International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 12, December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

Triple Phase Protection Technique to Prevent SQL Injection Attacks

Shyamsundar Pushpad[1], Chirag Juneja[2], Anmol Chawla[3]

Department of Computer Applications

National Institute of Technology Kurukshetra, Haryana, (India)

ABSTRACT

Any kind of threat to the data, an indispensable asset for an organization, implies the whole organization is in jeopardy. Therefore, protecting the underlying database is the most critical issue for corporations, especially, those dealing with personal information of users such as applications for banking, shares, e-commerce etc. Of all the attacks, SQL Injection Attack (SQLIA) is considered to be one of the top threats to the databases. We have shown the recent scenario of injection attacks which clearly depicts the requirement of improvement in this field of study. Through the user interface provided by the web application, the attacker injects malicious code to get access to the database. In this work, we have proposed a multi-layered approach to prevent the web applications from various SQLIA. This approach consists of an input inspection after which a phase of query splitting follows. And finally we have a method of query categorization. The proposed approach is an attempt to intercept SQLIA types of a particular kind. This method focuses on eradicating attacks which use SQL operators to inject malicious code. We have also made evaluation of the proposed technique based on certain valuable parameters.

Keywords: SQL injection attacks, Tokens

I.INTRODUCTION

In today's world organizations deal with huge amounts of data of thousands and millions belonging to their users. These information are stored in the form of databases in their servers. Generally these organizations make use of personal data of users which are very crucial to the organizations too. They store important information like passwords, secret account related data etc. These data, if compromised, may affect the organization very badly [1]. Also, this may impact bad effect on the user and organization's relationship. So, it is the prime objective of any organization to keep the data of its users safe.

The major issue here is the proper implementation of the underlying database(s) and lack of proper validations in the web applications used by the organization [2]. This gives any malicious user to get access to the database and perform the intended malicious activity, hence putting the whole organizational data in jeopardy. Therefore the security of the web application used by the organization becomes the task of utmost priority.

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 12, December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

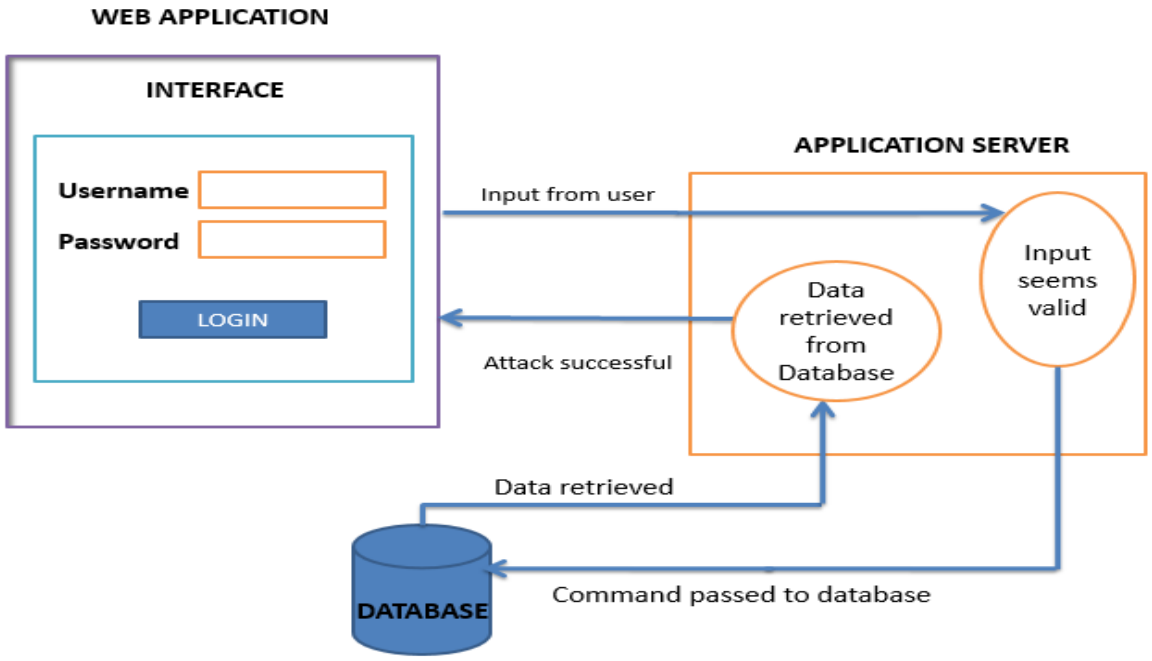

Figure 1: Attack performed on a web application's database

Any organization using a web application makes contact with its users through an interface to make exchanges of information as well as services. This interface is the part of the application from a path is created that reaches the database by indirect means. If validations are not applied in a proper way, then this interface may act as a loophole by means of which the attacker can directly get access to the database. Generally, the intruder targets the Metadata, which is the structure of the database. Figure 1 depicts an attack performed by an attacker into an application having a poor security.

