Asian Journal of Computer Science and Technology
ISSN: 2249-0701 (P) Vol.7 No.S1, 2018, pp.11-15

DOI: https://doi.org/10.51983/ajcst-2018.7.S1.1814

SQL Injection Attack on Web Application

S. Parameswari[1] and K. Kavitha[2]
[1]Research Scholar, [2]Assistant Professor
[1&2]Department of Computer Science and Technology, Annamalai University, Chidambaram, Tamil Nadu, India
E-Mail: eswari105@gmail.com, kavithacseau@gmail.com
(Received 8 September 2018; Revised 20 September 2018; Accepted 7 October 2018; Available online 14 October 2018)

***Abstract* - SQL injection attacks are one of the highest dangers for applications composed for the Web. These attacks are dispatched through uncommonly made client information on web applications that utilizes low level string operations to build SQL queries. An SQL injection weakness permits an assailant to stream summons straightforwardly to a web application's hidden database and annihilate usefulness or privacy. In this paper we proposed a simplified algorithm which works on the basic features of the SQL Injection attacks and will successfully detect almost all types of SQL Injection attacks. In the paper we have also presented the experiment results in order to acknowledge the proficiency of our algorithm.**
***Keywords:* SQL Injection, Hacking, Authentication, Back Tracking, Intrusion, SQL Queries**

I. INTRODUCTION

SQL injection vulnerabilities have been portrayed as a standout amongst the most genuine dangers for Web applications [3, 11]. Web applications that are defenseless against SQL injection may permit an assailant to increase complete access to their fundamental databases. Since these databases regularly contain touchy customer or client data, the subsequent security infringement can incorporate wholesale fraud, loss of secret data, and misrepresentation. Sometimes, aggressors can even utilize a SQL injection weakness to take control of and degenerate the framework that has the Web application. Web applications that are powerless against SQL Injection Attacks (SQLIAs) are far reaching—a study by Gartner Group on more than 300 Internet Web locales has demonstrated that the vast majority of them could be helpless against SQLIAs. Truth be told, SQLIAs have effectively focused on prominent casualties, for example, Travelocity, FTD.com, and Guess Inc.

SQL injection alludes to a class of code-injection attacks in which information given by the client is incorporated into an SQL question in a manner that part of the client's data is dealt with as SQL code. By utilizing these vulnerabilities, an aggressor can submit SQL charges straightforwardly to the database. These attacks are a genuine risk to any Web application that gets information from clients and fuses it into SQL questions to a basic database. MostWeb applications utilized on the Internet or inside big business frameworks work along these lines and could thus be defenseless against SQL injection.

The reason for SQL injection vulnerabilities is generally basic and surely knew: deficient approval of client info. To address this issue, engineers have proposed a scope of coding rules (e.g., [18]) that advance protective coding practices, for example, encoding client data and acceptance. A thorough and efficient use of these methods is a compelling answer for avoiding SQL injection vulnerabilities. Be that as it may, practically speaking, the application of such methods is human-based and, in this manner, inclined to blunders. Moreover, settling legacy code-bases that may contain SQL injection vulnerabilities can be a greatly work concentrated assignment.

SQL injection attacks represent a genuine security risk to Web applications: they permit assailants to get unlimited access to the application and to the conceivably delicate data its databases contain. In spite of the fact that scientists and professionals have proposed different strategies to address the SQL injection issue, current methodologies either neglect to address the full extent of the issue or have confinements that keep their utilization and selection.

Key Concepts of SQL Injection:

1. SQL injection is a product defenselessness that happens when information entered by clients is sent to the SQL mediator as a part of a SQL question.

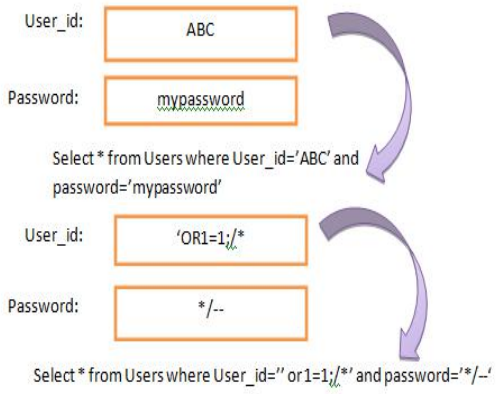

Fig. 1 SQL Injection

2. Attackers give exceptionally created info information to the SQL translator and trap the mediator to execute unintended charges.

