© IJCIRAS | ISSN (O) - 2581-5334
September 2020 | Vol. 3 Issue. 4

WEB APPLICATION FIREWALL: REVIEW

Muhammad Saidu Aliero[1], Bilyaminu Isah Shamaki[2], Ibrahim abubakar[3], Bello shamsudden kalgo[4], Abdul-azeez Muhammad Bello[5]

[1]ICT, Kebbi State University of Science and Technology Aliero, Nigeria
[2]National Space Research and Development Agency, Nigeria
[3]Asset management department Nigerian deposit insurance corporation, Nigeria
[4]Ministry Of Health Birnin Kebbi, Nigeria
[5]Waziri Umaru federal polytechnic Birnin kebbi, Nigeria

Abstract

SQL injection attack (SQLIA) is one of the most severe attacks that can be used against web database driving applications. Attackers use SQLIA to get unauthorized access and perform unauthorized data modification. To combat problem of SQLIA, different researchers proposed variety of tools and methods that can be used as defense barrier between client application and database server. However, these tools and methods failed to address the whole problem of SQL injection attack, because most of the approaches are vulnerable in nature, cannot resist sophisticated attack or limited to scope of subset of SQLIA type. with regard to this different researchers proposed different approach (experimental and analytical evaluation) to evaluate the effectiveness of these existing tools based on type SQLIAs they can detect or prevent. However, none of the researcher considers evaluating these existing tool or method based on their ability to be deployed in various injection parameters or development requirements therefore, in this we analytically evaluated the reviewed tools and methods based on our experience with respect to SQIAs types and injection parameters. The evaluation result showed that most researchers focused on proposing approaches to detect and prevent SQLIAs, rather than evaluating the efficiency and effectiveness of the existing SQLIA detection and prevention tools/methods. The study also revealed that more emphasis was given by the previous studies on prevention measures than detection measures in combating problem of SQLIAs. An analysis showed that these tools and methods are developed to prevent subset of SQLIAs type and only few of them can be deployed to various injection parameters to be considered in examining SQLIAs. It further revealed that none of the tools or methods can be deployed to prevent attacks that can take advantage of second order (server side SQLIA) SQLI vulnerability. Finally, the study highlights the major challenges that require immediate response by developers and researchers in order to prevent the risk of being hacked through SQLIAs.

Keyword: Malicious link, web application, web application security, web application vulnerably dynamic approach, analytical evaluation

1.INTRODUCTION

Web applicatios are associated with diffirent types of vulnerabilities such as Cross- Site –Scritping (XSS), SQL Injection, File Includion, Brute Force among other vulnerabilities. The most common techniques by web application can be prevented against malacious request is to deployed web application firewalls. A web firewall is a system for detecting of web application attacks. Web firewalls are used for a variety of purposes. Most prominently, they are one of the main barriers between stored database and client accessing the data to prevent it from SQL injection attacks. SQL injection attack is attacks can be performed against web driving database application to execute un-authorized data manipulations and retrievals.

Web firewall can be use as barrier against SQL injection attacks. Most of studies argue that the best approaches by which filter can be applied to differentiate- between malicious and valid request to application, such as blacklisting, whitelisting, pattern matching. However,

attack score is getting increase every year regardless of firewall deployed in various applications.
Therefore this study aimed to explore current detection and prevention tools and mechanisms and identify weaknesses and recommend future improvements. The result of our analysis could be serve as based evident for future improvement by a researchers.

2. BACKGROUND OF WEB APPLICATIONS ATTACKS

Expert shows that new vulnerabilities that are found in cyber environment are much higher in applications level than in operating system. There are number of standard organizations such as System Administration, Network and Security (SAN) Open Web Application Security Project, and so on, that keep track of new found vulnerabilities as well as vulnerabilities that present higher risk to any organization adopting web applications as backbone of their business. These organization filed monthly and yearly report regarding new and top vulnerabilities presenting high risk to web application to create awareness to the people in order to reduced risk of being
hacked as result of such vulnerabilities. Below are the description of top 10 2013
(Gartner, June 19, 2014, OWSAP, 2013).

