Early Malicious Query Detected in web application

Teena Sahu
Mtech scholar
Department of Computer Science & Engg.
Bharti College of Engineering & Technology, durg

L. P. Bhaiya
Department of Electronic & Telecommunication Engg.
Associate professor
Bharti College of Engineering & Technology, durg

Suman Kumar Swarnkar
Department of Computer Science & Engg.
Assistant professor
Bharti College of Engineering & Technology, durg

ABSTRACT
The development of the web is expanding step by step, for the most part content is database driven. There are numerous web applications like E-Commerce, saving money where he/she needs to trust on this application and need to give individual data into their fundamental database. On the off chance that there is no privacy and security of data, at that point any one can take or see our data or may use this data for getting into mischief action. One of them is SQL infusion, a programmer may embed his terrible/vindictive SQL code into other's database and running of those questions is fit to separate private and significant data or may devastate the database. In this paper, proposing a method to recognize SQL infusion utilizing the shrouded web slithering strategy consolidating with parse tree and advanced mark. The proposed plot finds a SQL infusion helplessness by reproducing web assault and break down the information of the reaction. The proposed procedure is contrasted with concealed web creeping strategy with dissect its viability. For trial assessment, execute this framework in PHP & PYTHON with MYSQL database to dissect the outcomes.

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

1. INTRODUCTION
At display web is an imperative wellspring of data and correspondence channel amongst client and specialist organizations. As utilizing of web application is expanding, there is increment of web assault too. One of them is SQL infusion assaults (SQLIA), this powerlessness may prompt unapproved access of assets, heightening of benefits and loss of privacy and honesty. As of late the episode of SQLIA has been high to the point that an overview done by new IBM-X Force Threat Intelligence [3] for year 2014 very nearly 10 % expansion in security assaults on business which releases one billion records of individual identifiable data (PII) were spilled. Every one of these assaults are because of SQLIAs and other digital assaults.

SQL is one of the web assaults utilized by programmers to swipe information from associations. It is an application layer assault. In this instrument, malignant SQL order is executed by the web application, uncovering the backend database. A SQL infusion assault can happen when a web application uses client provided information without appropriate approval or encoding as a component of a SQL question. Infused SQL ban can change SQL articulation and envelops the security of web application.

As appeared in Fig. 1 assailants infuse pernicious SQL code and recover individual data. In this a straightforward site page where the client needs to give his client id and secret key to login in "form.php" which is gone through a firewall, web server, application server lastly to database server.

