatures in the user input before it is processed by the application server. The most important feature of our proposed framework is that it is generic and does not

depend on the application server as well as the underlying database, it never overhead the data in the database. It can be applied to any existing web model without performing any alteration.

2. Unsafe mode: The unsecure mode in the demonstrating website is never uses any SQL injection prevention strategy. It simply pas the data from client to the database.

6. Experimental Results and Analysis

While working with demonstrating website, by using the vulnerability scanners and concurrent users, we successfully generate the above number of request in five different iterations. The proposed implemented protective layer gives the appropriate results by analyzing the inputs provided to them and finally output the counts of the valid and injected queries. Table 1 and figure 2, shows the results of the different testing sessions on the proposed implemented work. It shows the actual number of requests made to the web application out of which the proposed layer successfully detects the injections.

Table 1: Results obtain by using the vulnerability scanner

	Total Request	***Injected Data***	***Valid Data***
Test 1	4808	3345	1463
Test 2	3204	2121	1083
Test 3	1765	1173	592
Test 4	2968	1984	984
Test 5	4178	3045	1133

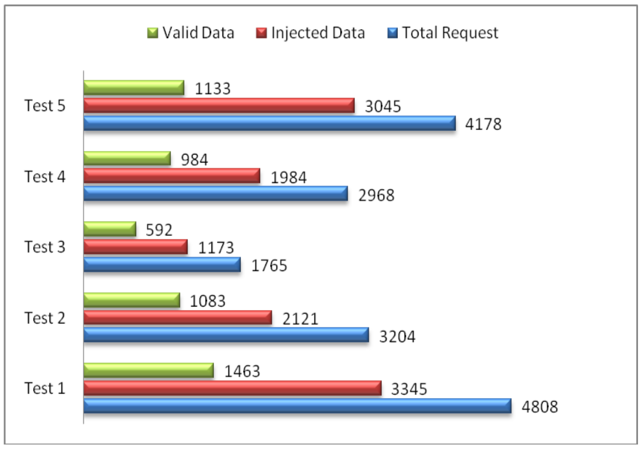

Fig.2 Graphical representation of results

7. Conclusions

In this paper, we have concentrated on the specific area of SQL injection. According to OWASP's Ten Most Critical Web Application Security Vulnerabilities [17], many SQL injection-related issues are among the most harmful threats to web applications. Though this is a narrow subject, we believe that this area need of further investigation, mainly because of two reasons: first, we can not be certain that we have compiled a definite list of all components that could be taken into consideration. Secondly, SQL injection attacks are most likely to evolve and new vulnerabilities will be found, together with new countermeasures to deal with them. One of our goals in this paper was to increase the level of security awareness among organization regarding web applications, especially towards SQL injection threats. The protective layer is generic and does not depend on the application server as well as the underlying database. The efficiency of the protective layer was tested by using a vulnerability scanner tool.

References

[1] K. Wei, M. Muthuprasanna, S. Kothari, "Preventing SQL Injection Attacks in Stored Procedures". Proc of the 2006 Australian Software Engineering Conference (ASWEC" 06).

[2] C. Marco, B. Davide, F. Viktoria, and V. Giovanni, "Swaddler: An approach for the anamoly based character distribution models in the detection of SQL Injection attacks", Recent Advances in Intrusion Detection System, Pages 63-86, Springerlink, 2007.

[3] N. A. Lambert and K. Song Lin," Use of Query Tokenization to detect and prevent SQL Injection Attacks", IEEE, 2010.

[4] G. J. William, F. Hal, O. Alessandro, "A Classification of SQL Inject ion Attacks and Countermeasures", College of Computing, Georgia Institute of Technology.Gatech.edu.

[5] G.T. Buehrer, B.W.Weide and P.A..G.Sivilotti, "Using Parse tree validation to prevent SQL Injection attacks", In proc. Of the 5th International Workshop on Software Engineering and Middleware(SEM'056), Pages 106-113, Sep. 2005.

[6] A. Srinivas, G. Narayan, S. Ram. "Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection, in Trust, Security and Privacy in Computing and Communications (TrustCom)",2012 IEEE 11th International Conference, pp.no. 1327 – 133, 25-27 June 2012.

[7] V. Shanmughaneethi, C. Emilin Shyni and S.Swamynathan, "SBSQLID: Securing Web Applications with Service Based SQL Injection Detection" 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies, 978-0-7695-3915-7/09, 2009 IEEE.

[8] K. Zhang, Ch. Lin, Sh. Chen, Y. Hwang, H. Huang, and F. Hsu, "TransSQL: A Translation and Validation-based Solution for SQL-Injection Attacks", First International Conference on Robot, Vision and Signal Processing, IEEE, 2011.

[9] R. Ezumalai, G. Aghila, “Combinatorial Approach for Preventing SQL Injection Attacks”, 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6-7 March 2009.

[10] R. Romil, R. Shailendra, "SQL injection attack Detection using SVM", in International Journal of Computer Applications, Volume 42– No.13, March 2012.

[11] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: A static analysis tool for SQL/JDBC applications," 2004, pp. 697-698.

[12] R. A. McClure and I. H. Krüger, "SQL DOM: compile time checking of dynamic SQL statements," 2005, pp. 88-96

[13]W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006.

[14]B. Indrani and Ramaraj, "An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service," International Journal of Computer Science and Network Security, vol. 11, pp. 197-205, 2011.

[15]A. Marin, I. Marsida, and Sh. Bojken "Using PKI to Increase the Security of the Electronic Transactions" 8th Annual South-East European Doctoral Student Conference. September 2013. Greece.

[16]K. M. Elinda, K. Lorena, V. Enid and Sh. Bojken "Protection of Web Application Using Aspect Oriented Programming and Performance Evaluation", 5th Balkan Conference in Informatics, 16-20 September 2012. Novi Sad, Serbia.

[17] The Open Web Application Security Project. A guide to building secure web applications, Version 1.1.1 Online documentation, sep 2002.

[18] Sh. Bojken, A. Shqiponja, A. Marin, and Xh. Aleksander, "Protection of Personal Data in Information Systems", International Journal of Computer Science, Vol. 10, No. 2, July 2013, ISSN (Online): 1694-0784.

[19] Sh. Bojken, “Analysis of SQL Injection Attacks on Web Applications”, 4th International Conference in Information Systems and Technology Innovation: towards a digital Economy. ISTI-2013. Tirana, Albania.

Bojken Shehu. He is a pedagogue in Polytechnic University of Tirana, Faculty of Information Technology, in Computer Engineering Department. In 2007 he has finished the Bachelor Thesis in Saint Petersburg State Polytechnic University, Russia and in 2010 he has finished the Master Thesis in Bauman Moscow State Technical University, Russia and now he is a PhD student in Polytechnic University of Tirana, Albania.

Aleksander Xhuvani. He is a pedagogue in Polytechnic University of Tirana, Faculty of Information Technology, in Computer Engineering Department. He has finished the PhD study at Bordeaux in France. At 2004 he is graduated as Prof. Dr.