A1. Injection flows: Injection flaws (such as SQL, OS, and LDAP injection) are type of web application vulnerabilities that occur when inputs from user are being sent to interpreter without proper sanitization. This allow attacker to trick interpreter by processing unintended query or command thereby gaining unauthorized access or data manipulations

A2. Broken Authentication and Session Management: Broken authentication and session management occur when functions associated with authentication and session management are not well implemented. This allow attacker to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

A3. Cross Site Scripting (XSS): Injection flows take un-trusted data and send it interpreter without sanitization while XSS flow occur when web applications are allowed to received un-trusted data without proper sanitization. This allow attacker to write script that enable them to redirect user into malicious page or site thereby stealing user information as well as session hijacking.

A4. Insecure Direct Object Reference: It is common mistake by programmer to expose object reference to internal implementation such as file, directory, and database key and so on to the end user. This practice allow attacker to manipulate these reference to gain unauthorized access to restricted information.

A5. Security Miss-configuration: Using default configuration without any security in mind increases the chance of being hacked. It is always recommended to reconfigure, application server, web server, database server, and platform. Security miss-configuration allow attacker to exploit default configuration vulnerability exist in technology being used in application. Likewise lack of frequent update of software technology result in exploiting new found vulnerabilities in application.

A6. Sensitive Data Exposure: Sensitive data exposure occur when web application transferring confidential information such as monetary transaction, user credential without proper protection. This practice enable adversary to intercept, modify, reply to users transactions. Sensitive data exposure can be reduced by applying proper cryptographic technique to communication medium.

A7. Missing Function Level Access Control: Applying authentication to access application functionality is very important; however applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

A8. Cross-Site Request Forgery (CSRF): XSS vulnerability enable attacker to impersonate trust that exist between web application and user by secretly tricking user to access malicious page, while CSRF exploit trust between user and web application by forcing the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

A9. Using Components with Known Vulnerabilities: Using vulnerable component that run with full privileges allowed attacker to escalade an attack by exploiting result in serious data loss or server takeover. Applications using components with known vulnerabilities may weakened application defense and enable a range of possible attacks and impacts.

A10. Un-validated Redirects and Forwards: User interaction with Web applications frequently involved redirect and forward from to other pages and websites.

Thus using un-trusted data without proper validation, attackers can redirect users to phishing or malware sites, or use forwards to access unauthorized pages. Despite danger that above mentioned web applications vulnerabilities present to enterprise information asset, this project will focus on SQL injection Vulnerabilities with emphasis on developing tool that will be use in performing SQL injection web firewall assessment.

2.1. Background of SQLI attacks

Most database driving applications required users to log into the application order to have access the information stored in information system. By login into the system users can have full access to the information to him cannot or have limited access other's information depending on the purpose of the application. However, because of the dynamic feature of SQL query and logical knowledge possessed by other people on how communications between application layer and database layer are constructed, it became possible for them manipulate these commutation to have unauthorized access to the system, bypass authentication mechanisms and unauthorized data manipulation on backend database through injection parameters (input provide to user to make request to application such as forms) without being login into the system or without proper login credential (Bau et al., 2010).

This is possible because developer of the system trust the end users by not considering security threat at a time of developing the query which is handling the users request, processes it and send respond back to the users. A query that is accepting whatever

input provided by the user and send it to the backend database of the application for processing without proper security check is called vulnerable query and can be subjected to SQL injection attack. The more details on how SQL injection attacks are been discovered can be described in the sections below.

2.2.1. Injection Parameters

(Halfond and Orso, 2007) and (Sadeghian et al., 2013) points by which an attackers inject SQL injection attacks into following injection through user input field, through cookies, through server variables, second order injection as shown in fig 1.0.