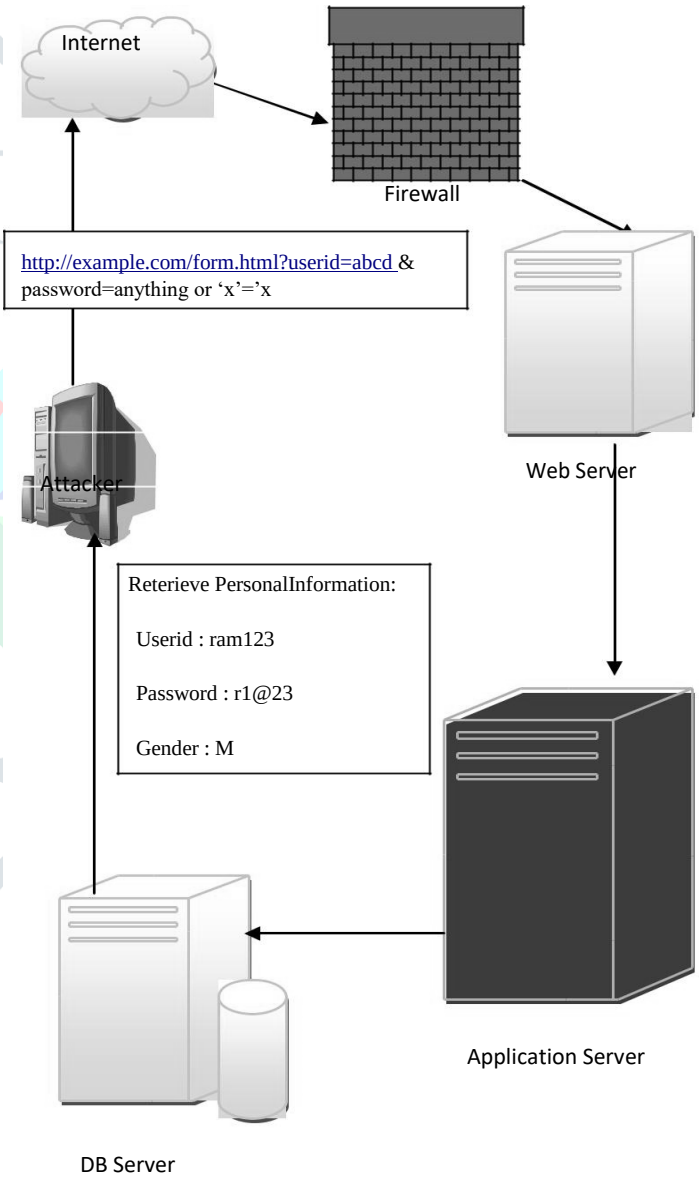


Fig. 1: SQL injection

SQLIAs isn't really anticipated by firewall and interruption identification framework in light of the fact that the sites should be open, security component permits open web movement to speak with web application (for the most part keep running more than 80/443).

In this paper, consolidating shrouded web creeping [4, 6, 11] with parse tree [5, 17] and computerized mark to recognition of SQL infusion as well as aversion at run time so the targets are counted as takes after:

- In shrouded web creeping method joining the recognition of SQL infusion at run time, examination of concealed web slithering strategy is sustained into running time discovery framework.
- To enhance the validation, utilize computerized marks, which enhance adaptability of the framework.
- Use the parse tree to identify suspected defenselessness with another proposed approach.
- Implement this framework and contrast with concealed web creeping with break down the outcomes.

To assess this approach, we test over PHP web application [12] to distinguish SQL infusion as evident positive, false positive and false negative outcomes.

The association of the paper as follows in area 2 portrayal of kinds of SQL infusion assaults, in segment 3 Hidden web slithering procedure and SQL infusion, in a segment 4 parse tree system and SQL infusion, in segment 5 proposed technique, in segment 6 conclusion the future work.

2. TYPES OF SQL INJECTION ATTACKS

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

Tautologies based SQL attack:

Repetition implies in each conceivable elucidation dependably ascertains to genuine, this assault is infused by utilizing restrictive OR administrator by which SQL question computes to genuine. This assault skirted the client validation and concentrates the information by embeddings contingent OR administrator in the WHER statement of a SQL inquiry. It will reshape the SQL question into redundancy by which database will be presented to an unapproved client. On the off chance that an assailant embeds in a question 'abcd' as secret word and anything' OR 'x'='x as watchword the inquiry moves toward becoming:

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 recommends that programmer infuses extra question with unique one by which database gets different SQL inquiries. In this technique unique inquiry is legitimate, yet another question is assaulting question with initial one. This sort of question is permitted in one inquiry because of miss setup of a framework. Assume an aggressor infuses abcd as userid and'; drop table pqr as a watchword then the subsequent inquiry seems to be:

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:
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 analyzing 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 this 5-attribute information where Vi has default value then AADT match these values against at if matches successfully then it replaces 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:

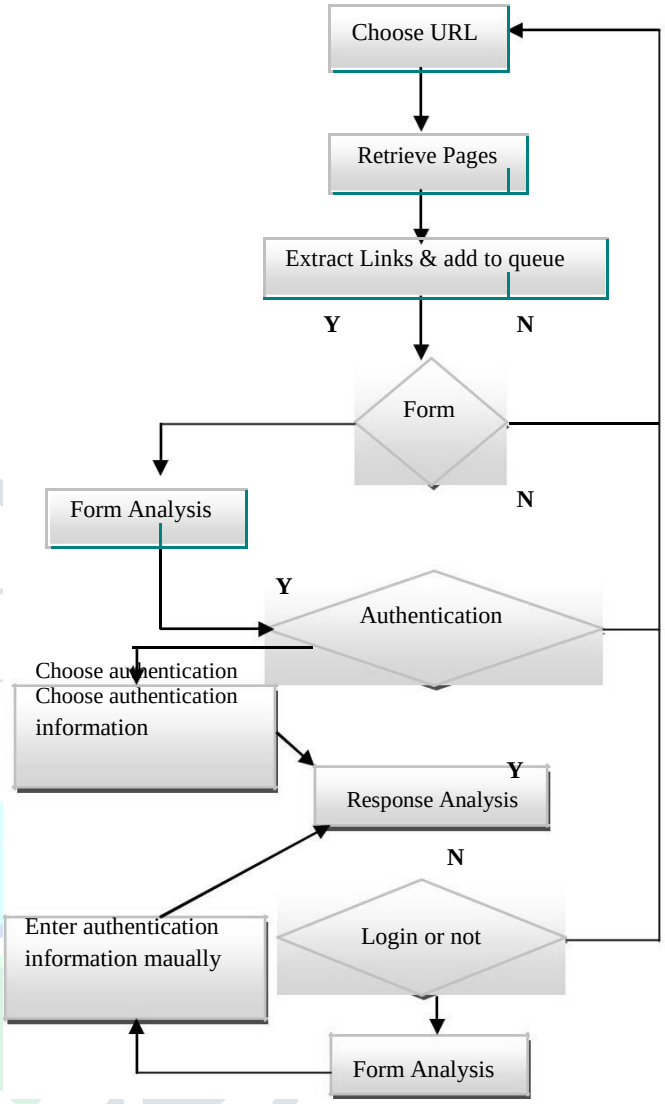


Fig 2: Hidden Web Crawling Strategy

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:

