An Automatic Detection System for SQL Injection

Divya Jain
Department of Computer Engg.
College of Technology and
Engineering, Udaipur, India

Naveen Choudhary
Department of Computer Engg.
College of Technology and
Engineering, Udaipur, India

ABSTRACT
The growth of the internet is increasing day by day, mostly content is database driven. There are many web applications like E-Commerce, banking where he/she has to trust on this application and have to provide personal information into their underlying database. If there is no confidentiality and security of information then any one can steal or see our information or may utilize this information for misbehaving activity. One of them is SQL injection, a hacker may insert his bad/malicious SQL code into other's database and running of those queries is capable to extract private and valuable information or may destroy the database. In this paper, proposing a technique to detect SQL injection using the hidden web crawling technique incorporating with parse tree and digital signature. The proposed scheme finds a SQL injection vulnerability by replicating web attack and analyze the data of the response. The proposed technique is compared with hidden web crawling technique to analyze its's effectiveness. For experimental evaluation, implement this system in Eclipse with MYSQL database to analyze the results.

Keywords
SQL injection, Hidden web Crawling, Parse tree, Digital Signature.

1. INTRODUCTION
At present internet is a very important source of information and communication channel between user and service providers. As using of web application is increasing, there is increase of web attack also. One of them is SQL injection attacks (SQLIA), this vulnerability may lead to unauthorized access of resources, escalation of privileges and loss of confidentiality and integrity. Recently the incident of SQLIA has been so high that a survey done by new IBM-X Force Threat Intelligence [3] for year 2014 almost 10 % increase in security attacks on business which leaks one billion records of personal identifiable information (PII) were leaked. All these attacks are due to SQLIAs and other cyber attacks.

SQL is one of the web attack used by hackers to swipe data from organizations. It is an application layer attack. In this mechanism, malicious SQL command is executed by the web application, exposing the backend database. An SQL injection attack can occur when a web application utilizes user supplied data without proper validation or encoding as part of a SQL query. Injected SQL interdiction can reforms SQL statement and encompasses the security of web application.

As shown in Fig. 1 attackers inject malicious SQL code and retrieve personal information. In this a simple web page where the user has to provide his user id and password to login in "form.jsp" which is passed through a firewall, web server, application server and finally to database server.