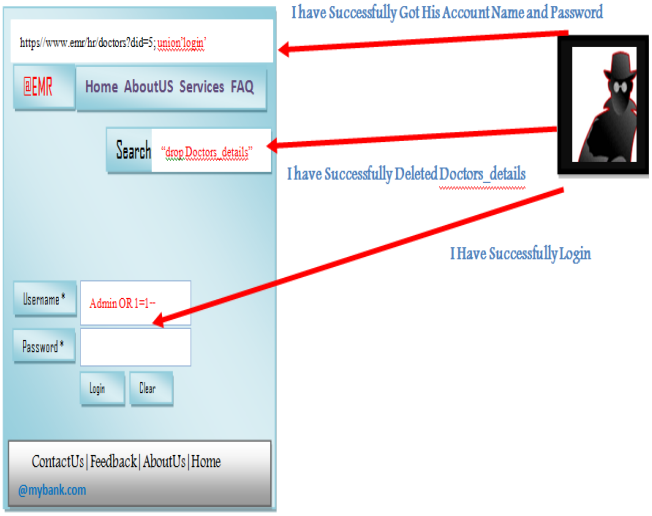

Fig 1.0 Injection parameters

Through User input field: user input fields are provided in web applications to enable web application users to request information from backed database to the user with help of HTTP POST and GET. These inputs are connected with backend database using SQL statements to retrieve and render requested information for users or to allow users to connect to the system. User input fields are vulnerable to SQL injection attack if input provided by the user is not sanitized before sending to the database engine for processing, which enables attackers to modify intended queries in order to perform malicious action in the system.

Injection through cookies: Cookies are structures that maintain persistence of web application by storing state information in the client machine. When a client returns to a Web application, cookies can be used to restore the client's state information. If a Web application uses the cookie's contents to build SQL queries, then an attacker can take this opportunity to modify cookies and submit to the database engine.

Injection through server variables: Server variables are a combination of variables that include HTTP, network headers, and environmental variables. Web applications use these server variables in variety of ways, such as monitoring statistical usage of and determining browsing trends. Using these variables in web applications without proper validation, will course injection vulnerabilities which allows attackers to change values that are in HTTP and network headers by entering

their crafted input into the client-end of the application or by injecting malicious request.

Second order injection: In second-order injections, attackers plant malicious inputs into a system or database to indirectly trigger an SQLIA. When that input is called at a later time when an attack occurs, the input that modifies the query to construe an attack does not come from the user, but from within the system itself.

2.2.2. SQLIA (SQL Injection Attack) Intent

Attacks can be classified based on what attacker trying to achieve or intent to do (Amirtahmasebi et al., 2009) and (Halfond and Orso, 2007) Identifying injection parameters, database fingerprinting, identifying database schema, database extraction, executing remote code, performing privilege escalading and authentication bypassing.

Identifying Injectable Parameters: Injectable parameters are text-input that allow users to request information from the database. This query request is sent to the database server though HTTP request, for example ULR; search box and authentication entries are considered as text-input. When these text-input are sending user requests to the database without proper validation they are considered as injectable parameters which allow attackers to inject SQL query attack. Identifying injection parameters is the first step to perform an attack.

Performing database fingerprinting: after identifying the injection parameters the second step is to know the database engine type and version. Knowing this is very important to an attacker because it enables him to know how to construct query format supported by that database engine and default vulnerability associated with that version as every database engine employs a different proprietary SQL language dialect.

Determine database schema: schema is the database structure. It includes table names, relationships, and column names. Knowing this information about database makes it easier for an attacker to construct an attack to perform database extraction or manipulate data language.

Database manipulation and extraction: most of the attacker their aim is gaining access to sensitive information such as secret formula employee bank details or changing friend salaries.

Evading detection: well structure attacks are always very difficult to detect as result of attacker employed the used of technique that hide his foot step hide malicious data from security guards implanted so that their actions cannot be detected or traced.

Executing Remote Commands: file inclusion attack is one of the precious attack by attacker that enable them to include file that can be executed by database server, shell command in order to find their own way into the system. Once attacker file get executed it will start action that it was intended to do by attackers such as creating backdoors, sending secret information, corrupting system memory and so on.

Bypassing authentication: Bypassing authentication mechanism is the one of the common aim of almost the entire attacker, which allows them to gain access to the database with user privileges.

Performing privilege escalation: when function responsible for assigning privileges is exposed or was not properly validated it is possible for an attacker to use this opportunity to increase his privilege to the system as some of the database attack requires admin privilege to be performed.

2.3. SQLI Vulnerability Detection Approaches

Disconnecting enterprise from internet is not reasonable option to prevent SQLI attack. In order to minimize the likelihood of successful SQLIA in web application, researcher proposed different approach to enable security administrator to identify the course of vulnerabilities and address them before been exploit by potential attacker. These approach can be categorized into two static and dynamic approach (Djuric, 2013, Livshits and Lam, 2005), (Huang et al., 2004).