The paper is constructed in the following mentioned manner. Section II contains a brief about the SQL Injection and the attack process. It also states a few to mention SQLIAs which we have focused in the implementation phase. In section III, we have discussed about various types of SQL injection attacks those have been witnessed in the very recent time. Section IV gives a detailed explanation of the proposed architecture. Section V shows the evaluation of the proposed model, that is, the nature, efficiency and the effectiveness of the proposed system.

II.SQL INJECTION

SQL Injection is one of the most popular and largely used breaching methods for an attacker to get illegal access to a database. SQL Injection is a technique in which attacker can inject malicious code through SQL to get illegal access to databases. Injected SQL commands can alter SQL statement and compromise the security of a web application.

Most of the web applications provide login pages or forms in which user provides information. This is the entry point for the attacker to enter the malicious code which will be incorporated in the SQL query and helps the attacker to enter the user's account without correct data. [5]

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 12, December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

Following is the whole process that shows how the injected code passes through the application server checking and gets access to the underlying database.

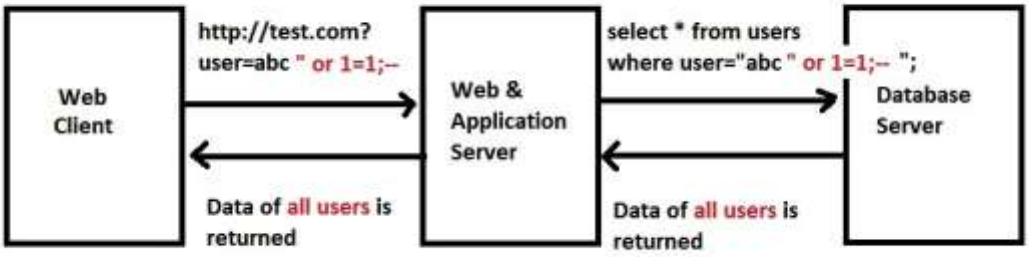

Figure 2: Malicious user input on the log in form

In this we present different type of SQL Injection Attacks. These attacks are generally not performed individually, many of them are combined together or performed sequentially to attack.
Many type of attacks can be made by doing slight variations.

A. Tautology

Tautology is a logical formula which is true in every possible interpretation. In this attack the code is injected using the conditional OR operator. The objective of this attack is to inject code in conditional statements so that they always evaluate to true. Tautology is injected in the query's 'where' condition. Making the conditional expression into a tautology make it possible for the attacker to login successfully without having username and password.
A typical SQL tautology id of the form "or <comparison expression>", where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. Comparison expression can have any type of operator like =, >, < etc.

Example: Bypassing login script
Query:
Select * from mydb.login where username= ' "+_username +"' and password = ' "+_password +" ';
This query takes input from the system user, suppose user enters:
Username: a' OR '1'='1
Password: a' OR '1'='1
Constructed Query: SELECT * FROM User WHERE Username='a' OR '1'='1' and Password='a' OR '1'='1';

The injected code '1'='1' in condition converts whole WHERE clause into a tautology and allows the attacker to enter into the application without username and password.

B. End-of-line comment:

The end of line comment (--) is used to disable part after '--' from being executed. Double hyphen makes everything after that till the end of line, a part of comment. In this attack user has to give correct username in the username field with double hyphen after that.

Example: Bypassing login script
Query:
Select * from mydb.login where username= ' "+_username +"' and password =' "+_password +" ';
This query takes input from the system user, suppose user enters:
Username: 'yashi'--
Password: '' (we can write anything in this field or not write anything)
Constructed Query: SELECT * FROM User WHERE Username='yashi'--' and Password= '';

The above query will disable the Password field and make it comment by writing '--' before it which allows the attacker to enter into the application without password.

C. Hybrid of Tautology and End of line comment attack:
This type of attack will combine both tautology and end of line comment attack which will help the attacker to get the benefits of both attacks. Now, attacker don't need to know the username due to tautology and tautology in password field is not required due to end of line comment.