3. Attackers use this weakness by giving exceptionally made information to the SQL mediator in such a way, to the point that the translator is not ready to recognize the proposed summons and the aggressors extraordinarily made information. The mediator is deceived into executing unintended summons.

4. SQL injection abuses security vulnerabilities at the database layer. By misusing the SQL injection defect, aggressors can make, read, adjust, or erase touchy information.

II. SQL INJECTION ATTACKS TYPES

There are different methods of attacks that depending on the goal of attacker are performed together or sequentially. For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query. Now the following classification of SQLIAs will be presented.

*A. Tautologies:*This type of attack injects SQL tokens to the conditional query statement to be evaluated always true. This type of attack used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause. “SELECT * FROM employee WHERE userid = '112' and password ='aaa' OR '1'='1'“ As the tautology statement (1=1) has been added to the query statement so it is always true.

*B. Illegal/Logically Incorrect Queries:*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 errors by purpose.In this example attacker makes a type mismatch error by injecting the following text into the pin input field:

Original URL:
http://www.arch.polimi.it/eventi/?id_nav=8864

SQL Injection:
http://www.arch.polimi.it/eventi/?id_nav=8864'

Error message showed:
SELECT name FROM Employee WHERE id =8864\'

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.

*C. Union Query:*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. Suppose for our examples that thequery 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 creditCardNumber,1 FROM CreditCarTable

We will have the following query:
SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1
FROM CreditCarTable which will join the result of the original query with all the credit card users.

*D. Piggy-backed Queries:*In this type of attack, intruders exploit database by the query delimiter, such as “;”, to append extra query to the original query. With a successful attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate. So attacker can inject any SQL command to the database. In the following example, attacker inject “ 0; drop table user “ into the pin input field instead of logical value. Then the application would produce the query: SELECT info FROM users WHERE login='doe' AND pin=0; drop table users Because of “;” character, database accepts both queries and executes them. The second query is illegitimate and can drop users table from the database. It is noticeable that some databases do not need special separation character in multiple distinct queries, so for detecting this type of attack, scanning for a special character is not impressive solution.

*E. Stored Procedure:*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, attacker exploits parameterized stored procedure.

CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC(“SELECT accounts FROM users WHERE login=‘“ +@userName+ “‘ and pass=‘“ +@password+ “‘ and pin=“ +@pin); GO For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input — ‘SHUTDOWN; - -‖ for username or password.

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.

*F. Inference:*By this type of attack, intruders 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.

1. *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:

SELECT accounts FROM users WHERE login='doe' 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 submit 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.

III. RELATED WORK AND LITERATURE SURVEY

Ke Wei *et al.,* [1] proposed a novel system to guard against the attacks focused at stored procedures. This strategy joins static application code investigation with runtime approval to kill the event of such attacks. In the static section, a put away technique parser is planned, and for any SQL proclamation which relies on upon client inputs, this parser is utilized to instrument the fundamental articulations keeping in mind the end goal to contrast the first SQL explanation structure with that including client inputs. The sending of this system can be robotized and utilized on a need-just premise.

William G.J. Halfond *et al.,* [2] displayed a broad audit of the diverse sorts of SQL injection attacks known not. For every sort of assault, portrayals and cases of how attacks of that sort could be performed are given. He also presented and broke down existing discovery and aversion systems against SQL injection attacks. For every system, its qualities and shortcomings are talked about in tending to the whole scope of SQL injection attacks.

William G.J. Halfond *et al.,* [3] proposed another exceptionally computerized approach for element discovery and counteractive action of SQLIAs. Instinctively, this methodology works by recognizing "trusted" strings in an application and permitting just these trusted strings to be utilized to make the semantically important parts of a SQL inquiry, for example, watchwords or administrators. The general component that we use to actualize this methodology depends on element polluting, which checks and tracks certain information in a project at runtime.

SruthiBandhakavi *et al.,* [4] proposed a straightforward and novel component, called Candid, for mining developer expected inquiries by powerfully assessing keeps running over benevolent competitor inputs. This component is hypothetically very much established and depends on surmising proposed inquiries by considering the typical inquiry registered on a system run. This methodology has been actualized in an apparatus called Candid that retrofits Web applications written in Java to protect them against SQL injection attacks.

Jin-Cherng Lin *et al.,*[5] introduced a propelled proposition embracing the idea of utilization level security entryway and more successfully determining the issue than comparable doors or intermediaries. This framework comprises of discovery testing, acceptance capacities and redirection instrument.

Mehdi Kiani *et al.,* [6] portrayed an irregularity based methodology which uses the character conveyance of certain segments of HTTP solicitations to recognize already inconspicuous SQL injection attacks. This methodology requires no client collaboration, and no alteration of, or access to, either the backend database or the source code of the web application itself. Its commonsense results recommend that the model proposed in this paper is better than existing models at distinguishing SQL injection attacks. Specialists additionally assess the adequacy of the model at recognizing distinctive sorts of SQL injection attacks.

Yu Chin Cheng *et al.,* [7] proposed a sort of novel Embedded Markov Model (EMM) to recognize diverse web application attacks, screen the on-line client conduct and safeguard the vindictive client immediately. Contrasting with past web application attacks distinguishing approaches, this EMM methodology can identify client's refuted data mistakes as well as discover the nonsensical page move conduct. By identifying outlandish page move, we can quickly safeguard the pernicious or senseless client conduct to maintain a strategic distance from the further web framework disappointments and touchy data exposure.

HossainShahriar *et al.,* [8] proposed a change based testing approach for SQLIV testing. He proposed nine transformation administrators that infuse SQLIV in application source code. The administrators result in mutants, which can be slaughtered just with test information containing SQL injection attacks. By this methodology, they constrained the era of sufficient test information set containing powerful experiments equipped for uncovering SQLIV. They executed a Mutation-based SQL Injection vulnerabilities checking (testing) device (MUSIC) that consequently produces mutants for the applications written in Java Server Pages (JSP) and performs transformation investigation.

NunoAntunes *et al.,* [9] proposed another programmed approach for the identification of SQL Injection and X-Path Injection vulnerabilities. In this approach an agent workload is utilized to practice the web administration and a vast arrangement of SQL/X-Path Injection attacks are connected

to uncover vulnerabilities. Vulnerabilities are recognized by looking at the structure of the SQL/X-Path charges issued in the nearness of attacks to the ones beforehand realized when running the workload without attacks. This methodology performs much superior to anything known apparatuses (counting business ones), accomplishing to a great degree high identification scope while keeping up the false positives rate low.

Dwen-RenTsai *et al.,* [10] proposed an ideal tuning technique using the application firewall generally utilized by the cutting edge endeavours. They investigated a few assaulting techniques generally utilized these days, for example, the mark of cross-site scripting and SQL injection, and acquainted another strategy with setup the parameters of the gadget to reinforce the barrier. To improve the security of the back-end application servers, they utilized catchphrase sifting and re-treatment to administer through the boycott, and to conform the framework settings to the goal that it can adequately hinder the ambushes or decrease the likelihood of fruitful attacks. What's more, they additionally re-enacted attacks to web searching and application through helplessness checking instruments to test the security of utilization framework and to ensure the important protection of the ideal tuning parameters.

Ivano Alessandro *et al.,* [11] introduced an exploratory assessment of the viability of five SQL Injection identification instruments that work at various framework levels: Application, Database and Network. To test the devices in a sensible situation, Vulnerability and Attack Injection is connected in a setup in light of three web uses of various sizes and complexities.

Xin Wang *et al.,* [12] proposed a component of SQL injection helplessness location in view of concealed web slithering and actualize a distinguishing framework with the motivation behind raising the website page scope and upgrading the SQL injection powerlessness recognizing capacity of web scanner. Analysts joined verification with the crawler model, and discovered SQL injection helplessness by mimicking web assaulting and breaking down the information of reaction.

TIAN Wei *et al.,* [13] proposed another testing strategy for chasing SQL injection, in which the injection parameters can be separated into a few arrangements of identicalness classes as per the characterized multi-barrier levels of test web frameworks. By infusing the most illustrative parameters chose from every proportionality classes, the fake assault testing for chasing SQL injection can be extremely successful and minimal effort.

Zhang Xin-hua *et al.,*[14] proposed static examination apparatuses ASPWC to recognize XSS attacks and SQL injection vulnerabilities in light of spoil investigation, It tracks different sorts of outside info, labels pollute sorts, building control stream chart is developed taking into account the utilization of information stream investigation of the important data, corrupt information spread to different sorts of weakness capacities, and distinguish the XSS or SQL Injection helplessness in web application's source code. Tests demonstrated that the identification methodology is a compelling way; it can be utilized to identify the XSS and SQL Injection weakness in the web application program in light of ASP innovation improvement.

Lijiu Zhang *et al.,* [15] proposed a novel way to deal with recognize web application vulnerabilities. In this methodology, given a URL, Researchers get an objective web structure. Subsequent to examining attributes of this web structure, Researchers allocate an arrangement of test qualities to each held in this structure. At that point they proposed a technique to produce test suites considering the heaviness of every test esteem. At long last, they executed those test suites and broke down comparing result in light of HTTP reaction code and reaction HTML. They implemented this approach into an instrument called D-WAV and picked a few web applications as benchmarks to lead observational studies. Last results demonstrated that this methodology can naturally and viably find web application vulnerabilities, for example, cross-website scripting and SQL injection.

IV. METHODOLOGY

A new algorithm is presented to protect Web applications or even the desktop application against SQL injection Attacks. SQL Injection Attacks are a class of attacks that many of these systems are highly vulnerable to, and there is no known foolproof defense against such attacks. Some predefined methods and integrated approach of encryption method with secure hashing can be applied in the database to avoid attack on login phase. This combined method will be applied to a system where user‘s information is kept and the designing of this system will be done by using .Net.

A. Algorithm Proposed: In your proposed concept we have proposed an algorithm, which will be used for performing a check that the query fired by the user is an SQL Injection or not.

The algorithm contains the following steps:

1. First the Query is provided as input in the form which we created for the Query Analysis.
2. In the First Check the Query is check for the DROP keyword as, to avoid SQL Injection which can delete the table structure.
3. In the Second check we check for the validity of the SQL statement, in order to check whether it is proper SQL statement i.e. begins with SELECT, INSERT etc.
4. In the third check we will avoid the SQL Injection for the value '1'='1' , this type of injection can be given in various ways , so we implemented this in two sub section , firstly containing OR statement , where we split the query on the basis of OR keyword and then checked the parameters for similarity and if same then it SQL Injection Attack and second a simple Query

which contains only statements like '1'='1' is handled after checking presence of = and checking parameters for equality.

5. Then we have check for the queries with intension of knowing the tables in the databases.
6. Finally we have checked the queries with have no results, just fired in order to know the table structure.

V. EXPERIMENTAL RESULTS

In this we have performed some SQL Injection queries on the simple unprotected interface and on the algorithm which is proposed by us. Below mentioned table presents the result of the analysis which we have performed.

TABLE I RESULT OF ANALYSIS WE HAVE PERFORMED

Pattern	String Pattern	Expected Result	
#		Secure	Insecure
1	_OR‖=‘	Login failed	Login Successful
2	0‘ or _1‘=‘1	Login failed	Login Successful
3	1‘ or _1‘=‘1	Login failed	Login Successful
4	_ OR _1‘=‘1‘	Login failed	Login Successful
5	1‘ or _a‘=‘a	Login failed	Login Successful
6	; and 1=1 and 1=2	Login failed	Login Successful
7	—‘ or 1=1 - -—	Login failed	Login Successful
8	OR '1'='1'“	Login failed	Login Successful
9.	emp_id= 'x' AND emp_name IS NULL	Attack Identified	Columns retrieved
10.	Select * From Employee; Drop TableEmployee	Attack Identified	Table Employee Deleted
11.	Select Table_Name FromInformation_Schema.Tables	Attack Identified	Table Names in thedatabase

VI. CONCLUSION

SQL-Injection is a relatively simple technique and on the surface protecting against it should be fairly simple. Auditing all of the source code and protecting dynamic input is not trivial, neither is reducing the permissions of all application users in the database itself. It is possible to develop a filter to prevent SQL-Injection. Checking through log files, making sure that code is perfectly secure and relying on the least privileges principle does not seem sufficient. It is difficult to detect attacks and again, an audit of log is required. The use of packet sniffers does not allow for the prevention of damage as the packets collected do not allow for the removal of malicious SQL query statements. Provide useful information on the impact of the TDSProxy on web interface usage. The future enhancement of this paper is extended to handle other databases such as MySQL, Oracle and Postgres as well as other operating systems and involve an investigation into the performance impact of the proxy server on data transfer.

REFERENCES

[1] A. Petukhov and D. Kozlov, “Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing, “*Proceedings of Application Security Conference, Ghent, Belgium*, 19-22 May, 2008.

[2] K. Ahmad, J. Shekhar, and K.P. Yadav, “A Potential Solution to Mitigate SQL Injection Attack,”*VSRD Technical & Non-Technical Journal*, Vol.1, No. 2, pp. 145-152, 2010.

[3] B. Indrani and E. Ramaraj, “An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service,”*IJCSNS International Journal of Computer Science and Network Security*, Vol.11 No.1, January 2011.

[4] P. Ramasamy and S. Abburu,“SQL Injection Attack Detected and Prevention,”*International Journal of Engineering Science and Technology (IJEST)*, Vol. 4, No.04, April 2012.

[5] S. Manmadhan and Manesh T, “A Method of Detecting SQL Injection Attack to Secure Web Applications,”*International Journal of Distributed and Parallel Systems (IJDPS)*, Vol.3, No.6, Nov. 2012.

[6] L. Kishori and K. Sunil, “Detection and Prevention of SQL-Injection Attacks of Web Application Using Comparing Length of SQL Query,” Vol. 1, February, 2012.

[7] A.Keromytis, and V. Prevelakis, “Countering codeinjection attacks with instruction-set randomization in Proceedings of the 10th ACM,”*Conference on Computer and Communication Security Washington D.C.*, pp. 272-280.

[8] BojkenShehu, and AleksanderXhuvani“A literature Review and comparative analysis on SQL injection: Vulnerabilities, attacks and their detection and prevention Techniques”*International Journal of Computer Science Issues*, Vol. 11, No. 1, 2014

[9] Hussein AlNabulsi, IzzatAlsmadi, and Mohammad AlJarrah“Textual Manipulation for SQL Injection attack” I.J. computer Network and Information Security, 2014

[10] F. Valeur, D. Mutz, and G. Vigna“A learning-based approach to the detection of SQL attacks”*LNCS*, Vol. 3548, pp. 123-140, 2005.

[11] A. Anitha, and V. Vaidehi, “Context based Application Level Intrusion Detection System” in Washington, DC, *USA: IEEE Computer Society*, pp. 16, 2006.

[12] L. Chen, Z. Li, C. Gao, and Y. Liu, “Modeling and Analyzing Dynamic Forensics System Based on Intrusion Tolerance” in Washington, DC, USA:*IEEE Computer Society*, pp. 230-235, 2009.

[13] C.J. Ezeife, J. Dong, and A.K. Aggarwal, “SensorWebIDS: A Web Mining Intrusion Detection System”, *International Journal of Web Information Systems*, Vol. 4, pp. 97-120, 2007.

[14] E.Bertino, A.Kamra, and J. Early, “Profiling Database Application to Detect SQL Injection Attacks”, *In the Proceedings of 2007 IEEE International Performance, Computing, and Communications Conference, 2007*

[15] William G.J. Halfond, Alessandro Orso, and Panagiotis Manolios, “WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation”, *IEEE Transactions on Software Engineering*, Vol. 34, No. 1, pp. 65-81, 2008.