3. DEFENSIVE CODING

The primary motive of SQL injection vulnerability is mistaken validation of person input. Entry validation is a way by which programmers observe protection code exercise to cozy every static question manually. One of the targets of defensive programming is to write at ease queries so that it behaves in a predictable way in spite of sudden inputs or user moves. Underneath are some of the common ways by using which programmer observe defensive coding in an software. Entry type checking/data type validation: from time to time

programmers make easy mistakes by allowing input fields to simply accept different styles of information without figuring out that an attacker can take this opportunity to insert malicious enter to database engine. Second order injection attacks can be performed by using injecting instructions into either a string or numeric parameter. Even a simple check of such inputs can save many attacks. As an example, in the case of numeric inputs, i.e. if the sphere is a telephone quantity, the programmer can truly reject any input that incorporates characters apart from digits. This method however cannot guaranty that it's going to fully forestall the SQL injection but it makes the technique tougher for the attacker. occasionally an attacker issues sq. injection attack with a declaration that usually returns a value of actual in order that to the database engine interprets user enter as square so that when backend database engine executes this kind of statement it lets in the user to bypass authentication mechanisms or use meta-characters to carry out an unlawful question that allows you to trick the database engine into offering the attacker with some mystery data approximately the backend database. Applying encoding practice along with hashing, encryption, conversion of enter into ASCI format prevents attackers from tricking database engines.

White listening/fine sample matching: there are two number one principles of sample matching, blacklist and white listing. Blacklisting includes checking if the input carries unacceptable statistics at the same time as white list checks if the input includes desirable statistics. Programmer should establish input validation workouts that clear out bad enter and permit proper input. This approach is commonly called nice validation.

Instead of making dynamic queries by concatenating the parameters with SQL statement, it will replace the placeholders with the value of parameters at the runtime become aware of all input: Parameterized question is a sort of question which has some placeholders. In these instead of making dynamic queries by concatenating the parameters with SQL statement, it will replace the placeholders with the value of parameters at the runtime as shown in fig 2. Defensive coding and dynamic approach.

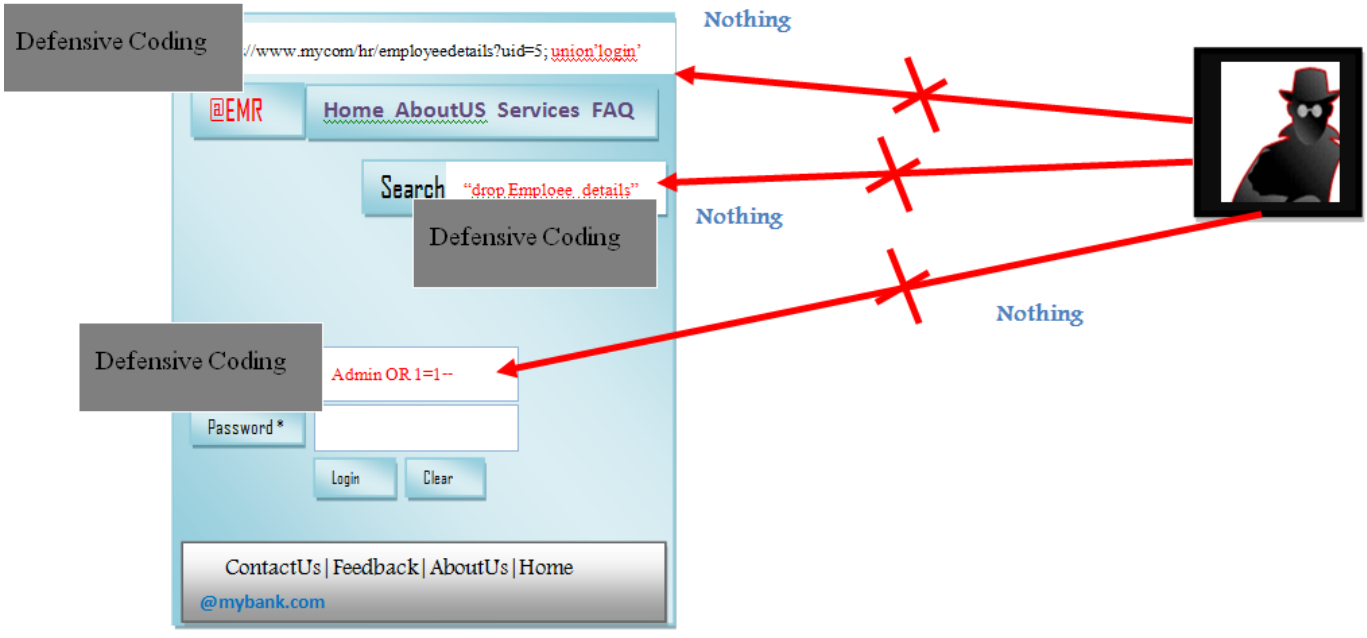

Fig 2.0 defensive coding approach

3.1. Disadvantage of defensive coding Approach

SQL injection firewall tools implementing static approach are known to be effective and very accurate in detecting and preventing SQL injection attacks with not

