

Detection and Prevention of SQL Injection Attack by Dynamic Analyzer and Testing Model

Rana Muhammad Nadeem¹
Computer Science Department
Govt. Post Graduate College
Burewala, Pakistan

Rana Muhammad Saleem²
Computer Science Department
UAF Sub Campus Burewala
Burewala, Pakistan

Rabnawaz Bashir³
Computer Science Department
Comsats Institute of Information Technology
Vehari, Pakistan

Sidra Habib⁴
Computer Science Department
UAF Sub Campus Burewala
Burewala, Pakistan

Abstract—With the emergence and popularity of web application, threats related to web applications has increased to large extent. Among many other web applications threats Structured Query Language Injection Attack (SQLIA) is the dominant in its use due to its ability to access the data. Many solutions are proposed in this regard that has success in specific conditions. The proposed model is based on the dynamic analyzer model. The proposed model also has certain advantages like wide applicability, fast response time, coverage to large number of techniques of SQL Injections (SQLI) and efficient in term of resource usage.

Keywords—Structured Query Language (SQL); injection attack; request receiver; analyzer and tester

I. INTRODUCTION

It is the information age and information is critical for business process. Web applications are major source of information for business process critical for the survival for any organizations [1]. With the popularity of web applications there is also increase in web application vulnerabilities. Across many types of web vulnerabilities SQL Injection (SQLI) has become the predominant method due to its rewarding nature to have access to the data and due to advances in its techniques. It is observed that SQLI is the most widely used techniques for the web applications [2]. According to Open Web Application Security Project (OWASP) (Organization that ranked the web Applications risks) in SQLIA is the dominant web application security risk as shown in Fig. 1.



Fig. 1. OWASP SQLIA ranking over the years [3].

Due to huge rewarding of having access to the database the SQLIA has become the predominant web application security risks and their technique has become more sophisticated over time [4].

Due to emergence of different sophisticated techniques SQLIA has shown a tremendous increase in its spread to web applications of finance banks, educational institutes, global market and many more [5]. The following Fig. 2 shows the relative spread of SQLIA as compared to other types of Web Vulnerabilities. SQLIA is also among the top when compare the spread or choice of web vulnerability among the intruders.

Percentage of web sites vulnerability by class

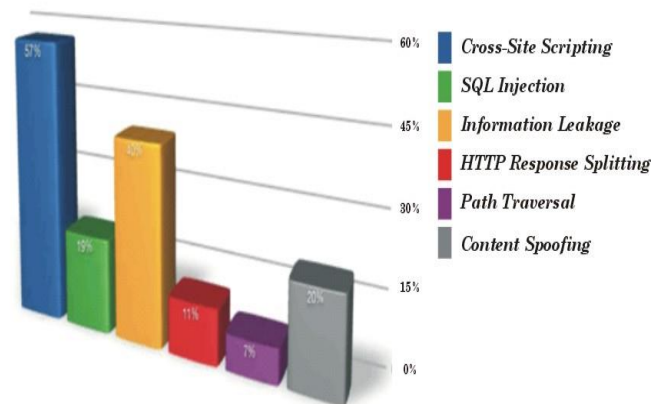


Fig. 2. Volume percentage of web application security risks [6].

For smooth operations of the organization that utilize web applications it is necessary that web applications operate at reasonable level of security. Due to complexities of web technologies and varieties of risks it is not an easy task to save the web applications from intruders and threats [7]. Even sometimes it is very difficult to detect that some serious threat has been occurred [8]. In Fig. 3 to 5 statistics shows the relative difficulty in detection of web application threats.

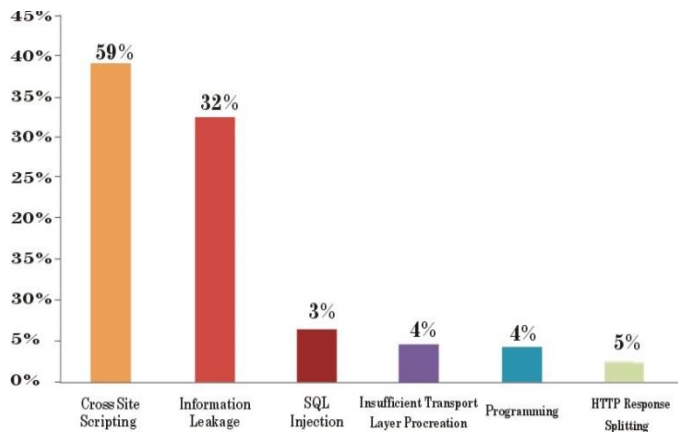


Fig. 3. Percentage of web security risks in web applications [6].