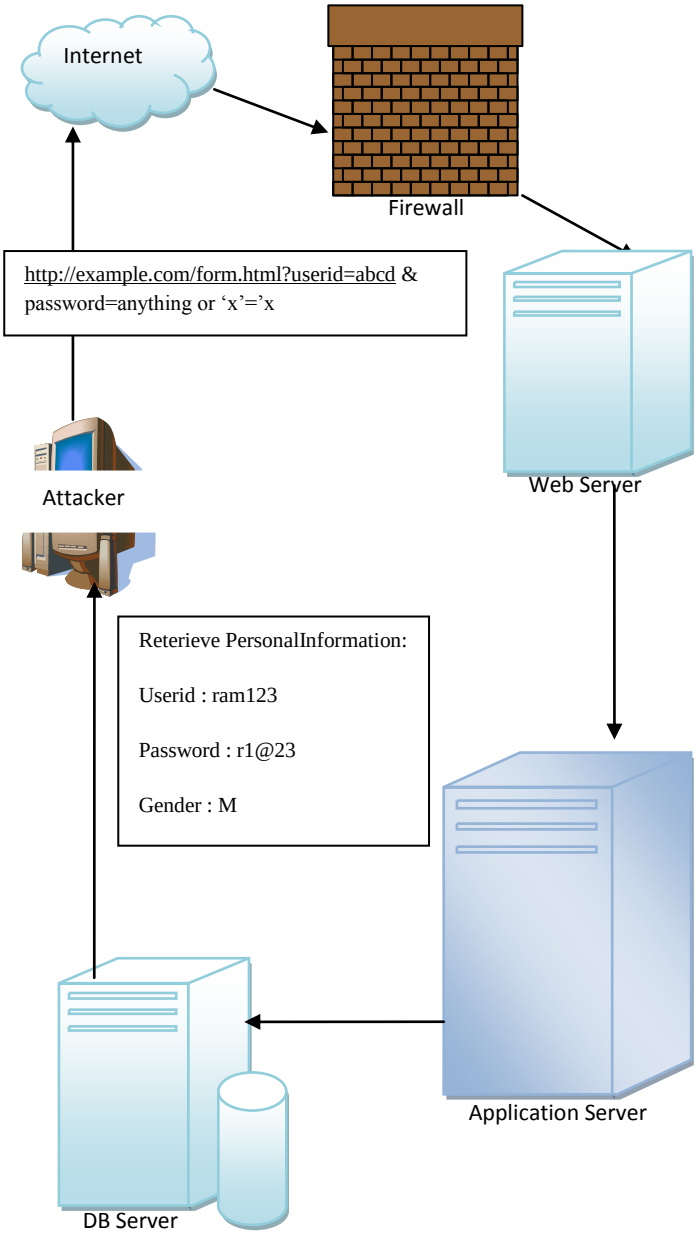

Fig. 1: SQL injection

SQLIAs is not necessarily prevented by firewall and intrusion detection system because the websites need to be public, security mechanism allows public web traffic to communicate with web application (generally run over 80/443).

In this paper, incorporating hidden web crawling [4, 6, 11] with parse tree [5, 17] and digital signature to not only detection of SQL injection but also prevention at run time so the objectives are enumerated as follows:

- In hidden web crawling technique incorporating the detection of SQL injection at run time, analysis of hidden web crawling technique is fed into running time detection system.
- To improve the authentication use digital signatures, which improve scalability of the system.
- Use the parse tree to detect suspected vulnerability with a new proposed approach.
- Implement this system and compare with hidden web crawling to analyse the results.

To evaluate this approach we test over PHP web application [12] to detect SQL injection as true positive, false positive and false negative results.

The organization of the paper as follows in section 2 description of types of SQL injection attacks, in section 3 Hidden web crawling technique and SQL injection, in a section 4 parse tree technique and SQL injection, in section 5 proposed methodology, in section 6 conclusion the future work.

2. TYPES OF SQL INJECTIO ATTACKS

The basic types of SQL attacks [7] are as follows:

Tautologies based SQL attack:

Tautology means in every possible interpretation always calculates to true, this attack is injected by using conditional OR operator by which SQL query calculates to true. This attack bypassed the user authentication and extracts the data by inserting conditional OR operator in the WHER clause of a SQL query. It will reshape the SQL query into tautology by which database will be exposed to an unauthorized user. If an attacker inserts in a query 'abcd' as password and anything' OR 'x'='x as password the query becomes:

Select * from userdetails where userid=‘abcd’ and password =’anything’ or ‘x’=’x’

On the basis of operator precedence rule, the WHERE clause is evaluated to true for one row, so the query will return whole records. By this an assailant will be able to access personal information of the user.

Piggybacked Queries attack:

As the name suggests that hacker injects additional query with original one by which database gets multiple SQL queries. In this method original query is valid, but another query is attacking query with first one. This type of query is allowed in one query due to miss configuration of a system. Suppose an attacker injects abcd as userid and'; drop table pqr-- as a password then the resulting query is:

Select * from userdetails where userid=‘abcd’ and password = “; drop table pqr--‘

In this original query executed normally returns zero rows, a query delimiter (";") is recognized by the database and executed the additional injected query. The consequences of this query will wipe out valuable information from the database.

Union Query:

The union query attack is done by introducing a UNION keyword into a vulnerable parameter which will return the union of original and injected query.

The SQL UNION operator fetched the results (rows) from participating queries. Suppose the code injected by an attacker is ‘UNION select * from empldetails-- in user id field and abcd in password field so the query becomes:

Select * from userdetails where userid = ‘’ UNION selects * from empldetails –‘ and password = ‘abcd’;

Using comment operator (--) will ignore the rest of the query, i.e. password = 'abcd'. So, in this query original query acknowledges a null set value as there is no matching details in the table userdetails and the injected query will return all the data from empdetails table.

Illegal/Logically Incorrect Queries:

In this type of injection this is pre attacking steps for more attacks; it means that collection of information about the type and structure of the database. In this method some error messages returned by the application server by analysing these messages, an attacker is able to take the advantage of this weakness. Sometime these logical error messages not only give the data type of certain columns, but also the name of the table and columns.