false negative or false positive .However this approach has major drawbacks that make it in adequate to apply or use in production systems such as intrusive instrumentation, high performance overheads language dependence (Garfinkel and Spafford, 2002), (Masanès, 2006, Takanen et al., 2008, Hurty, 1965).

- **Intrusive Instrumentation**: defensive coding approach need well gained transformation of the target application. Every queries in the application needs to be transformed to introduce additional statements that propagate taint. Such instrumentation can affect the stability and enhance detection of the target application, which result in making developer reluctant to use these techniques on production systems.
- **High Performance Overheads:** defensvie approach especially those developed in C or binary code have high tendency of overheads, often slowing down programs by a factor of two or more.
- **Language Dependencies**: This is the one of the features that makes developer to not design tool that implements defensive approach. defensive approach always required source code of application in order to perform security assessment on target application and it was shown that in (Garfinkel and Spafford, 2002) source-code based approach is language dependent and need to be redesigned and re-implemented for each language. Even for binary based techniques, it is not straight forward to apply them across all languages.

For example; applying a machine-code based taint-tracking to Java requires the JVM to be taint-tracked, which can pose challenges in terms of false positives and false negatives. As a result, previous techniques have either been applicable to Java or to C/C++/compiled binaries, but not both.

- **Security Exposure:**
- Defensive coding firewall tool usually required source of application to perform security analysis which raise the major challenge in security point of view. Not all users or enterprise allow access to their application, this is because knowing how application was designed will also make easier to attack application.

3.2. Dynamic Approach

Dynamic security analysis approach is used to detect and prevent attacks via attack pattern , frequency, signature. Although, recent studies shows that dynamic security analysis firewalls are less effective, and less accurate compare to static security analysis due to their inability to cover crawling activities, attacks needed to trigger anomaly and inability to performs deferent type of analysis on server response. However, even this issue developer and security
administrators prefers to implement firewall tools in dynamic approach because of the following reasons such as language compatibility, users friendly, no source code required and flexibility (Takanen et al., 2008), (Arkin et al., 2005).

- **Language Compatibility** : Most of the SQL injection vulnerabilities scanners implemented in C, Java, Python or any other programming language can performs vulnerability assessment in application in respect of the type of language target application was designed. For example static vulnerability scanner designed to assess PHP application cannot be effectively used to asses application design in Java platform.
- **User Friendly:** Static vulnerability scanner required the user to have knowledge of how vulnerability looks like in order to effectively determine whether vulnerability exist in the application. In case of dynamic approach user do not worry about how vulnerability are been discover in application in order to perform full security assessment on application (Luk et al., 2005).
- **No Source Code is Required:** This is the one of the major advantage of dynamic SQL injection assessment scanners over static one, which is ability to perform security assessment without providing the source code of application. This is because most of the organization/individuals do not approves the external parties to have access to the source code of their application which might raise other security concern (Garfinkel and Spafford, 2002).
- **Flexibility:** Ability for scanner to perform more than one analysis on responds return by the application server. For example most of the SQLI static vulnerability scanners can only detect particular vulnerability type while in dynamic approach developer can simulate as much as

number of analysis component needed to perform SQLIV analysis.

3.3. Review on Static and Dynamic SQLI Attacks

This section, present review of different existing techniques scanners that are implementing both static and dynamic approaches to vulnerability analysis. Consequently this section classified existing techniques and scanners into three categories; academic, open source and commercial scanner as described in (Gartner, June 19, 2014).

3.3.1. Academic SQLI firewall

Academic SQLI Attack represent those Attacks proposed by individuals in a field of research such as SQLI web firewall (SQL Injection web firewall) EnhancedMySQLinjector, secuBat, wave, Amnisia etc. Majority of proposed academic SQLI Attack are language specific, their developments are ongoing process and public access to these SQLIA is unavailable. However, method used in development of such tools are publicly available to shade light to individuals or academic researchers who want to improve existing tools or proposed new methods with enhanced features. This section is also classified into two approaches.
Approach number one that is focusing on proposed techniques SQLIA and approach number two focusing on evaluation of existing techniques SQLI web firewall (Djuric, 2013), (Masanès, 2006).