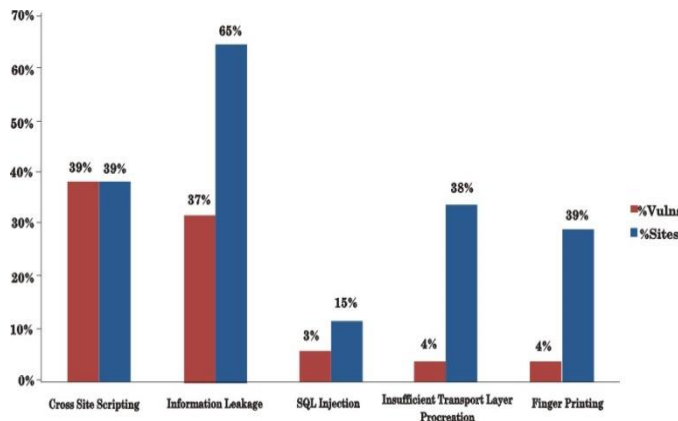


Fig. 4. Probability to detect Vulnerabilities in web application [6].

According to the Web Application Security Consortium (WASC) 78% of web applications are susceptible of security risks and 49% of web applications are susceptible to risks of high level [6].

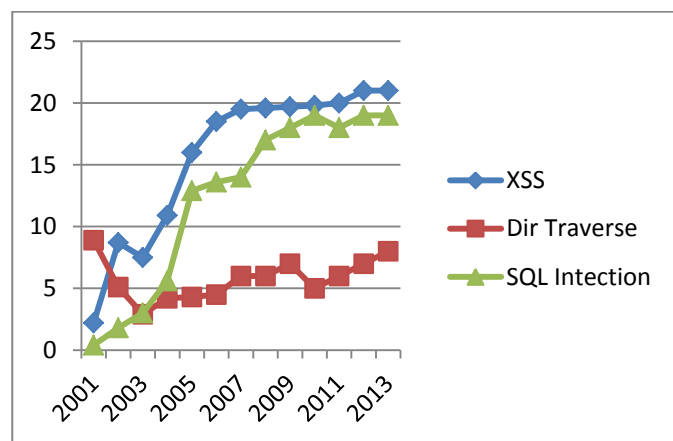


Fig. 5. Growth of web Application vulnerabilities from 2001 to 2010 [9].

SQL Injection Attacks (SQLIA) are among the top of the Input validation attacks and in top five among all the web application security risks [10].

It is important to observe that SQLIA injection Attacks are 30% of total web application security risks due to potential

advantage associated with the SQLIA for the intruders to gain access to the data and much useful information [11]. Due to emergence of needs of more secure web applications it is strongly required that research should focus on the SQLIA and come with a solution that can overcome the problems associated with previous proposed solutions like performance issues, code change and inefficient use of the resources [9].

A. SQL Injection Attack (SQLIA) Process

Data driven web sites are vulnerable to SQL Injection attack where database is a black box in three tier architectures. In this architecture SQL statements are generated in response to HTTP requests [12]. These HTTP request may contain parameters that are used by attackers to produce a query of their interest to have illegal access to the database as shown in Fig. 6.

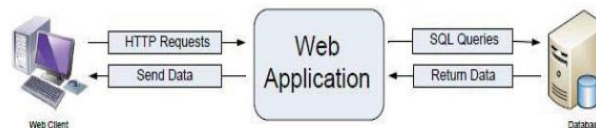


Fig. 6. SQL Injection attack process.

Log In page as shown in Fig. 7 is the most vulnerable for the SQLIA attack and following is the PHP code snippet that produce dynamic query in response to user input [9] as shown in Fig. 8 and 9.

Username:

Password:

Fig. 7. Log In form.

```

// connect to a database
mysql_connect(servername,username,password);
// store user input in the variables collected from the user input login form
$username=$_POST['username'];
$password=$_POST['password'];
// dynamically build the query from the user input
$query="SELECT* FROM tbl_users WHERE username = '$username' AND password = '$password'";
// execute a query
$result = mysql_query($query);
if($result)
    return true;
else
    return false;
  
```

Fig. 8. PHP Code snippet to generate dynamic query in response to user input.

