Volume 6, Issue 10, October 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Defense Mechanism to Avoid SQL Injection with Query Filters

Angel. N, Dr. Chandrasekar. A
Department of CSE, St. Joseph's College of Engineering,
Chennai, India

Abstract— Security of any web-application is very important due to its excessive use in daily routine life (such as business, education, health etc). One of these attacks is SQL injection which can give attackers illegal access to the databases. This paper presents the prevention of SQL injection attack by providing query filters to the access to the database by blocking the unusual client through whitelist. The goal of this paper is to provide improved security by developing a method which prevents illegal access to the database

Keyword-whitelist, SQL Injection, whitelist, attacks, filters.

I. INTRODUCTION

For questions SQL injection is a code injection technique, used to attack data-driven applications, in which nefarious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

SQL injection attack is done by insertion or "injection" of a SQL query with the input data from the user to the application. A successful SQL injection can read sensitive data from the database, alter database data and perform query such as Insert/Update/Delete and perform administration operations on the database such as shutdown the Database, recover the data present in a file on the Database and perform some commands on the operating system.

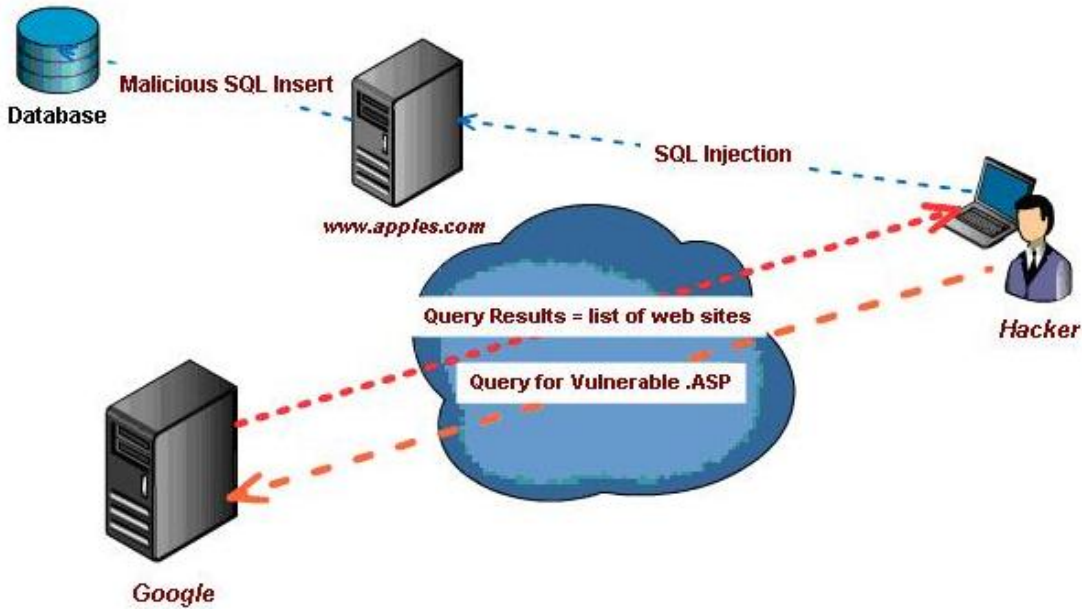


Fig.1: SQL Injection Example

II. CLASSIFICATION OF SQLIA

Classification of SQLIA

***Tautology*:** In the tautology attack the attacker tries to use a conditional query statement to be evaluated always true. Attacker uses WHERE clause to inject and turn the condition into a tautology which is always true. The simplest form of tautology.

Example
SELECT *FROM Accounts WHERE user=''or1=1 'AND pass=''AND eid=

The result would be all the data in accounts table because the condition of the WHERE clause is always true.

Illegal/Logical Incorrect queries*:* When a query is rejected an error message is returned from the database including useful debugging information. This information helps attackers to make move further and find vulnerable parameters in the application and consequently database of the application.

Example
SELECT * FROM Accounts WHERE user=' ' AND pass=''AND eid =convert(int,(SELECT TOP 1name FROM sysobjects WHERE xtype='u'))

In the example the attacker attempts to convert the name of the first user defined table in the metadata table of the database to 'int'. This type conversion is not legal therefore the result is an error which reveals some information that should not be shown.

Union queries: In this type of queries unauthorized query is attached with the authorised query by using UNION clause.

Example
SELECT * FROM Accounts WHERE user='' UNION SELECT *FROM Students'AND pass=''AND eid=

The result of the first query in the example given above is null and the second one returns all the data in students table

Piggy-Backed query: In the query attack attacker tries to add an additional queries in to the original query string. In this injection the intruders exploit database by the query delimiter, such as ";", to append extra query to the original query

Example
SELECT*FROM Accounts WHERE user='';drop table Accounts—'AND pass=' ' AND eid=

The result of the example is losing the credential information of the accounts table because it would be dropped branch from database.

Inference: In this type of attack, intruders change the behaviour of a database of application. These are the well known types of inference

Blind Injection: This is little difficult type of attack for attacker. During the development process sometime the developer hides some error details which help the attacker to compromise with database. In this situation the attacker face the generic page provided by developer in place of an error message

Example
SELECT * FROM Accounts WHERE user='user1'AND1=1 'AND pass=' 'AND eid=

During injection it is always evaluated as true if there are no any error message, and the attacker realizes that the attack has passed user parameter is vulnerable to injection.

Alternate encoding**:** In this type of attack the regular strings and characters are converted into hexadecimal, ASCII and Unicode. Because of this the input query is escaped from filter which scans the query for some bad character which results SQLIA and the converted SQLIA is considered as normal query.

Example
SELECT * FROM Accounts WHERE user='user1';
exec(char(0x8774675u8769e)) - -' AND pass=' ' AND eid=

The example char () function and ASCII hexadecimal encoding are used. The functions will get integer number as a parameter and return as a sample of that character. In the example it will return "SHUTDOWN", so whenever the query is interpreted the SHUTDOWN command is executed.

III. LITERATURE SURVEY

Code based detection techniques**:** This approach generally occupies for developing test suit based on codes for detecting the SQLI vulnerabilities. But the suit does not find vulnerable program points explicitly.

SQLUnitGen is a prototype tool that uses static analysis tool to generate the user input to database access point and generate unit test report contacting SQLIA patterns for these points.

Concrete attack generations: This type of approach uses state of art symbolic execution techniques to automatically generate test inputs that expose SQLI vulnerability in Web program. The symbolic execution based approaches use constraint solvers that can only handle numeric operation. Because inputs of Web applications are string by default. If a constraint solver can solve myriad string operations applied to inputs, developers could use symbolic execution to both detect the vulnerability of SQL statements that use inputs and generate concrete inputs that attack them.

Learning based prevention**:** This approach is based on a runtime monitoring system deployed between the application server and database server, it intercept all queries and check SQL keywords to determine whether the queries syntactic structure are legitimate before the application sends them to the database.

In 2008, Mehdi Kiani et al.[2] describe an anomaly based approach which utilizes the character distribution of certain sections of HTTP requests to detect previously unseen SQL injection attacks. Their approach requires no user interaction, and no modification of, or access to, either the backend database or the source code of the web application itself. Their practical results suggest that themodel proposed in this paper is superior to existing models at detecting SQL injection attacks. They also evaluate the effectiveness of their model at detecting different types of SQL injection attacks.

In 2010, Atefeh Tajpour et al. [3] suggest that Database driven web application are threaten by SQL Injection Attacks (SQLIAs) because this type of attack can compromise confidentiality and integrity of information in databases. Actually, an attacker intrudes to the web application database and consequently, access to data. For stopping this type of attack different approaches have been proposed by researchers but they are not enough because usually they have limitations. Indeed, some of these approaches have not implemented yet and also most of implemented approaches cannot stop all type of attacks. Authors evaluate these approaches against all types of SQL injection attacks and deployment requirements.

In 2012, Ramya Dharam et al. [4]present a framework which can be used to handle tautology based SQL Injection Attacks using post-deployment monitoring technique. Their framework uses two pre-deployment testing techniques i.e. basis path and data flow testing techniques to identify legal execution paths of the software. Runtime monitors are then developed and integrated to observe the behavior of the software for identified execution paths such that their violation will help to detect and prevent tautology based SQL Injection Attacks.

In 2013, AmirmohammadSadeghian et al. [5] first they provided background information on this vulnerability. Next they present a comprehensive review of different types of SQL injection attack. For each attack they provide an example that shows how the attack launches. Finally they propose the best solution at development phase to defeat SQL injection and conclusion.

IV. PROPOSED SYSTEM

A. Algorithm

Inputs:

SQLo: Original query
SQLn: Query created for selecting user input fields
resulto: result variable of SQLo
resultn: result variable of SQLn
Ui : user inputs
indexi: user input field index of each U

Output: *Result of SQLI*

1. Execute SQLo and Execute SQLn
2. Call function SQLIdet() with input parameters for the query, Ui and indexi
3. Execution of function
   i. Compute no. of rows in result
   ii. If no. of rows>0 then
   Fetch first row of resultn from database
   Compare fetched data of $indexi column with Ui
   If both matches
   Then perform required action
   Else print “injection”
   iii. Else return
4. *End*

B. Least Privileges

To minimize the potential damage of a successful SQL injection attack, you should minimize the privileges assigned to every database account in your environment. Do not assign DBA or admin type access rights to your application accounts.

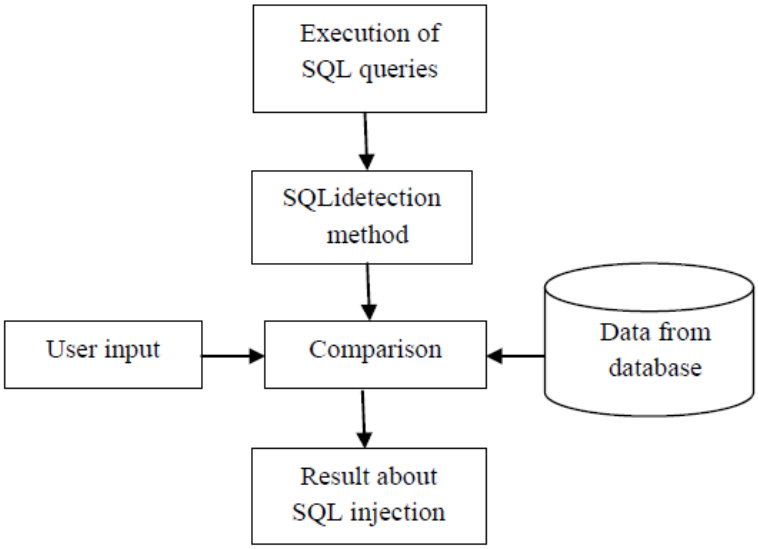
SQL injection is not the only threat to your database data. Attackers can simply change the parameter values from one of the legal values they are presented with, to a value that is unauthorized for them, but the application itself might be authorized to access. As such, minimizing the privileges granted to your application will reduce the likelihood of such unauthorized access attempts, even when an attacker is not trying to use SQL injection as part of their exploit.

As an example, a login page requires read access to the username and password fields of a table, but no write access of any form (no insert, update, or delete). However, the sign-up page certainly requires insert privilege to that table; this restriction can only be enforced if these web apps use different DB users to connect to the database.

C. Whitelist Input Validation

Input validation can also be a secondary defense used to detect unauthorized input before it is passed to the SQL query. Validated data is not necessarily safe to insert into SQL queries via string building.

V. PROPOSED ARCHITECTURE



VI. RESULTS

The web application is implemented in J2EE using servlets and MySQL as backend database is chosen for experiment. This application is experimented with number of requests in three different iterations as shown in Table 1. The result of these iterations are shown as graphical representation in the Fig 2

Table: 1 Valid and Infected Request Details

No. of Requests	Valid Request	Infected Queries
500	430	70
700	600	100
1000	810	190

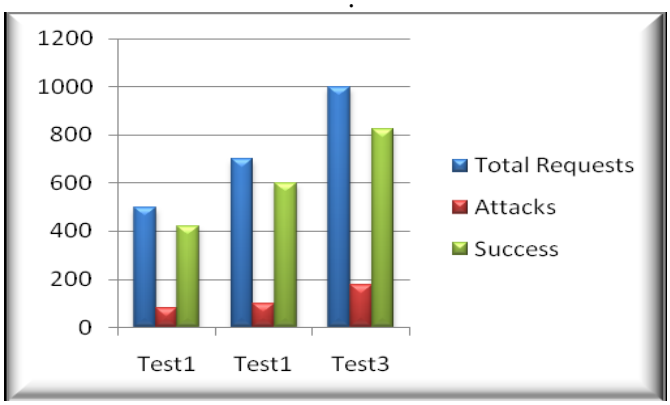


Fig. 2 Performance Analysis

The Fig 2 shows that the valid request are greater than the attacks. So by using the query filter through whitelist can prevent considerable amount of Injection attacks

VII. CONCLUSION

In this paper various types of SQL injection mechanis, detection type and prevention techniques The system with attack filters has been successfully developed and its performance has been analysed. This algorithm can able to proof its performance with any number of inputs.

REFERENCES

[1] Manish Kumar , L.Indu, *Detection and Prevention of SQL Injection attack*, International Journal of Computer Science and Information Technologies, Vol. 5, 2014.

[2] Mehdi Kiani, *Evaluation of Anomaly based character Distribution Models in the detection of SQL injection Attacks*", IEEE Conference, 2008.

[3] Tajpour, A.; JorJorZadeShooshtari, M., "*Evaluation of SQL Injection Detection and Prevention Techniques,*" Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on , vol., no., pp.216,221, 28-30 July 2010.

[4] Dharam, R.; Shiva, S.G., "Runtime monitors for tautology based SQL injection attacks," Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on , vol., no., pp.253,258, 26-28 June 2012.

[5] Sadeghian, A.; zamani, M.; Abdullah, S.M., "A Taxonomy of SQL Injection Attacks," Informatics and Creative Multimedia (ICICM), 2013

[6] Tejinderdeep Singh Kalsi, Navjot Kaur ," *Detection And Prevention Of Sql Injection Attacks Using Novel Method In Web Applications"* International Journal of Advanced Engineering Technology,2015

[7] Kai-Xiang Zhang; Chia-Jun Lin; Shih-Jen Chen; Yanling Hwang; Hao-Lun Huang; Fu-Hau Hsu, *"TransSQL: A Translation and Validation- Based Solution for SQL-injection Attacks*," Robot, Vision and Signal Processing (RVSP), Nov. 2011.

[8] Justin Clarke, SQL Injection Attacks, Syngress Defence, 2nd Edition, 2012

AUTHOR PROFILE

Angel N, received her MCA degree from Standard Fireworks College for Women affiliated to Madurai Kamaraj University, M.E(CSE) degree from Sathyabama University.
She is currently working as a Associate Professor in the Department of Computer Science and Engineering in St.Joseph's College of Engineering, Chennai. Her area of interest includes Web Security, Advanced Java Programming, Component Based Technology.

Chandra Sekar A, received his B.E(CSE) degree from Angala Amman College of Engineering and Technology affiliated to Bharathidasan University, M.E(CSE) degree from A.K. College of Engineering affiliated to Madurai Kamaraj University, and Ph.D in Information and Communication Faculty (Computer science & Engineering) from Anna University, Chennai, India.
He is currently working as a Professor in the Department of Computer Science & Engineering in St.Joseph's College of Engineering, Chennai. His area of interest includes Network Security and Analysis of Algorithms.