3.3.1.1. Proposed Techniques and firewall

Code miner was proposed in (Agosta et al., 2012) for detection of SQL and XSS vulnerabilities in PHP web-based applications. In this approach all PHP codes are converted abstract syntax tree, through the use of a parser. This enable firewall of easily extraction of the control and data flow information needed for analysis firewall start sink point detection in which source code are scanned and checks for the presence of sensitive functions. Their presence is ascertained through the comparison with a language-dependent, extensible knowledge base that is provided as input to the tool. So all sink points are characterized by different vulnerability patterns defined by programmer in firewall.

After detection of the sink points is completed firewalls starts examining the abstract syntax tree provided by the parser to perform Static taint Analysis and extract information on the chain of modification which every variable undergoes each vulnerable function encountered scanner make it taint. The idea of using abstract syntax tree is to construct valid semantic feature that enables firewall o further extend our tool to analyze code in other programming languages with minimum effort. However static analysis has their inherit problem for security tester because some cases most company doesn't want give source code of application for some reasons.
In (Zhang and Wang, 2010) static code scanner was proposed to detect XSS and SQI vulnerabilities. This scanner tend to examine source code with the objective of finding vulnerability by using morphological analysis and semantic analysis in order to produce semantic tree and control flow program. Examination of the control flow program is conducted based on analysis rules which would then report vulnerability of the exact input program. The idea is to analyze the control flow program and semantic tree based on spot broadcast algorithm in order to detect the vulnerability of input variables without data sanitation. Using static code analyzer to
detect vulnerabilities required at least basic knowledge of code analysis and this approach can only work on ASP web applications.
Dynamic Web Application firewall (D-WAF) was proposed in (Huang et al., 2004) to detects XSS and SQLI vulnerabilities. This firewall consist similar component used in (Liban and Hilles, 2014) SQLI web firewall, attacking and analysis component. It takes seed URL as its input, visits all Web Pages and stored the page with injection parameter, it then later takes stored links and forms and generates attack that would be lunch against these links and forms. Experimental result show the scanner faces challenges in identifying forms that required partial refreshment and login authentication. Beside this D- WAV also does not have functionality to predict whether application is vulnerable if it's not responding to SQL query related errors if abnormal request was received from client.
Static code firewall was proposed in (Livshits and Lam, 2005) that uses static code analysis to detect SQL injection Attack and prevent attacker from exploiting such vulnerability in java web-based application. In this

approach PQL was used as a syntactic model for queries library, which allow users to define vulnerability patterns in a familiar Java-like syntax. Any piece of code that accepts input parameters from user and are passed to the backend database are marked tainted and tracked until it used in a sink. The advantage of this approach is that it enables detection of all potential security violations early, even without executing the application. However this approach requires source code application to carry out this function. In addition, it cannot detect unknown patterns of SQL injection attack.

DIGLOSSIA is a tool that detects and prevents SQL injection attack by computing shadow values for the results of all string and character array operations that depend on user input. In this approach programmer defines valid queries in the form of a tree-like structure to compare against dynamic queries entered by the user (Bau et al., 2010). When input query is sent to database the tool intercepts the query and tries to construct a tree like a dynamic query based on queries already defined by the programmer and also computes the shadow of the entered query storing it in the shadow value table indexed by the address of the memory location for the original value, performing both grammar and shadow checks using a dual parser. Using the dual parser to detect injected code is based on the idea that query strings can be parsed to either its original grammar, or the shadow grammar. If the tool cannot produce tree-like structure of query, the tool rejects the query and reports a code injection attack. Otherwise, it compares the query with its shadow to check whether the query is syntactically isomorphic, and that the code in the shadow query is not tainted. If either condition fails, it considers the dynamic query as an attack. The problem with this approach is that when users input non-malicious queries that are supported by database engine but violate the rule of query code in DIGLOSSIA it will consider that query as an attack. This method is totally based on the idea that when the web application submits the query any input type by the user will considered as an attack.

SQL UnitGen is a tool that uses static analysis to detect and prevent SQL injection attack. It uses unit case that is library that lists a number of attack patterns which helps to detect existing SQL injections in a dynamic query (Shin et al., 2006).

This method cannot detect new or existing attacks whose pattern has not been addressed in the unit test library.

AMNESIA tool that combine static and dynamic approach to SQL vulnerability by building static queries (predefine queries) and these queries are compare against query enter by user at runtime. Thus, in this approach all queries entered by user are being checks to see if the query complies with model defined in the static phase. If the query matched the model it allows execution in the database engine, otherwise it is blocked. The accuracy of AMNISIA depends on the accuracy of the developed Queries model. The authors show in the evaluation that their technique was capable of addressing all attacks.

Intrusion Detection System (IDS) was proposed in (Valeur et al., 2005) that utilizes multiple anomaly detection models to detect attacks against back-end SQL databases based on machine learning approach. In this approach HTTP POST, and GET request are intercepted and IDS selects what features of the query should be modeled using training data set in training phase. This starts when feature vectors are created by extracting all tokens marked as constant and inserting them into a list in the order in which they appear in the query. After features were extracted then different statistical models are used depending on what data type model is selected. If a dynamic query does not match the model, the query will consider it as an attack.

After evaluation IDS was found to be effective in detecting all kinds of SQL injection attack with false positive result.

In (Bandhakavi et al., 2007) CANDID was proposed to detect and prevent SQL injection attack. In this approach dynamic queries are mined at runtime and compared with legitimate query statements. If the result is not the same, it is a SQL injection attack. CANDID's natural and simple approach turns out to be very powerful for detection of SQL injection attacks.

Technique that uses key-based randomization called SQLIA and was proposed in (Boyd and Keromytis, 2004) which enables programmers to develop queries using instruction randomization without using SQL keywords. In this approach when attacker modifies the dynamic SQL query and sends to the database the proxy will intercept it and compare it with queries that the programmer created using instruction randomization which enables SQLrand to detect malicious queries since

dynamic queries were not created using instruction randomization. Experimental evaluation show the effectiveness of this approach but this approach has a number of drawbacks. However, the security of SQLIA and defense on attacker capability to compromise the key.

CSRS is the scanner that uses dynamic approach to detect injection vulnerability which undergoes series of activities to detect SQLI Attack.

Crawling, attacking and analysis. One of the important features in CSRS crawler is that it was designed to keep track of visited page and non-visited page, ability to handle partial page refreshment and authentication issues. Implementing state aware crawler is not an essay tasks as it is backbone to detection of SQL injection vulnerabilities (Singh and Roy, 2012). The scanner also has database of attacks that can be use to lunch attack against target applications. Due to the inability of scanner to perform different response analysis CSRS is only capable of detecting first order SQL injection vulnerability.

Two similar approach Mysqlinjector and Enhanced Mysqlinjector proposed by (Shakhatreh, 2010) and (Liban and Hilles, 2014) respectively. Both scanners composed of three components designed to detect SQL injection vulnerability in PHP-based web application. Enhanced Mysqlinjector is updated version of Mysqlinjector designed to address problem of false positive report by Mysqlinjector because Mysqlinjector does not have ability trigger and analyzed SQLI vulnerability that can be exploited using time-based SQLI attacks.

Therefore author of Enhanced

Mysqlinjector update database of attack used by Mysqlinjector with SQLI time based attack and also dataset that enable scanner to deduced existing of such vulnerability in target application.

WEBSSARI uses content management that was developed using taint analysis. In this approach scanner will start by inspecting the source code of application line-by-line and apply taint to the point that are potential to injection attacks (Huang et al., 2004). According (Shin et al., 2006) Taint analysis is effective and accurate way to check for vulnerability in application. However, scanners implementing taint analysis approach are language specific; that is to say they are only capable of detecting vulnerability in one type of language. WEBSSARI was tested on vulnerable application and compared against other scanners using accuracy metric and result of evaluation shows that there is need to improve the effectiveness.

SEFELI uses static code analysis approach to determine SQL injection vulnerability. SEFELI was backed with knowledge library database defined by programmer which helps in identifying un-sanitized input point (Fu et al., 2007).

When it reaches any injection point, the threshold library containing knowledge of defect function is consulted, identify vulnerable injection point based on constraint is constructed by programmer. SEFELI show a good result when author tested it on ASP-based application but fails to detect vulnerabilities on other languages. