# A Machine Learning Approach for Prevention of SQL Injection

**Vennila V[1], Savitha S[2], Baby Anitha E[3], Panneerselvam D[4], Sandhya E[5]**

[1,2]Assistant Professor, Department of Computer Science & Engineering, K.S.R College of Engineering, Tamilnadu, India,
Email: vennilview@gmail.com[1], infosavi@gmail.com[2]

[3]Associate Professor, Department of Computer Science & Engineering, K.S.R College of Engineering, Tamilnadu, India,
Email: ebabyanitha@gmail.com

[4,5]Student, Department of Computer Science & Engineering, K.S.R College of Engineering, Tamilnadu, India,
Email: panneerselvamdksrce@gmail.com[4], sandhyaselvaraj20@gmail.com[5]

**Abstract –** In today's scenario, web application firewalls are an essential protection mechanism for online software system. In Internet age, the most critical security risk of vulnerable web applications is SQL Injection attacks. With the increasing threats of SQL Injections, Web Application Firewall (WAF) must be updated and tested regularly to prevent attackers from easily attacking them. As technology grows, the number of attackers who intend to attack the applications find numerous new ways to enter the system. Thus, the existing systems find it difficult to cope up with the new hackers with new technologies to completely save the system. In the existing WAF, the white box testing and static analysis approach needs access to source code. The model-based testing requires more sets of rules. The black box testing is not efficient for detecting SQL injection attacks. Machine learning is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. The concept of collaborating machine learning with web application firewalls increases the efficiency of the existing system. The approach used in this paper is Unsupervised Learning Technique. The algorithm used for Unsupervised learning technique is the k-means algorithm which is used for clustering problems. The flow of the system can be given as follows. The end user makes a request in the Web application, the values of the request are extracted and sent to the SQL injection detector, which provides two layers of security. In the first layer of security, patterns are generated using CFGs for low level attacks. The second layer of security for high level attacks is trained using Unsupervised Learning Algorithm.

**Keywords -** SQL Injections, SQL Injection Detector, Two-layer Security, Unsupervised Learning Technique

## I. INTRODUCTION

A web application firewall (WAF) is used in HTTP applications as application firewalls. It applies a collection of rules to HTTP conversation. Generally, these rules cowl common attacks like cross-site scripting (XSS) and SQL injection. While proxies typically shield clients, WAFs shield servers. A WAF is deployed to shield a selected internet application or set of internet applications. A WAF may be thought as a reverse proxy. WAFs might be available in the form of an appliance, server plugin or filter and can also be custom-built to an application. The hassle to perform this customization may be insignificant because, if there is any need to change the application, the WAFs should also be maintained according to the change.

Web applications with high security requirements are commonly protected by WAFs. In an overall system architecture, a WAF is placed in front of the web application that must be protected. Every request that is sent to the web application is examined by the WAF before it reaches the web application. The WAF hands over the request to the web application only if the request complies with the firewall rule set. Since the threat of cyber-attacks are growing day by day, the WAFs are getting complicated. Also, manually testing and maintaining the principles is an issue. Therefore, automatic testing techniques for WAFs are crucial to stop malicious requests from reaching internet applications and services. In this paper, the focus is on testing efforts on a common category of attacks, namely, SQL injections (SQLi). SQLi has received a lot of attention from practitioners [2 - 3], [5 - 6], [12], [15 - 18], [20]. The Open Web Application Security Project (OWASP) finds that the prevalence of SQLi vulnerabilities is common and the impact of a successful exploitation is severe.

SQLi is an attack technique in which attackers inject malicious SQL code fragments into input parameters that lack proper validation or sanitization. An attacker might construct input values in a way that changes the behavior of the resulting SQL statement and performs arbitrary actions on the database(e.g., exposure of sensitive data, insertion or alteration of data without authorization, loss of data, or even taking control of the database server). SQLi attacks have never lost it's trend and always possess a major threat to the web applications of various domains.

This paper considers three main categories of SQLi attacks in the grammar 1) Boolean; 2) Union and 3) Piggy-backed. These types of attacks aim at manipulating the intended logic by injecting additional SQL code fragments in the original

SQL queries. This paper briefly discusses each attack category and provide example attacks that can be derived using the grammar.

1) Boolean Attacks: The intent of a Boolean SQLi attack is to influence the where clause within an SQL statement to always evaluate either to true or false. As a result, a statement, into which a Boolean SQLi attack is injected, returns on its execution either all data records of the queried database tables (in case the where clause evaluates always to true) or none (in case the where clause evaluates always to false). This attack method is typically used to bypass authentication mechanisms, extract data without authorization, or to identify injectable parameters.