Inference:

In this method attacking code is applied to a secured database which does not give any logical error messages. This method normally works on the basis of true false statement. After collecting sensitive information, the assailants inject different conditions (how the database behaves as true or false means working or not on this injecting code) and determine the situation carefully. If the injecting code evaluates to true implies that page working is normally and if it is false means that page behaving is not normal. This type of attack is blind attack. Similarly to blind attack there is time attack. In this attack, an attacker tries to gather information of those parameters which are based on time delays in the response of a query or database.

http://www.example.com/product.php?product_id=100 AND if (version () like ‘5%’, sleep (15), ‘false’))--

Here in this attack, an attacker is determining the version of MYSQL is 5.X or not and also introduces a delay of 15 seconds to respond this query.

Stored Procedures:

In access relational database system, there is a subroutine called stored procedure and stored in the data dictionary. In this there is the definition of data validation and access control mechanism. In this centralized logic is built to access resources and complex queries are moved into a stored procedure. In this attack first an attacker uses pre attacking code to find the database type and version using illegal/logically incorrect queries. After finding this an attacker uses various procedures through injecting code. As the code of stored procedure is written by the developer, so these procedures are not vulnerable to SQLIAs. They may be vulnerable to provide the administrative access.

Suppose an assailant injects ‘; SHUTDOWN; -- into either the user id or password fields then the resulting query is:

Select* from userdetails where userid='abcd' and password = ''; SHUTDOWN; -- '

This query will cause the database to shut down.

Alternate Encodings:

In this method defensive coding is used by an assailant to bypass injected code which is encoded text. Encoding methods like hexadecimal, ASCII and Unicode character encoding. Scanner and detection techniques are not effective against such attacks. See the following illustration:

Select* from userdetails where userid= '' and password = ' '; exce (char (0x736875746446j776e)) '

Here in char () function ASCII hexadecimal encoding scheme is used; this will return the actual character of the hexadecimal encoded character. This encoded text means is shut down of database when this attacking code is executed.

3. HIDDEN WEB CRAWLER AND SQL INJECTION [6]

To detect SQL injection vulnerability in hidden web crawler is based on response analysis of a web page. On the basis of collected information by crawler, attacking code query is submitted to web servers then the behavior of page is analyzed whether the SQL injection is performed or not.

3.1 Strategy of Hidden Web Crawler

Now a day, users have to provide correct authentication information to web services, to access corresponding web services. This authentication information is utilized in hidden web crawling to improve the overall security detection system. This methodology is based on access authorization data table (AADT) which is 5 attributes information is follows:

Ai = (TOi, Hi, Ni, Ti, Vi) where TOi is target website address, Hi is hash value of target website, Ni is the name of authorization input form, Ti is type of form and last Vi is the used to save the value which is assigned to authorization input

For example, if target website URL is www.examplecode.com is detected, Ai and Ai+1 is calculated as follows:

Ai = (www.examplecode.com, be1e49a29c8d31ej187r, username, password, Jony)

Ai+1 = (www.examplecode.com, be1e49a29c8d31ej187r, Passfully, password, 123457)

Firstly AADT is established before traversing of target website. The analyzing engine of crawler identify all vulnerable spots or it can say collect all information where user submits his/her information. When page requires authentication information the AADT compute these 5 attribute information where Vi has default value then AADT match these values against At if matches successfully then it replace Vi's default value with its correct value and get to access web services. For response analysis is used such as cookies, session and so on. The crawler is recursively started to perform deep crawling on founding of any URL or hyperlink which improves overall detection. The strategy as shown in figure 2:

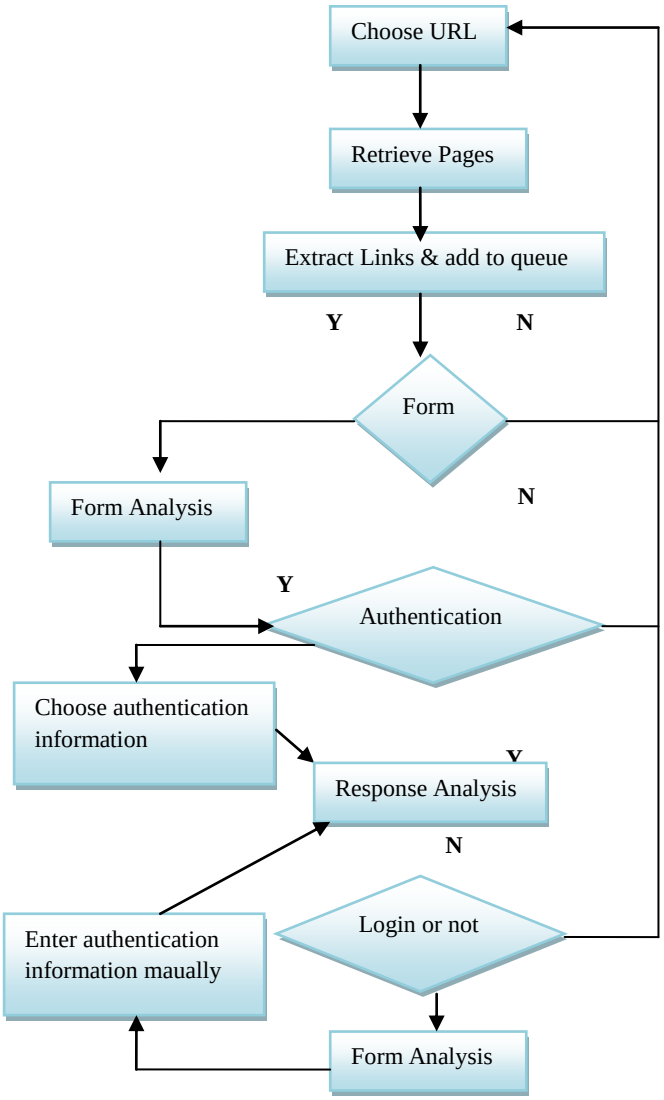

Fig 2: Hidden Web Crawling Strategy

3.2 Attacking Code & Response Analysis

To test this strategy attacking code is constructed to analyze the response as follows: If the response analyzing result shows that the SQL command executed invalidated by the values of "attacking code" injected by the attacker or if the values of "attacking code" lead to database logical exceptions raised by the database server. When there is no way to find out confirmed the result, then these are doubtful cases.

4. PARSE TREE AND SQL INJECTION DETECTION [5]

To detect SQL injection vulnerability in parse tree the SQL query is represented as a tree format. The grammar knowledge of statement is required for parsing. With the help of parse tree they determine whether the queries are same or not.

When an attacker submits any SQL attacking code to database server then the structure of attacking query is different from actual query as shown in fig. 3:

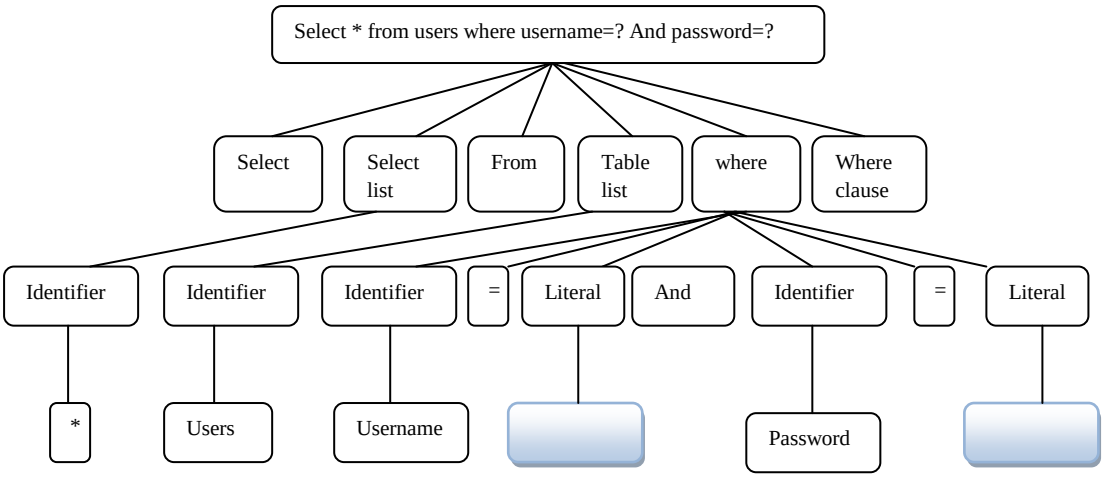

Fig 3a: Parse Tree of actual query

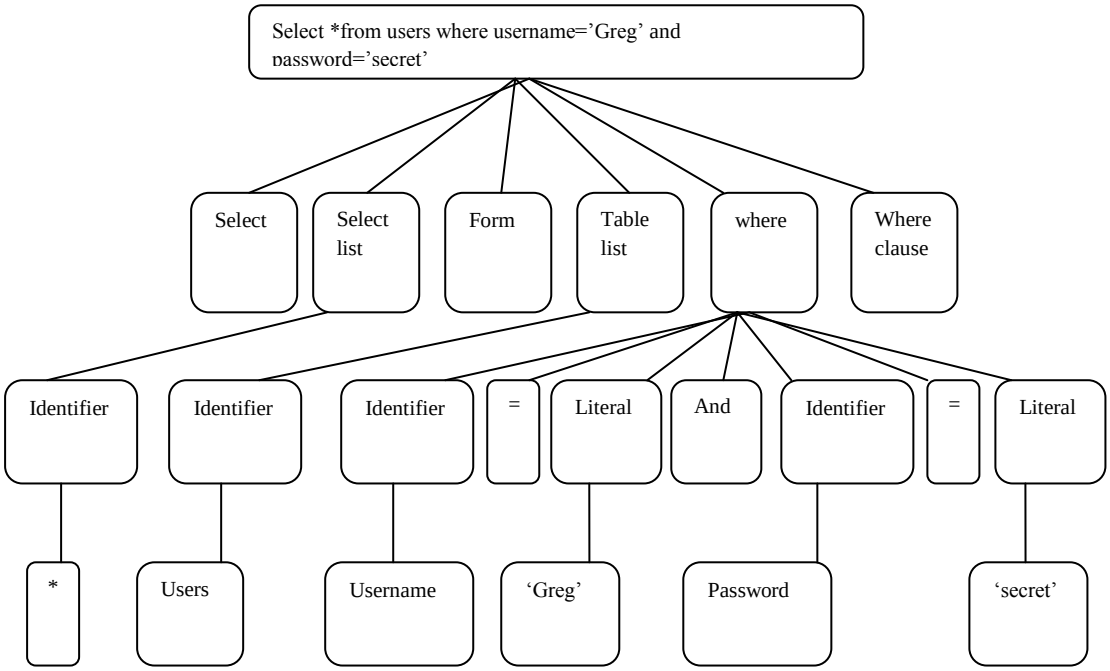

Fig 3b: Parse Tree of attacking query

Here attacking code means any modification or changes done to the original query or it can say crafting of user input. In parse tree user input are present as empty literal at leaf nodes of tree. When the input is supplied then the input is filled into empty leaf nodes. The value of leaf nodes must be in position and literal.

As shown in fig 3a the parse tree of a SQL query is **Select * from user where username = ? and password = ?.** These question marks are replaced by user supplied input by which comparison is made that structure of SQL query is same or not.

The SQLGuard class have the capability of string building and parsing so this class is used to implement this solution in java with 3 ms overhead. A fresh key is generated when any SQL string is prepended with SQLGuard.init(). For every query new key is generated because of loading of page. When any query is submitted to database server, with help of SQLGuard.wraps(s). It is first pre-postened with current key. By this way an attacker can't guess the key. The private method of SQLGuard class verify() is used to remove the key from beginning of query and use it to identify wrapped used input which is used for building for parse tree. After building of parse tree comparison is made on the basis of structure by which malicious query is detected.

5. PROPOSED METHODOLOGY AND SQLINJECTION

To detect SQL injection at run time incorporating hidden web crawling technique with parse tree and digital signature. Implement this system in Eclipse on Window 7, 3 GB RAM configuration with 2.40 GHz processor. The architecture of the proposed method is shown in figure 4:

In Evalustion phase, firstly go through a hidden web crawler to find out all links and vulnerable spots with digital signature, here using the digital signature for authorization of user instead of AADT table to improve scalability of system but the results at this stage have false positive and false negative. Comparision of the proposed scheme is done with the hidden web crawler [4, 6, 11] results and earlier tools ZAP [14] and Vega [15] to analyze it's effectiveness.

.

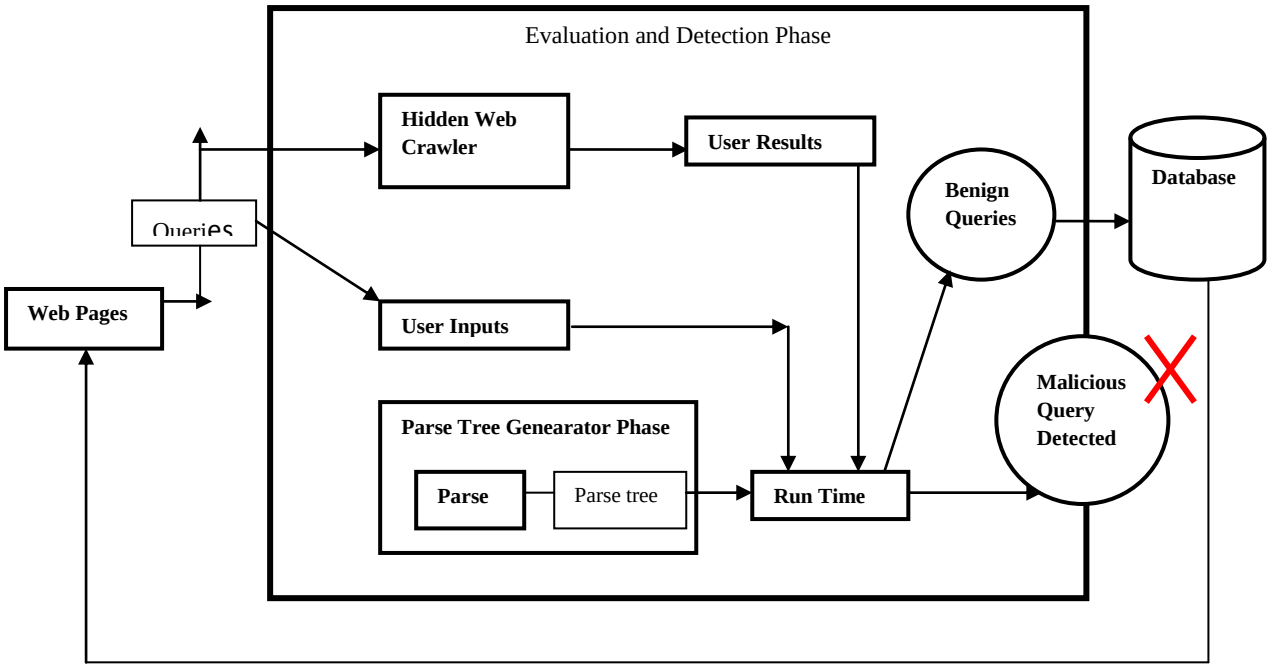

Fig 4: Evaluation and Detection phase of proposed methodology

At Detection phase, at run time parse tree technique is used to remove the suspected vulnerability in hidden web crawler. For this doing parsing of SQL statement before inclusion of user input after inclusion of input for PHP web application. Then combing both results to remove the suspected vulnerability by a hidden web crawler method in this way:

R= (A OR RSM) AND (SM AND RSM)

Here A is detection of vulnerability by hidden web crawler.

RSM is run time parsing of SQL statement.

SM is parsed of SQL statement before inclusion of input.

The proposed methodology has been tested over [12] web application with these attacking codes:

Table 1: Attacking code construct

Attacking code
1’ or ‘1
Anything’ or ‘x’=’x
x' OR user like '%r%
and 1’=’1
‘ or ‘x’=’x
x’ and email is null;--
‘ or 1=1--
x’; drop table members;--
%3b
%20and%20’1’=’1

5.1 Result and Response Analysis

Test of 10 attacking code against php web application [12] and for effectiveness of implemented system results comparing with not only hidden web crawler [4, 6, 11], but also other crawler tool ZAP [14], VEGA [15].The result analysis on response is done by this way:

- The result is true positive If the analysing result shows that the SQL command executed invalidated by the values of “attacking code” constructed by the detector.
- The result is false positive when attacking code leads to database exception error.
- Doubtful cases come into existence when there is no way to find out true positive and false positive results.
- False negative results are those which are not identified by the system.

Table 2: Results of Our System

	Total injected	True Positive	Doubtful	False Positive	False Negative
Our system	10	10	0	0	0
Hidden Web crawler	10	5	1	3	1
Scanner Zap	10	1	1	1	7
Scanner Vega	10	2	1	0	7

The solution is tested on php web application [12] with the list shown in table1 attacking code in which implemented system is able to detect all these vulnerability where as in hidden web crawler there is 1 false negative and 3 false positive and scanner ZAP and Vega has 1,0 and 7,7 as false positive and false negative respectively so implemented system is better than hidden web crawling. Implement this

system for PHP web application [12] and the total time overhead for this is 3114 ms which is greater than parse tree [5] technique.

The results in graphical format is as follows:

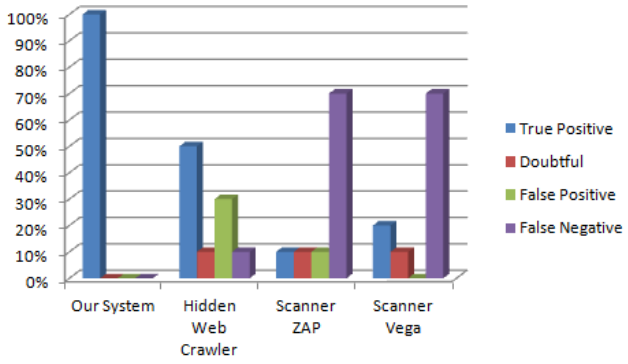

Fig. 6 Results of our system

In this it is seen that implemented system is able to detect all attacking code which are listed in table1, in hidden web crawler is able to detect 50% where as ZAP [14] and VEGA [15] are able to detect 10% ,10% respectively.

6. CONCLUSION AND FUTURE WORK

Most of all web application based on middleware technology, to retrieve information from relational database SQL. From the above results and graph discussion it can be say that implemented system is more secure where as hidden web crawler is able to detect 50% vulnerability and 10% by ZAP or 10% by VEGA. In proposed scheme time overhead increases. Implemented system provide a new approach to secure a web application. In near future we may enhance the algorithm used in hidden web crawler and parse tree to detect SQL injection.

7. REFERENCES

[1] Dwen, T., Chang, A., Liu, P. and Chen, H. 2009. Optimum Tuning of Defence Settings for Common Attacks on the Web Applications Security technology, 43rd Annual International CarnahanConference.

[2] Jovanovic, N., Kruegel, C., Kirda, E. 2006. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities Security and Privacy, IEEE Symposium.

[3] Website http://git.okt-srl.com/poste/0/43.

[4] Gupta, N., Kapoor,S. 2014. Extraction of Query Interfaces for Domain Specific Hidden Web Crawler International Journal of Computer Science and Infomation Technologies, Vol5 (1).

[5] Buehrer, G.,Weide, B., Sivilotti, P. 2005. Using Parse Tree Validation to Prevent SQL Injection Attacks Proceedings of the 5th international workshop on Software engineering and middleware.

[6] Wang, X., Wang, L., Wei, G., Zhang, D., Yang, Y. 2010. Hidden Web Crawling for Sql Injection Detection Broadband Network and Multimedia Technology (IC-BNMT), 3rd IEEE International Conference

[7] Halfond, W., Viegas, J., Orso A. 2006. A Classification of SQL Injection Attacks and Countermeasures In Proceedings of the International Symposium on Secure Software Engineering.

[8] Shar, L., Tan, H. 2013. Defeating SQL Injection Computer (Volume:46 , Issue: 3) 69-77.

[9] Halfond, W., Orso, A. 2005. AMNESIA: Analysis and Monitoring for NEutralizing SQLInjection Attacks Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering.

[10] Website http://en. wikipedia.orglwiki/surface-web

[11] Gupta, S.,Bhatia, K. 2014. A Comparative study of Hidden Web Crawler International Journal of Computer Trends and Technology Volume 12 number 3.

[12] Testing website http://social.selfiecreation.com.

[13] Shehu, B., Xhuvani, A. 2014. A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques IJCSI International Journal of Computer Science Issues, Vol. 11, Issue 4, No 1.

[14] OWASP Zed Attack Proxy website https://www.owasp.org/index.php/OWASP_Zed_Attack_project.

[15] Vega website https://subgraph.com/vega/.

[16] OWASP website https://www.owasp.org/index.php/Top_10_2013_10.

[17] Ogheneovo, E.E., Asagba P. O. 2013. A Parse Tree Model for Analyzing And Detecting SQL Injection Vulnerabilities West African Journal of Industrial & Academic Research Vol.6 No.1.

[18] Boyd, W. B., Keromytis D. A. 2004. SQLrand: Preventing SQL Injection Attacks In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, pages 292–302.

[19] Thomas, S., Williams, L. 2007. Using Automated Fix Generation to Secure SQL Statements SESS '07 Proceedings of the Third International Workshop on Software Engineering for Secure Systems