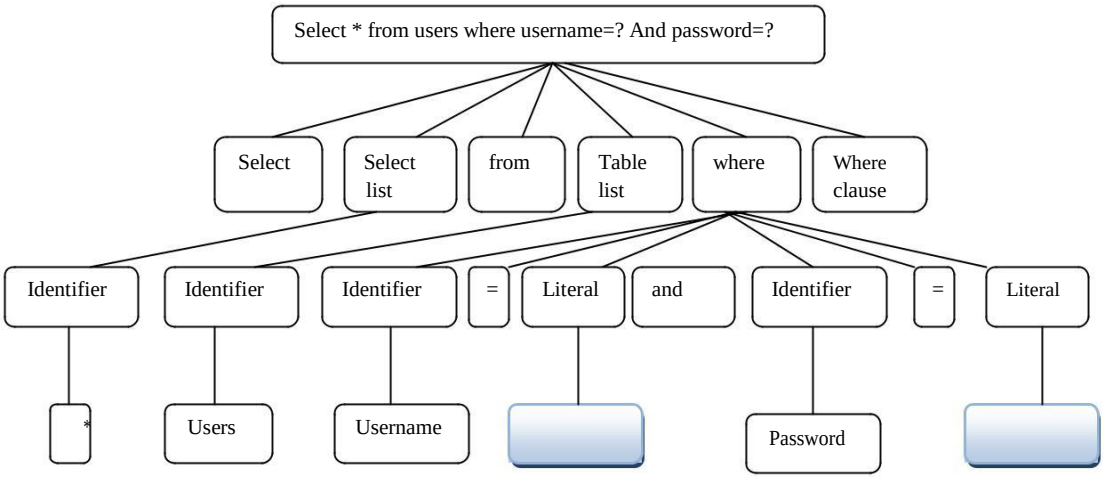


Fig 3a: Parse Tree of actual query

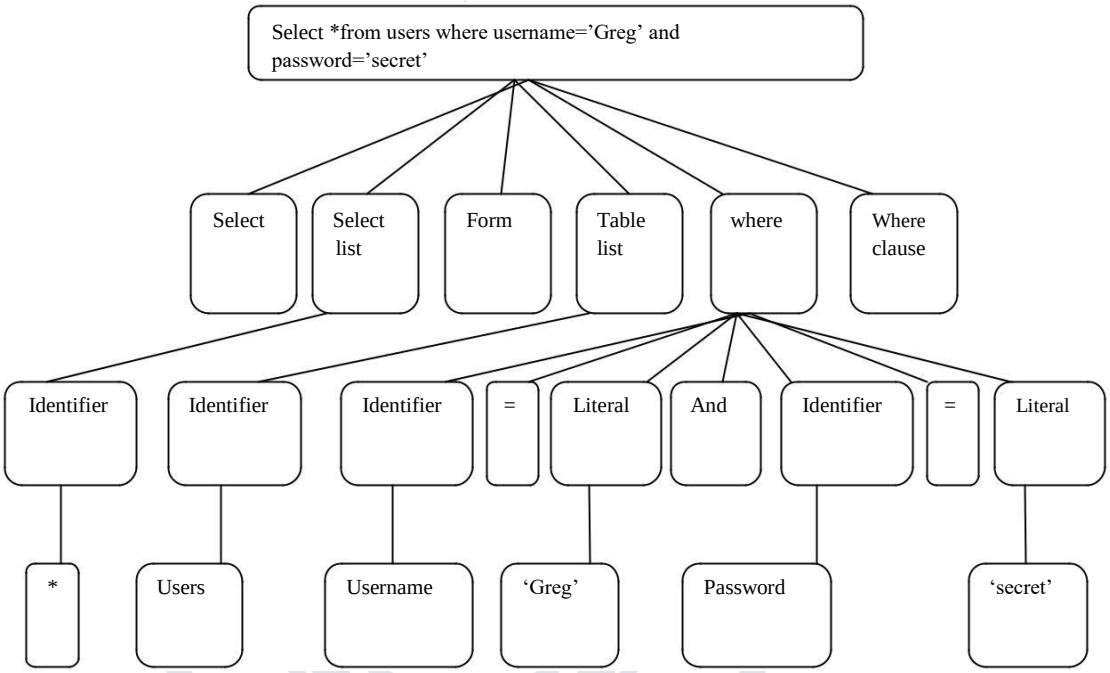


Fig 3b: Parse Tree of attacking query

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 distinguish SQL infusion at run time joining concealed web creeping strategy with parse tree and computerized signature. Execute this framework in Eclipse on Window 7, 3 GB RAM design with 2.40 GHz processor. The design of the proposed technique is appeared in figure 4:

In Evaluation stage, right off the bat experience a shrouded web crawler to discover all connections and defenseless spots with advanced signature, here utilizing the computerized signature for approval of client rather than AADT table to enhance adaptability of framework yet the outcomes at this stage have false positive and false negative. Comparison of the proposed plot is finished with the concealed web crawler [4, 6, 11] outcomes and prior apparatuses ZAP [14] and Vega [15] to dissect its adequacy.

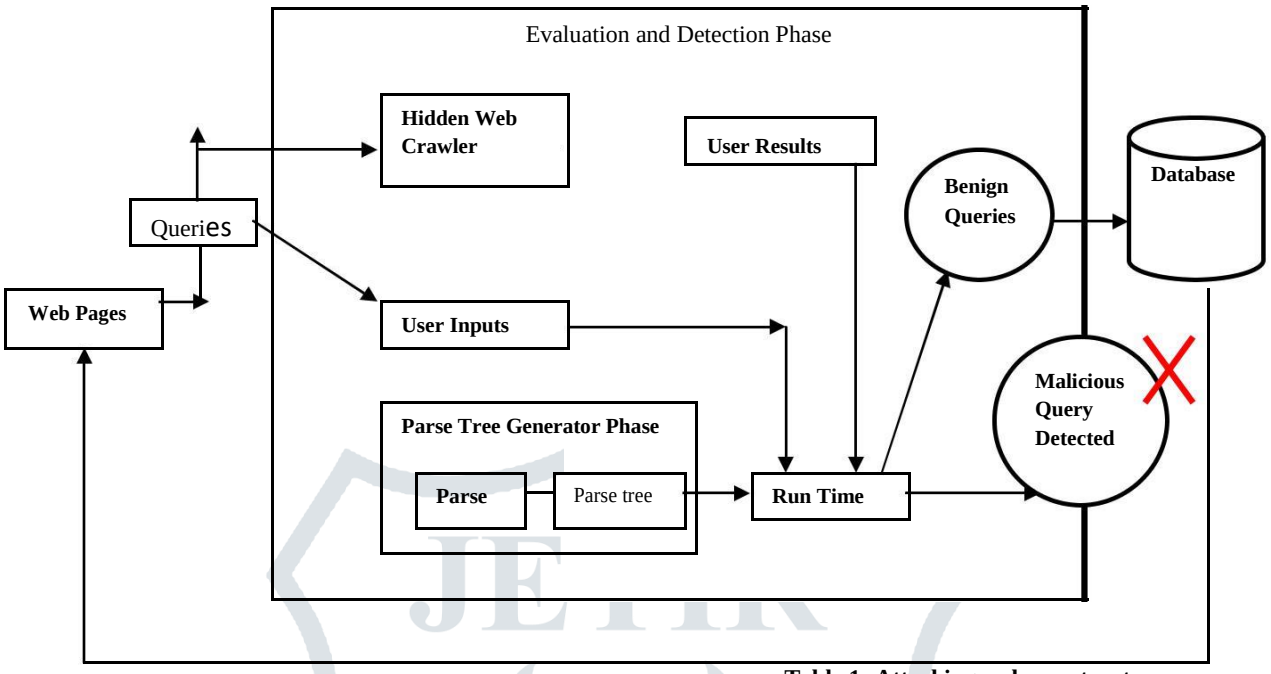


Fig 4: Evaluation and Detection phase of proposed methodology

At Detection stage, at run time parse tree method is utilized to expel the presumed powerlessness in shrouded web crawler. For this doing parsing of SQL explanation before incorporation of client contribution after consideration of contribution for PHP web application. At that point brushing the two outcomes to evacuate the presumed powerlessness by a shrouded web crawler strategy along these lines:

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

Trial of 10 assaulting code against php web application [12] and for adequacy of actualized framework comes about contrasting. The result examination on reaction is finished by along these lines:

- The result is genuine positive If the breaking down outcome demonstrates that the SQL summon executed negated by the benefits of "assaulting code" built by the finder.
- The result is false positive while assaulting code prompts database special case blunder.
- Doubtful cases appear when there is no real way to discover genuine positive and false positive outcomes.
- False negative outcomes are those which are not distinguished by the framework.

Table 2: Results of Our System as per whitelist

Dataset	Query	True Positive	False Positive	True Negative	False Negative
BenignOnly.txt	24	0	0	0	24
ClassDataSet.txt	51	27	0	0	24
MaliciousOnly.txt	27	27	0	0	0
WhitelistDataSet.txt	27	23	0	4	0

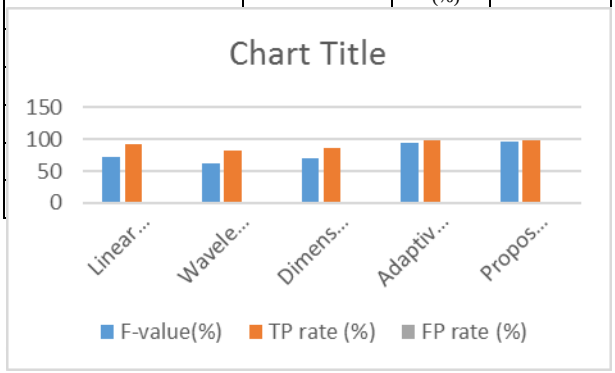
Table 3: Results of Our System as per blacklist

Dataset	Query	True Positive	False Positive	True Negative	False Negative
BenignOnly.txt	24	0	0	0	24
ClassDataSet.txt	51	27	0	24	0
MaliciousOnly.txt	27	27	0	0	0
WhitelistDataSet.txt	27	23	0	4	0

The arrangement is tried on php web application [12] with the rundown appeared in table1 assaulting code in which actualized framework can identify all these weaknesses where as in shrouded web crawler there is 1 false negative and 3 false positive separately so executed framework is superior to anything concealed web slithering. Actualize this

Table 4: Results of F-Value, TP Rate & FP Rate

Method	F-value (%)	TP rate (%)	FP rate (%)



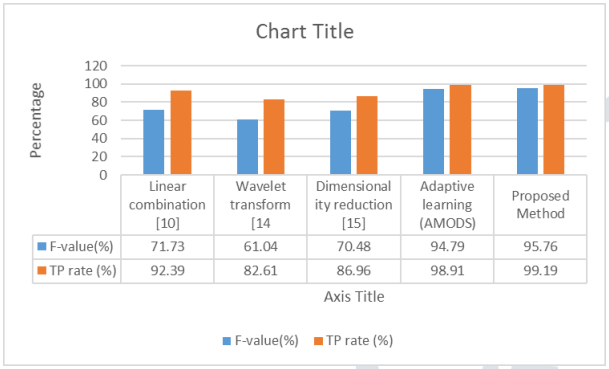


Fig. 6 Chart of F-Value & TP Rate

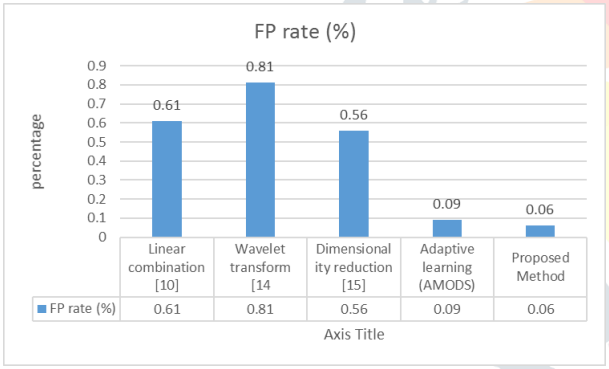


Fig. 7 Chart of FP Rate

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 95.76% .

6. CONCLUSION AND FUTURE WORK

Above all web application in light of middleware innovation, to recover data from social database SQL. From the above outcomes and diagram discourse it can be say that actualized framework is more secure whereas shrouded web crawler can distinguish half powerlessness.In proposed conspire time overhead increments. Actualized framework gives another way to deal with secure a web application. In not so distant future we may improve the calculation utilized as a part of shrouded web crawler and parse tree to identify SQL infusion.

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