2) Union Attacks: The union keyword joins the result set of multiple select statements and hence union SQLi attacks are typically used to extract data located in other database tables than the original table in specified in the query. For example, consider an application that retrieves a list of product names based on a search term. The SQL statement to retrieve the product names might be

SELECT name FROM products WHERE name LIKE "%search term%

where search term is a string provided by the user. However, an attacker could provide the search term phone%" UNION SELECT passwd FROM users #%", which would result in the statement

SELECT name FROM products WHERE name LIKE "%phone%" UNION SELECT passwd FROM users

Hence, in addition to a list of products containing the search term phone, the attacker could obtain the passwords of all users with the modified query above.

3) Piggyback Attacks: In SQL, the semicolon (;) can be used to separate two SQL statements. Piggy-backed attacks use the semicolon to append an additional statement to the original statement and can be used for a wide range of attack purposes (e.g., data extraction or modification, and denial of service). An example of a piggy-backed attack is "; DROP TABLE users #". If this attack is injected into a vulnerable SQL statement, it drops the table user. Thus, it potentially breaks the application.

## II. RELATED WORKS

Various techniques are proposed within the literature to observe SQLi attacks based on a range of approaches, including white-box testing [15], static analysis [21], model-based testing [11], and black-box testing [1]. However, such techniques present some limitations that may adversely impact their practical pertinency as well as their vulnerability detection capability. As an example, white-box testing techniques and static analysis tools need access to source code [22], which might not be possible when dealing with third-party components or industrial appliances and are linked to specific programming languages [13]. Model-based testing techniques require models expressing the protection policies or the implementation of WAFs and the net application under test [11], that are typically not offered or terribly troublesome to manually construct. Black-box testing ways don't need models or the source code however they're less effective in sleuthing SQLi vulnerabilities. Indeed, comprehensive reviews on black-box techniques [1],[9] have unconcealed that many sorts of security vulnerabilities (including SQLi attacks) remain mostly undiscovered, thus, warrant more analysis.

A Machine-Learning Driven Evolutionary Approach for Testing Web Application Firewalls [4] proposed a Supervised Machine Learning Technique that is used for analyzing the SQL Query and to detect the SQLi attacks (to help in detecting vulnerabilities in Web Application Firewall). The major issue in this approach is, it helps in detecting only the injections specified in the defined class. If any specific type of SQLi attack is not already defined in the class, there will not be any detection of attacks.

A Simple and Efficient Framework for Detection of SQL Injection Attack [19] used Static Analysis and Runtime Monitoring for analyzing the query that is to be executed on the server side. It helped in prevention of Static and Dynamic SQL Injection attacks. One of the major issues with this approach is that it is efficient only for simple patterns and when the attack is based on complex pattern, this approach is not efficient.

A novel method of SQL injection attack detection based on removing SQL query attribute values[8] used an approach called Analysis for Monitoring and Neutralizing SQL Injection Attacks(AMNESIA). This approach does the required String analysis and also there is no need any Source code adjustments here. One of the major issues in this approach is that automatic detections is not available. Detection Rates are less than 75% which makes this approach unreliable.

A grammar based Whitebox fuzzing [15] addresses the critical memory corruption vulnerability in file processing application that starts application under a test with given well-formed inputs and uses symbolic execution to form input constraints.

XSS Analyzer provides a learning approach to web security testing [14] where the XSS Analyzer generates attacks from a grammar and learns which pattern an attack cannot contain in order to escape detection.

Automated pseudo-live testing of firewall configuration enforcement [7] where a framework automatically test if a approach is correctly imposed by a firewall. Therefore, the framework generates a set of approaches as well as test traffic and checks whether how well the firewall handles the generated traffic according to the generated approach. Systematic

structural testing of firewall policies[10] defined structural coverage condition of approaches under test and developed a test generation technique based on impulsion solving that tries to maximize structural coverage.

## III. PROPOSED SYSTEM

In the Existing system, there is no automatic detection system for detecting and preventing SQLi Attacks. It makes the detection based on certain set of rules. The system checks the query with each and every rule and then it detects for attack. If a certain type of rule is not maintained in the set and if that attack is invoked, then the system will allow the query to be executed since that rule is not present in the rule set. Also, we need to define more complex rules for huge applications. The system for preventing SQL injections using machine learning helps in thwarting the exploitation of any database in the server side. In this system, a component is developed that resides in the server side. Before executing the request made by the client, the system will check for the SQLi attack pattern that are appended with the values passed to the application server. There are two levels of security in this system, the first one being the patterns generated by the context free grammar rules and comparing the values with the pattern generated by the defined rules for SQL attacks. The second level of security includes the machine learning algorithm(k-means) that clusters the pattern based on the given data set and categorizes the new data into one of those clusters and detects the injection. Figure 1 shows the System Architecture, describing how the system is built. The web browser is used by the client to send necessary requests to the web server. The web browser sends the values provided by the client to the web server where a security component known as the SQL Injection Detector resides. It tests the values passed by the user for SQLi Attacks. If the values are valid, then it is sent to the Request Processor which processes the request and executes the query. Finally, the database is accessed, and the request of the client is processed and executed.

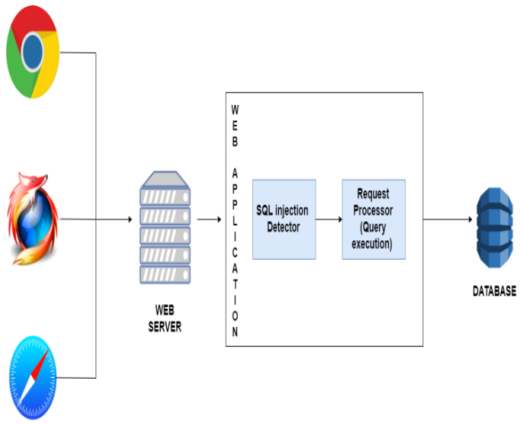


Fig. 1 System architecture

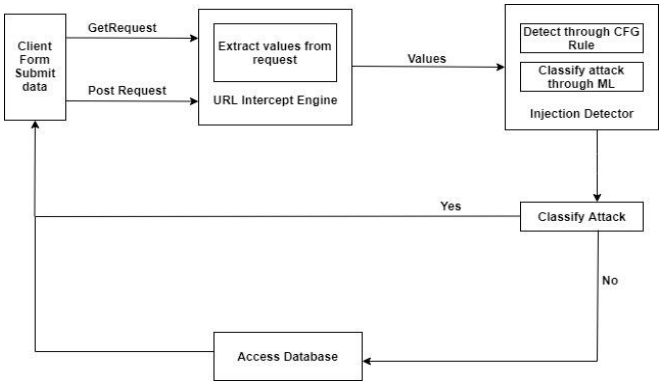


Fig. 2 Functional Architecture

This system consists of three modules, namely, URL Intercept Engine, Context-Free Grammar for SQLi Attacks and Classify pattern through Machine Learning.

Figure:2 shows the Functional Architecture, describing how the system works. First the client submits the data in the form. The request is sent to the URL Intercept Engine where the values are extracted from the request. The values are then passed to the Injection Detector which checks for pre-defined patterns using CFG and then classifies attack using K-means clustering. If there is no attack, the database is accessed with the help of the values and then the response is given to the client. If there is a attack, the access is prevented and the attack type is found.

### A. URL Intercept Engine

The text fields in the web applications are the most vulnerable part where most of the sql injection attacks happen. The end user or hackers who are intended to affect the database of an organisation, enter some malicious sql queries in the text field which can be appended to the sql query, already defined in the server side and thus, affects the database. So the first module extracts the values given in the text fields and sends the values to pattern checking algorithm. Figure(3) illustrates how the values are extracted by the URL Intercept Engine.

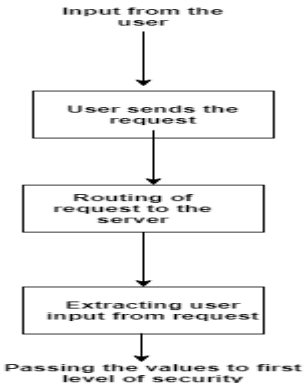


Fig. 3 Proposed Approach of URL Intercept Engine

## B. Context Free Grammar for SQLi Attacks

The second module is the first level of security from the sql injection attacks. This module consist of context free grammar rules for the sql injection attacks that generates the sql injection attack pattern. The value extracted from the first module is checked with the rule that generates the different attack patterns. If the value extracted, matches the attack pattern generated by the Context free grammar rules, the value is passed to the second level security that determines whether the value entered is malicious or not. Even if the value entered does not match the pattern generated by the Context free grammar rules, the value is passed to the second level security. The Context free grammar rules look like,

Start ------>Boolean Attack
Boolean Attack -------> Or Attack | And Attack
OrAttack-------->OpOr,wsp,terDigitOne,opEqual , terDigitOne
Where
opOr -----> 'or'
wsp -------> blank space
terDigitOne ------> '1'
opEqual ------> '='

These rules generates pattern until no terminals are left. Figure(4) illustrates the process of checking the value against generated patterns.

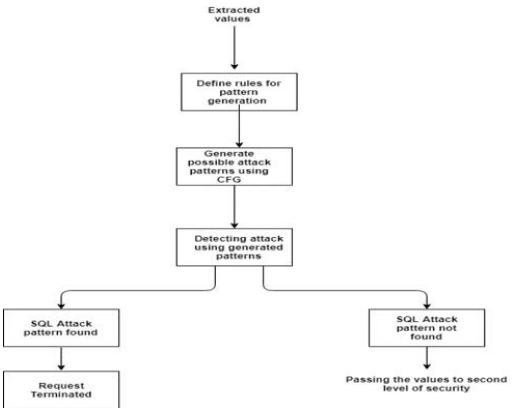


Fig. 4 Proposed Approach of CFG for Sqli Attacks

## C. Classification of Pattern Through Machine Learning

This module is the second level of security provided to the system. This module consist of unsupervised machine learning algorithm that classifies the different types of attack into different cluster and tests the value passed to it by

checking the fitness of the value to the cluster and determines whether the value has malicious query or not. If not, the system allows the request to execute the query and access the database. If it matches with the cluster associated with any attack pattern, the system prevents the request from executing and remains in the same state.

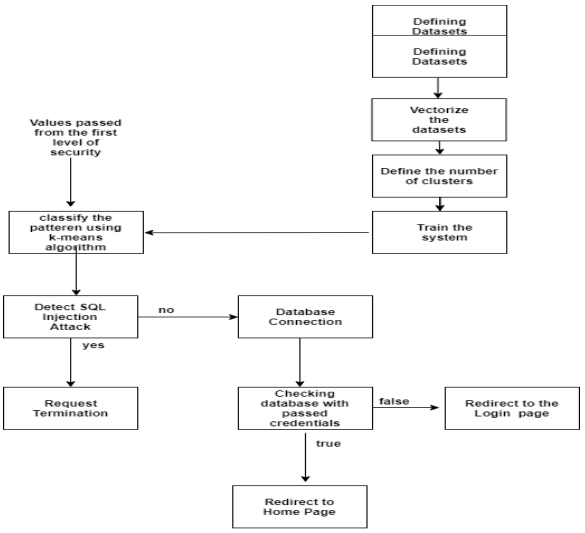


Fig. 5 Proposed Approach of Machine Learning Guided Attack Generation

## IV. EXPERIMENTAL RESULTS

The input to our model is from the forms submitted by the end user or any injected values to the url by the end users or hackers. All the values submitted in the form or the values injected in the url are finally inserted into a query and then the query is executed to access the database for retrieving or inserting the new data. The training datasets used for our Machine Learning Algorithm is the set of statements used in the SQL query language to classify the pattern used in piggy backed attack & union attacks and some of the attack patterns to classify the boolean type of attacks.

### A. Extract Values from Request

All the requests submitted by the user is received by the server and the server processes the request. The URL Intercept Engine is developed for extracting the values passed in the request submitted by the end users or the values injected by the end users or hackers.

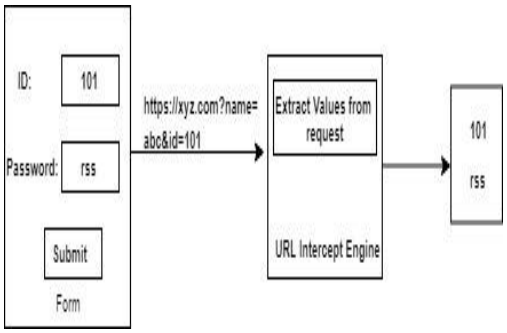


Fig. 6 Block Diagram of URL Intercept Engine

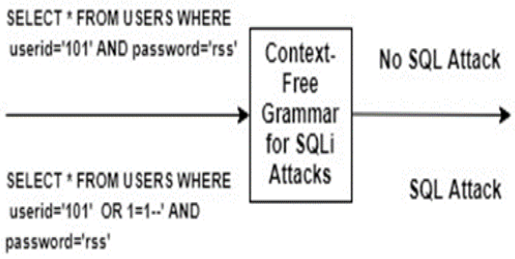


Fig. 7 Block Diagram of CFG for Sqli

### B. Detecting Attacks using CFG Pattern

SQL Grammar rules are defined for the different types of attack and the rules are generated to form different types of attack pattern that can be injected along with the values passed in the form. The pattern generated consists of the value passed by the end user and the different attack that can be concatenated with the values.

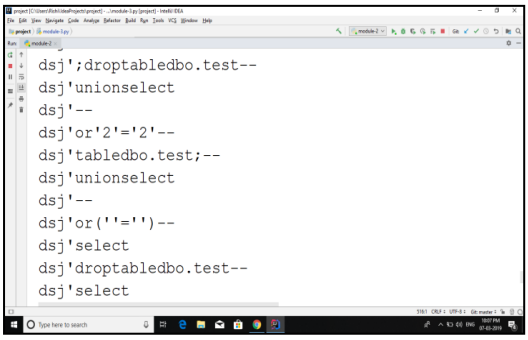


Fig. 8 Snapshot of Cfg Patterns

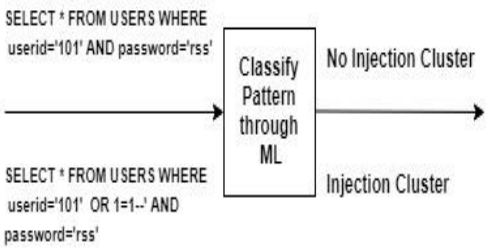


Fig. 9 Block diagram of machine learning

### C. Classifying Patterns and Detection

The Machine Learning Model is developed using K-means Algorithm and uses id vectorizer. The dataset for the algorithm is the set of keywords used in the SQL query and some attack patterns. The model will learn from the dataset and able to classify the type of attack and detects the attack.

## V. FINAL RESULT


Fig. 10 Snapshot of user input

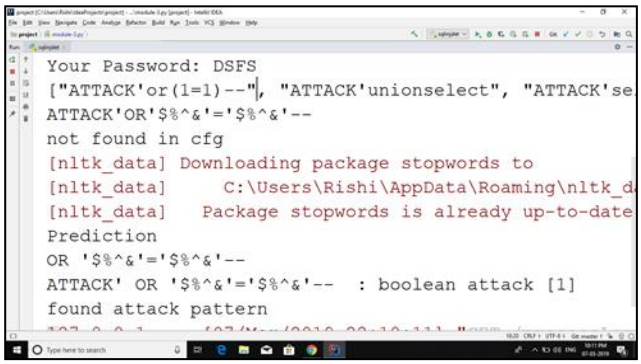


Fig. 11 Snapshot of Output of Sql Injection Detection

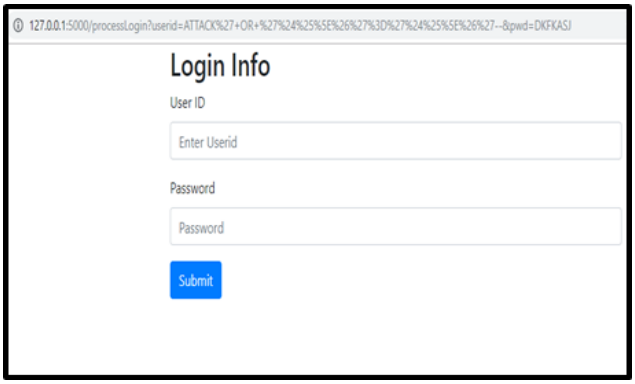


Fig. 12 Snapshot of Prevention of Request

## VI. CONCLUSION

In this paper, the system proposed uses a machine learning approach to detect and prevent SQL injection attacks via user requests in a web application. There are three modules in the system, namely, URL Intercept Engine, Context-Free Grammar for SQLi Attacks and Classify pattern through Machine Learning. At first, the values entered by the user on the client side is extracted using URL Intercept Engine which is then passed to the first level of security. The first level of security detects the pattern using the Context-Free Grammar rules. The patterns are passed to the Machine Learning Algorithm which classifies the value to clusters based on the pattern. If the value is found malicious by the implemented security, then the request made by the client will be revoked. If not malicious, the request will be executed normally. This system makes effective use of k-means clustering algorithm for the efficient detection of SQLi attacks. The accuracy and time taken to cluster using k-means is much better than the other algorithms. Future enhancements of this project can include the prevention of Cross-Site Scripting (XSS) and Path Traversal Attacks.

## REFERENCES


[1] A. Doupe, M. Cova, and G. Vigna, “Why johnny can ´ aˆA˘Zt pentest: An analysis of black-box web vulnerability scanners,” in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, 2010, pp. 111–131.

[2] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, “Automatic creation of SQL injection and cross-site scripting attacks,” in Proc. 31st Int. Conf.Softw. Eng., 2009, pp. 199–209.

[3] A. Liu, Y. Yuan, D.Wijesekera, and A. Stavrou, “Sqlprob: A proxy-based architecture towards preventing sql injection attacks,” in Proc. 2009 ACM Symp. Appl. Comput., 2009, pp. 2054–2061.

[4] Dennis Appelt, Cu D. Nguyen, Annibale Panichella and Lionel C. Briand, “A Machine-Learning -Driven Evolutionary Approach for Testing Web Application Firewalls”, IEEE Transactions on Reliability, vol. 67, pp. 733 - 757, 2018.

[5] D. Appelt, N. Alshahwan, and L. Briand, “Assessing the impact of fire-walls and database proxies on SQL injection testing,” in Proc. 1st Int.Workshop Future Internet Testing, 2013, pp. 32–47.

[6] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, “Automated testing for sql injection vulnerabilities: An input mutation approach,” inProc. 2014 Int. Symp. Softw. Testing Anal., 2014, pp. 259–269.

[7] E. Al-Shaer, A. El-Atawy, and T. Samak, “Automated pseudo-live testing of firewall configuration enforcement,” IEEE J. Sel. Areas Commun.,vol. 27, no. 3, pp. 302–314, Apr. 2009.

[8] Inyong Lee, Soonk Jeong, Sangsoo Yeo and Jongsub Moon, ‘A novel method for SQL injection attack detection based on removing SQL query attribute values’, Elsevier, vol. 55, pp. 58 - 68, 2012.

[9] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, “State of the art: Automated black-box web application vulnerability testing,” in Proc. IEEE Symp. Security Privacy, 2010, pp. 332–345.

[10] J. Hwang, T. Xie, F. Chen, and A. X. Liu, “Systematic structural testing of firewall policies,” in Proc. IEEE Symp. Rel. Distrib. Syst., 2008, pp. 105–114.

[11] J. Jurjens and G. Wimmel, “Specification-based testing of firewalls,” in Perspectives of System Informatics (Lecture Notes in Computer Science,vol. 2244), D. Bjørner, M. Broy, and A. Zamulin, Eds. Berlin, Germany: Springer, 2001, pp. 308–316.

[12] L. K. Shar and H. B. K. Tan, “Defeating sql injection,” Computer, vol. 3, pp. 69–77, 2013.

[13] M. Felderer, M. BAijchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, “Security testing,” Adv. Comput., vol. 101, pp. 1–51, 2016.

[14] O. Tripp, O. Weisman, and L. Guy, “Finding your way in the testing jungle: A learning approach to web security testing,” in Proc. Int. Symp.Softw. Testing Anal., 2013, pp. 347–357.

[15] P. Godefroid, A. Kiezun, and M. Y. Levin, “Grammar-based whitebox fuzzing,” ACM Sigplan Notices, vol. 43, pp. 206–215, 2008.

[16] S. W. Boyd and A. D. Keromytis, “Sqlrand: Preventing sql injection attacks,” in Applied Cryptography and Network Security. New York, NY,USA: Springer, 2004, pp. 292–302.

[17] W. G. Halfond and A. Orso, “Amnesia: Analysis and monitoring for neutralizing sql-injection attacks,” in Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng., 2005, pp. 174–183.

[18] W. G. Halfond, S. Anand, and A. Orso, “Precise interface identification to improve testing and analysis of web applications,” in Proc. 18th Int.Symp. Softw. Testing Anal., 2009, pp. 285–296.

[19] Witt Yi Win, and Hnin Hnin Htun, “A Simple and Efficient Framework for Detection of SQL Injection Attack”, IJCCER , vol. 1, pp. 26 - 29, 2013.

[20] W. Halfond, J. Viegas, and A. Orso, “A classification of sql-injection attacks and countermeasures,” in Proc. IEEE Int. Symp. Secure Softw.Eng., 2006, vol. 1, pp. 13–15.

[21] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, “A static analysis framework for detecting sql injection vulnerabilities,” in Proc.31st Annu. Int. Comput. Softw. Appl. Conf., Jul. 2007, vol. 1, pp. 87–96.

[22] Y.-F. Li, P. K. Das, and D. L. Dowe, “Two decades of web application testing: A survey of recent advances,” Inf. Syst., vol. 43, pp. 20–54,2014..