SELECT * FROM tbl users WHERE username = ' user_Name AND PASSWORD='pwd' . . ;

Fig. 9. SQL query as a result of code.

In next Fig. 10 at the same form user try to attempt a simple SQLIA to bypass the authentication.

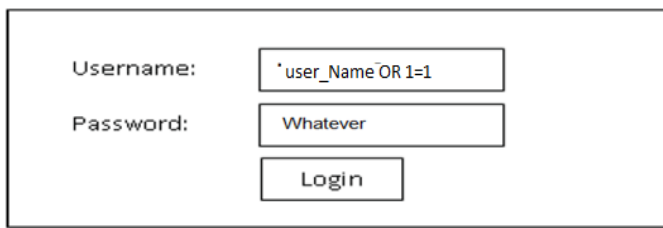


Fig. 10. Simple attempt for SQLIA.

```
SELECT * FROM tbl users WHERE username = 'user_Name OR 1=1 AND password= 'Whatever'
```

Fig. 11. Dynamic generated query in response to above input.

In Fig. 11 attacker try to ignore the password by using the – comment operator as everything would be ignored after the comments operator even the password. In this scenario user name is tried to be true using the OR operator. This, the simple scenario and with different techniques intruders want to add query of their interest to have access to the information of their interest.

B. Techniques of SQL Injection Attack (SQLIA)

1) Tautology Based Attacks

In tautology attack, malicious contents are added using the conditional statement that always evaluate to true. Previous scenario is the perfect example of this attack [13].

Select * from tbl users Where username='rabnawaz' or '1'='1' and password='whatever'

2) Union Attack

In this technique, malicious query is added with the safe query using the UNION keyword [14].

['UNION SELECT pwd FROM user-info WHERE id='abc' and pwd='']

3) Logically Incorrect query Attack

In this type of technique logically incorrect type of query is performed to have information about some structures of the data base to proceeds further [15].

4) Piggybacked Query

Certain delimiters like “;”, “;”, “;” used to join the legitimate query with the illegitimate one [6].

Select * from users where id='rabnawaz' and pwd=''; Drop table users...'

5) Alternate Encoding

By changing the coding schema, the illegal query can be bypassed through the filter that tests the legitimacy of the query [16].

6) Inference Attack

Blind and timing techniques are used in inference attacks. In blind attack, a series of the simple queries are performed to have guess about the structure of the data base. In timing attack the query processing time is observed to infer some information presence in the data base.

C. Consequences of SQL Injection Attacks

It has been observed that due to access to the data base SQLIA has become the dominant web application security risks over the last ten years. Database is the very critical for successful operations of any organization. Sensitive information in the database can be used in many ways to serve the attacker purpose [17]. Followings could be the intentions of the attackers to use SQLIA.

To gain information about data base finger prints like type of data base, SQL language used, etc. This information helps the attacker to proceeds or use more sophisticated attacks [18].

- 1) To gain information about user credentials [19].
- 2) To get the database schema [20].
- 3) To extract and modify the data base [1].
- 4) To perform Denial of Services like shutting down the data base, dropping tables, etc. [21].
- 5) Replacements of files with false or tempered information [19].
- 6) Execution of remote commands.
- 7) Shop lifting, account balance change.
- 8) Interacting with underlying operating system.

II. INTRODUCTION TO EXISTING TOOLS

Following's are the major tools available for detection and prevention of SQLIA vulnerabilities:

A. Acunetix

Acunetix web application vulnerability detection scanners that use the XSS black box and Advance SQL injection techniques. It crawls and scans sites and with help of black box and grey box hacking techniques for identification of serious vulnerabilities. Acute nix claimed to detect more than three thousand web application vulnerability including SQL injection Attacks, XSS and host header injections [22].

B. SQLmap

SQLmap is open source analysis tool that automatically detect SQL injection vulnerabilities. It is a powerful tool that has powerful detection engine many niche database penetration features [23].

C. SQLiX

SQLiX is a scanner that crawls and detects SQL Injections. This tool can detect normal and blind SQL injections and there is no need to change the original SQL request [24].

D. Wapiti

Wapiti is web vulnerability scanner for the web application that helps to audit or assess the security of a web application. It uses black box scan that do not scans the code instead use the script and forms where actually injection took place. Wapiti can detect the various techniques of SQLIA [25].

E. Paros

Paros is also a scanner for detection of web vulnerabilities that is java based HTTP/HTTPS proxy. It allows to analysis of the HTTP request with support of spiders, proxy-chaining, XSS, SQL Injection Client certification, etc. [26].

F. Pixy

Pixy is open source tool to detect web application security risks [27].

III. PROPOSED SOLUTION

In this article, a solution is proposed that is based on dynamic analyzer.

A. Proposed Solution Architecture

Proposed model based on the dynamic analyzer that work as user would request the page and that request is received and analyzed to check that request is for pages without vulnerabilities (P') and with vulnerabilities (P), with help of knowledge base. If the user request is for P pages then request is served and if the request is for the P' pages then tester would handle the situation by testing the user request. Tester would generate the possible expected response from the user and user request would be served. On response from the user the response is compared with the expected result and any discrepancy is observed. If the user response is normal then the request is served otherwise user request is rejected and knowledge base is updated for page vulnerabilities and possible rule addition. The complete flow of proposed solution is shown in Fig. 12.

$$P = \sum_{i=1}^n P_i = P_1 + P_2 + P_3 \dots \dots P_n$$

Equation 1: Set of Pages without possible vulnerabilities and P' is the pages where no serve side scripting or not vulnerable.

$$P' = \sum_{i=1}^n P'_i = P'_1 + P'_2 + P'_3 \dots \dots P'_n$$

Equation 2: Pages with possible vulnerabilities

$$R = \sum_{i=1}^n r'_i = r_1 + r_2 + r_3 \dots \dots r_n$$

Equation 3: Set of knowledge base rule

B. Algorithm of Proposed Solution

```

Function Analyzer (Requested_Page)
{
  Mark_Page=Mark(Requested_Page)
  If (Mark_Page is Vulnerable)
  {
    Tester(Requested_Page);
  }
  else
  {
    Serve_Request(Requested_Page)
  }
  Tester(Requested_Page)
  {
    Generate_Expected_Response(Page_Request);
    Serve_Request(Requested_Page);
    Response= Test_Reponse();
    If (Response is Expected)
    {
    }
    else
    {
      Block_Request ();
      Update_Knowledge_Base ();
    }
  }
}
    
```

C. Flow Chart of Proposed Solution

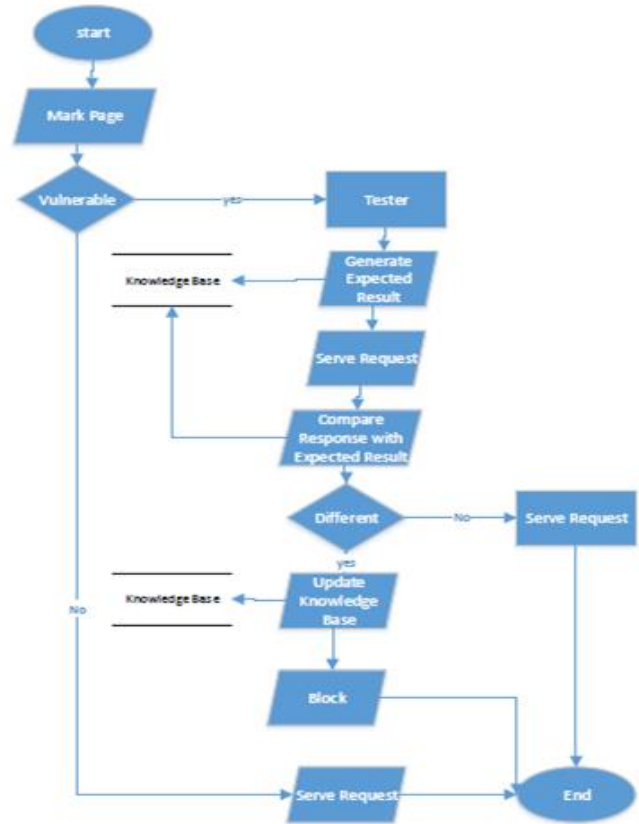


Fig. 12. Flow chart of the proposed solution.

D. Implementation and Evaluation of the Proposed Solution

To evaluate the proposed solution its performance is compared with existing tools described in previous sections. These tools and proposed solutions are applied to detect the SQLIA and block the SQLIA in different types of web application specified in Table 1. These tools are evaluated against different criteria mentioned below.

1) Implementation

Using ASP.Net different classes of web Application are used to evaluate the different tools against the different SQL Injection Attack.

2) Test Scenarios

Following criteria are used to judge the performance of different tools.

- a) No of SQLIA attacks detected
- b) No of SQLIA attacks blocked
- c) Time taken to prevent SQL Injection Attack
- d) Time taken to block SQL Injection Attack
- e) No of types of SQLIA detected
- f) No of types of SQLIA blocked
- g) No of Database supported.

Following dataset is used to evaluate the above-mentioned conditions.

TABLE. I. DATA SET FOR EVALUATION OF DIFFERENT TOOLS AND TECHNIQUES

Applications	No. of Inputs
Portals	100
Classifieds	100
Online Shopping	100
University Database	100
Financial Database	100

E. Evaluation Results

The different evaluation results have been achieved by using above mentioned test scenario as shown in Fig. 13 to 19.

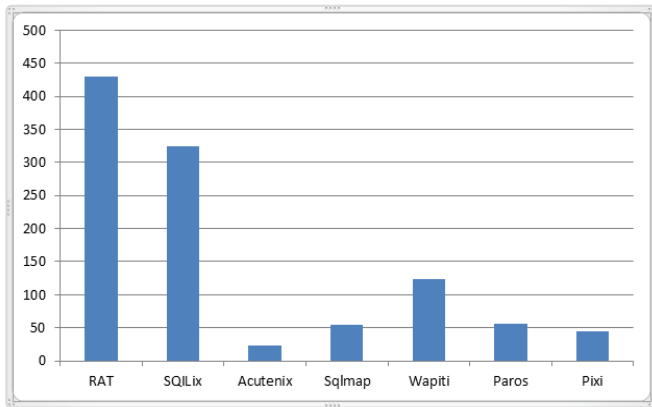


Fig. 13. Number of SQL injection attacks detected.

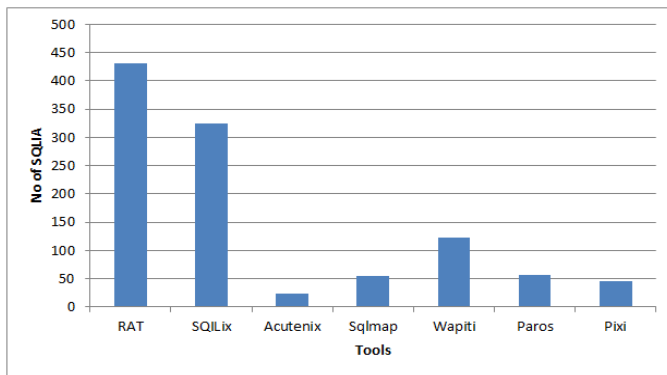


Fig. 14. Number of SQL injection attacks blocked.

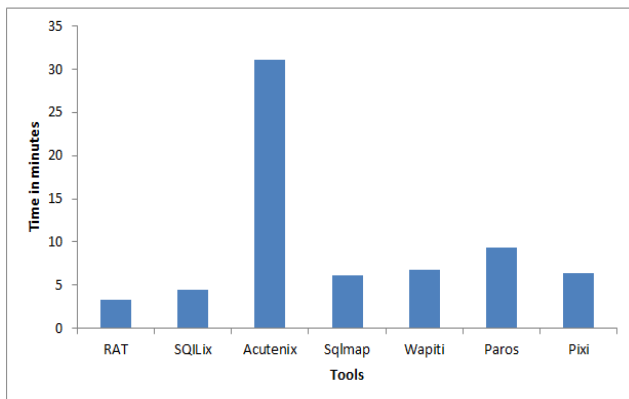


Fig. 15. Average time taken to detect SQLIA.

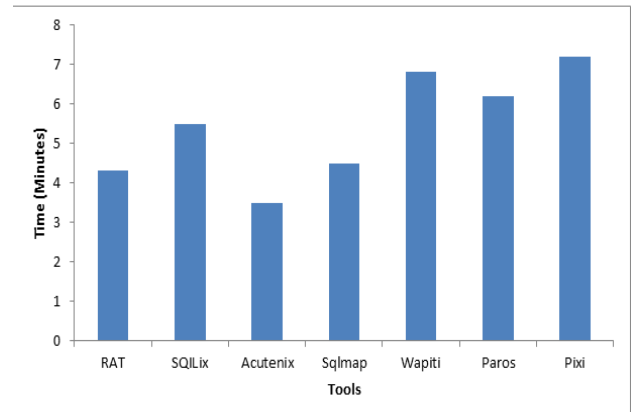


Fig. 16. Average time taken to block SQLIA.

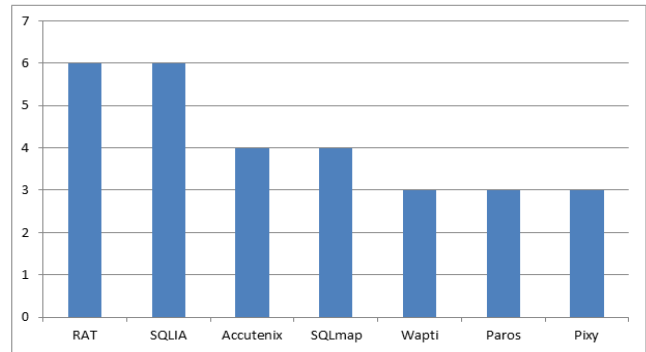


Fig. 17. Number of types of SQL injections techniques detected.

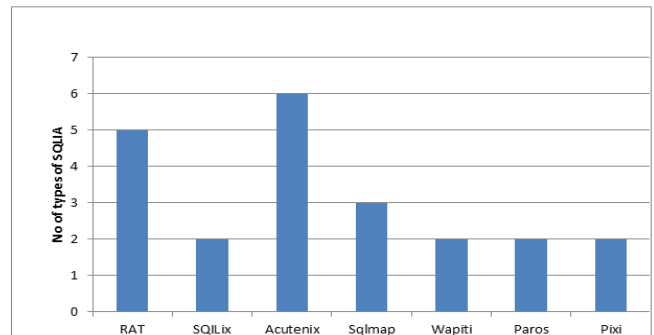


Fig. 18. Number of types of SQL injection techniques blocked.

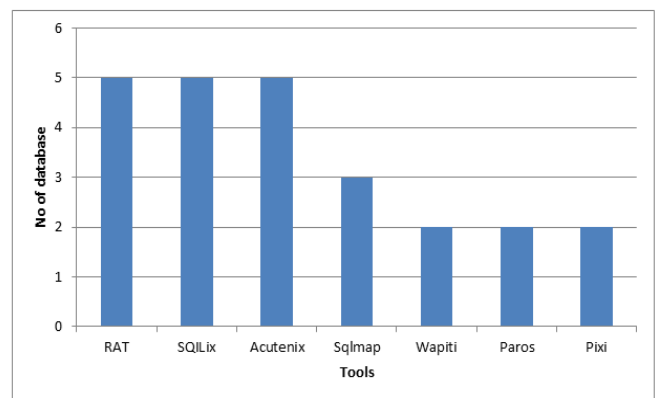


Fig. 19. Number of database supported.

IV. CONCLUSION

SQL Injection Attack has emerged as major threats to web applications. Many solutions were proposed to detect the SQLIA vulnerabilities in web application. Proposed solution based on dynamic Analyzer and tester performed well to detect and block the SQLIA and response time is also excellent as compared to another tool. The proposed solution also needs not to change the source code of the web application and use minimum resources of the system. One major advantage of the proposed solution is that it can handle the advanced SQLIA techniques as knowledge base is updated to handle modern types of threats.

V. FUTURE WORK

The proposed solution use MS SQL analyzer for possible vulnerabilities detections and page marking. The tools need to improve in such a way that any sort of analyzer can be configured for analysis. Knowledge base maintains the techniques and knowledge about different attacks. Knowledge base should be updated using different machine learning approaches.

REFERENCES

- [1] A. Anchlia and S. Jain, "A novel injection aware approach for the testing of database applications," in Proceedings of the 2010 International Conference on recent trends in information, telecommunication and computing ITC, Wasington DC, 2010.
- [2] A. Ciampa, C. A. Visaggio and M. D. Penta, "A heuristic-based approach for detecting sql-injection vulnerabilities in web applications," in In Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, SESS '10, New York, NY, USA, 2010.
- [3] "https://www.owasp.org," 01 June 2017. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Main. [Accessed 12 June 2017].
- [4] A. Kieyzun, P. J. Guo and K. Jayaraman, "Ernst. Automatic creation of sql injection and cross-site scripting attacks," in 31st International Conference on Software Engineering, ICSE '09,, Washington, 2009.
- [5] A. Liu, Y. Yuan, D. Wijesekera and A. Stavrou, "Sqlprob: a proxy-based architecture towards preventing sql injection attacks," in 2009 ACM symposium on Applied Computing, SAC '09, New York, 2009.
- [6] S. Gordeychik, 15 December 2013. [Online]. [Accessed December 2013].
- [7] A. Razaq, A. Hur, N. Haider and F. Ahmad, "Multi-layered defense against web application attacks," in Sixth International Conference on Information Technology: New Generations, Washington, DC, 2009.
- [8] A. Tajpour, M. Massrum and M. Heydari, "Comparison of sql injection detection and prevention techniques," in Education Technology and Computer (ICETC), 2010 2nd International Conference, 2010.
- [9] D. A. Anup Shakya, "A Taxonomy of SQL Injection Defense Techniques," Karlskrona Sweden, 2011.
- [10] A. Tajpour and .. Shooshtari, "Evaluation of sql injection detection and prevention techniques," in Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference, 2010.
- [11] A. Ciampa, C. A. Visaggio and M. D. Penta, "A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications," in Proceeding SESS '10 Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, New York, 2010.
- [12] A. Tajpour, S. Ibrahim and M. Masrom, "SQL injection Prevnetion and detection Techniques," International Journal of Advancements in Computing Technology, vol. 3, no. 7, pp. 85-91, August 2011.
- [13] B. Indrani and E. Ramaraj., "X-log authentication technique to prevent sql injection attacks," International Journal of Information Technology and Knowledge Management ., vol. 4, pp. 4:323-328,, 2011.
- [14] C. T. M and B. J., "Design considerations for a honeypot for sql injection attacks," in LCN'09, 2009.
- [15] D. Das, U. Sharma and D. Bhattacharyya, "An approach to detection of sql injection attack based on dynamic query matching," International Journal of Computer Applications, vol. 1, no. 25, p. 28-34, February 2010.
- [16] K. Amirathimasebi, S. Jalalinia and S. Khadem, "A Survey of sql injection defence mechanisms," in International Conference Internet Technology and Secured Transactions ICITST 2009, 2009.
- [17] A. Moosa, "Artificial Neural Network based Web Application Firewall for SQL Injection," World Academy of Science, Engineering and Technology, vol. 40, pp. 42-51, April 2010.
- [18] Z. Lijiu, Q. Gu, S. Peng and X. Chen, "D-WAV A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms," in Fifth International Conference on Software Engineering Advances (ICSEA), 2010, Nice, 2010.
- [19] Z. Jan, M. Shah, A. Rauf, M. Khan and S. Mahfooz, "Access control mechanism for web databases by using parameterized cursor," in Future Information Technology (FutureTech), 2010 5th International Conference, 2010.
- [20] Xiang Fu and K. Qian, "SAFELI – SQL Injection Scanner Using Symbolic Execution," in Workshop on Testing, Analysis and Verification of Web Software, July 21, 2008.
- [21] M. Cova, D. Balzarotti, V. Felmetzger and G. Vigna, "Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications," 12 December 2013. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.6909>.
- [22] I. Musacat, "https://www.acunetix.com/blog/docs/blind-sql-injector/," 1 Feburary 2017. [Online]. Available: <https://www.acunetix.com/blog/docs/blind-sql-injector/>. [Accessed 15 June 2017].
- [23] B. Damele A. G. and . S. Miroslav, "http://sqlmap.org/," 12 June 2016. [Online]. Available: <http://sqlmap.org/>. [Accessed 13 June 2017].
- [24] AnirudhAnand, "https://www.owasp.org/index.php/Category:OWASP_SQLiX_Project," 16 March 2014. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_SQLiX_Project. [Accessed 10 June 2017].
- [25] "http://wapiti.sourceforge.net/," 20 October 2014. [Online]. Available: <http://wapiti.sourceforge.net/>. [Accessed 10 June 2017].
- [26] "http://sectools.org/," 15 December 2015. [Online]. Available: <http://sectools.org/tool/paros/>. [Accessed 05 june 2017].
- [27] J. N., C. Kruegel and E. K. , "Pixy: a static analysis tool for detecting Web application vulnerabilities," Security and Privacy, 2006 IEEE Symposium on, pp. 41-46, 2006.