Example: Bypassing login script
Query:
Select * from mydb.login where username= ' "+_username +"' and password =' "+_password +" ';
This query takes input from the system user, suppose user enters:
Username: a' OR '1'='1'--
Password: '' (we can write anything in this field or not write anything)
Constructed Query: SELECT * FROM User WHERE Username='a' OR '1'='1'--' and Password= '';

In the above query there is tautology in the username field and after that '--' will disable the Password field and make it comment which allows the attacker to login successfully without username and password.

D. Illegal/logically incorrect queries:
In this type of injection an attacker is trying gather information about the type and structure of the back-end database of a Web application. Attackers inputs the illegal/logically incorrect queries so that SQL database servers return error messages. These errors contain a plenty of useful information. This attack helps an attacker to collect significant information about the name, type, version, data type and the structure of the back-end database of web application. This attack is mostly used as preliminary for other attack techniques. Additional

error information is provided to help the programmers in debugging, further helps the attackers also to get information about the schema further helps the attackers also to get information about the schema.

Example:

Query: select * from mydb.login where username= ' "+_username +"' and password =' "+_password +" ';

This query takes input from the system user, suppose user enters:

Username: yashi' cvds

Password: '12354' (we can write anything in this field)

Constructed Query: SELECT * FROM User WHERE Username='a' OR 'yashi' cvds ' and Password= '12354';

Above query is logically incorrect query which will give error. Error can be like:

Exception: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'yashi' cvds' at line 1.

From the above error attacker got the information about the database server type which will further helps him to attack.

References: [6, 7, 8]

III.PRESENT SCENARIO AND RECENT ATTACKS

Among various web attacks and vulnerabilities, SQLIA is the top most [3]. From a very long time, SQLIA has continued to be among the top ten attacks. According to the reports conducted by the OWASP, it is very clear that SQL injection has proved to be a top preference for the malicious purposes related to web applications which are database driven. The following statistic from OWASP makes it very comprehensible regarding the top attacks.

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Figure 3: Percentage of vulnerability by class

From the above figure, it is very clear that SQLIA is one of the top attacks from a very long time. The ranking may vary in different web security vulnerability rankings according to different organizations, but it is among the top ten attacks everywhere and is mostly among top five [4].

i. Application name: India BHIM mobile money app
   Date: 2017-01
   The app also had SQL injection vulnerability, using which hackers extracted bank account details.
ii. Application name: Indian Embassy websites
   Date: 2016-11
   Recently several Indian Embassy websites were hacked by Kapustkiy and Kasimierz, two Pentesters from Netherland, using the most basic security flaw of Sequence Query Language (SQL) Injection.
iii. Application name: World Anti-Doping Agency
   Date: 2016-08
   Hacked412MB of data including 3,121 email accounts and passwords. The attack was executed by SQL injection flaw with SQLMap SQL Injection Automation Tool, according to Hacked-DB's analysis.
iv. Application name: Bitcoin
   Date: 2016-08
   Hacker hacked hundreds of Bitcoins and showed how you can steal bitcoin with SQLI.
v. Application name: Gaana Music Service
   Date: 2015-05
   User data exposed for 12.5 million users.
vi. Application name: World Trade Organization
   Date: 2015-05
   Anonymous Hacker breached WTO database and Leaked data of internal staff using SQLI

IV.PROPOSED SYSTEM

We have proposed a system that is focused on the concept of detecting the attack by split checking the generated query. The data which is passed on to the application server for this checking, is the validated data. That is, first the data is checked in the client end for any kind of malicious content and after that, it is passed on to the server. The data which is inputted from the user or the attacker reaches the web application server in the form of an SQL query. Be it parameterized or normally, the data is converted in a form that can be directly processed by the database server. After the checking performed here at the application server end, the query is then classified on the basis of legality according to the application. This final phase makes it easy for the future use by reducing much of processing by the application server.

This suggested model focuses on SQLIA of those types which make use of SQL operators in general for performing the injection. Those attacks include End-of-line comment attack, tautology, logically incorrect

queries, etc. The basic idea behind this approach is creating tokens by splitting the query on the basis of spaces and quotes, and then performing checks on those tokens.

Following are the processes which are performed in our proposed model of preventing the intruder from performing SQLIA. These phases are performed individually one after the other.

A. Inspection of Input

The provided interface is the area where the user gives input according to the application. This phase, basically, targets on validation of the given input. The kinds of attacks discussed above can be done only by the use of some malicious content. And this malicious content is possible only by the use of operators like quotes and spaces. If the attacker is not allowed to enter some kind of malign input, then the attack can be intercepted at this phase itself.

By the use of JavaScript, the interface like forms, login page, etc. are validated. According to the requirement and type of the application, proper validations can be made. Suppose an application can run without the use of quotes, then this validation can be of great help. Restricting the intruder from entering quotes can intercept a few attacks like tautology.

B. Query Spllitiing

The concept behind this phase is that there are very few attacks that can be made possible by not increasing the size of the normal query. This means that, if the attacker wants to inject malicious code to the normal query of size, suppose 'N', then there are a few observations to be focused on, that can help intercepting that attack. These observations can be proved to be true for the attacks of the type which we have focused on.

i. Firstly, the size of the query is definite to increase if the attack is to be made. That is, the malicious query will have size greater than 'N'
ii. Secondly, single quote is the prime requirement in the query to perform the attack of a particular kind.
iii. Thirdly, every query is required to have spaces inside it. For example, between the table name and the keywords used.

Based on these observations, this phase states that the generated query from the user or attacker input is split into chunks called tokens or small components of the query. This process will be performed for both the normal query as well as the query generated from the user input. Then these tokenized format of queries are compared on the basis of size. For that, they are required to be stored into some data structure. We use arrays as insertion of tokens one the after is very easy and calculation of the size of the arrays can be done very efficiently. Now the generated tokens of both the queries are inserted in the separate arrays. Both the arrays are now compared and two cases can be there:

i. Size of query 1 is equal to the size of query 2
ii. Size of query 2 is greater than the size of query 1

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 12, December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

According to 'Case i', as the size of both arrays is same, this clearly states that there is no injection done. The query is a legal one and does not contain any kind of malicious content. No attack out of tautology, end-of-line comment and hybrid of both attacks is present in this user generated query.

The 'Case ii' unambiguously states that affirmation can be made from the fact that size of both arrays is different and some extra content has been added to the normal possible query. This means that attack has been made of some kind.

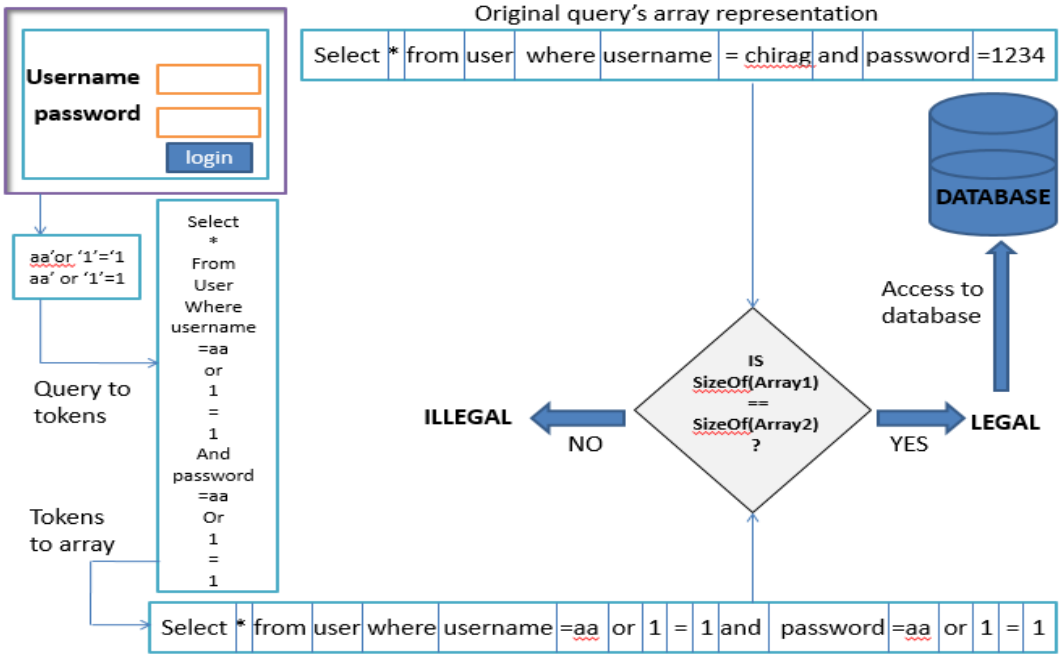

Figure 4: Query splitting process

Hence, after the splitting process and then on performing the check of length checking, the attack can easily be said to have detected. This will not allow the injected query to go further. That is, the application server will not pass this injected query to the database server for further processing, thus disallowing the intruder from getting access to the database.

C. Categorizing the Queries

After the processing of above phases, the proposed system is capable of inferring the type of query. It implies that now the application server can say that the query is a legitimate or an illegitimate one. This kind of labelling can be used as a future measure for deciding the legitimacy of the queries generated of the similar category.

The query can be one of the two types:

i. Label 1 - Legitimate
ii. Label 2 - Illegitimate

The 'Label 1' queries are free from any kind of attack, therefore, no further processing is required for such kind of queries. The 'Label 2' queries are categorized into the malicious type. These queries also require no further processing, but they will not be allowed to move in any of the further tiers.

V.EVALUATION OF THE PROPOSED SYSTEM

Every system is evaluated on the basis of some parameters and constraints. This approach is to be evaluated on the grounds of attacks focused on. The approach discussed in this paper targets on only those attack types which include the use of SQL operators and keywords in the attack technique. So base on this, we can assess the model and make following deductions in the form of evaluation:

A. Application specific validations: In this system, the validations part can be applied according to the use or type of the web application involved. So a web application which does not require the use of quotes can be checked in the validation part itself, making the process faster. Also, the validation is done at the client side, due to which the checking is done swiftly.

B. Making difficulty for the attacker to intrude: This system provides checking and validations at different tiers, vis-à-vis, at the client side and at the application server side. This makes the intruding task very daunting for the attacker. The more the time required for the attacker to intrude, the less will be the chances of successful attack.

C. Processing time: The different phases involved in the proposed system makes it very efficient in terms of time constraint. That is, the time required for all the checking and validations is very less. The reason for this being, very simple but efficient calculations are applied here.

D. Simplicity of the approach: Very simple but effective methods, calculations and data structure are used in the suggested model. The data structure involved here is arrays and calculations include only the length checking of the arrays and very less time complexity is associated with it.

E. Ease of processing frequent queries: The final phase in the system is the categorization of the queries which have been processed. We get a classification of a list of processed queries at the end of processing of this step. Therefore, f similar query occurs in the future, then it will be checked from that list of classifications and further processing will not be required which will save a lot of processing time and overhead of the web application using this approach.

VI.CONCLUSION AND FUTURE WORK

SQL injection attack is one of the web security issues and has gained attention of the attackers due to access to database. Due to potential of the risks associated with SQLIA, this issue has gained the attention of researcher over the last few years. Many efforts in their direction carried out result in emergence of different tools and techniques. Though different tools and techniques proposed for the issue but many of them have problems as some require source code modification, some limited to certain database and some suffer with performance issues. In web application the response time maters much. If we use the techniques that take too much time, then it will not be a good system. The SQL injection detection and prevention technique used in this paper is faster. It

detects and prevents the SQL injection effectively. Our system will pass legal requests only, because the input given by the user is first inspected then we split the query in to the tokens and after that categorization of that queries is done. Tokenization based Architecture system detects more SQL injection. In addition to the detection and prevention of SQL injection our system takes less time to process web request.

The future work includes putting light on various other SQL injection attacks. We have focused on some common injection attacks like tautology, union queries, incorrect queries etc. But still many of the attacks are there which are needed to be studied. The current injection attacks which are focused on, consists of those which use SQL operators and keywords to inject malicious code. Work can be done to append some other approach to the current system so that it can be able to target attacks of other kinds also.

Also, the space issues related to the three phase technique can be eyed upon. The use of extra array increases the space complexity. This can also be made the center of interest for the future work of the proposed system.

REFERENCES

[1] HALFOND, W. G. J., ORSO, A., AND MANOLIOS, P. 2006. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'06). ACM, New York, 175–185.

[2] Al-khashab, E., F.S. Al-Anzi and A.A. Salman, 2011. PSIAQOP: Preventing SQL injection attacks based on query optimization process. Proceddings of the 2nd Kuwait Conference on E-Services and E-Systems, April 5-7, 2011, Kuwait, USA.

[3] D. Aucsmith. Creating and Maintaining Software that Resists Malicious Attack. http://www.gtisc.gatech.edu/bio aucsmith.html, September 2004. Distinguished Lecture Series.

[4] Ahttp://codecurmudgeon.com/wp/sql-injection-hall-of-shame/

[5] N. W. Group. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1. Request for comments, The Internet Society, 1999.

[6] S. McDonald. SQL Injection: Modes of attack, defense, and why it matters. White paper, GovernmentSecurity.org, April 2002.

[7] "Category:OWASP Top Ten Project". OWASP.

[8] Praveen Kumar, "The Multi-Tier Architecture for Developing Secure Website with Detection and Prevention of SQL Injection Attacks," International Journal of Computer Applications (0975 – 8887) Volume 62– No.9, January 2013.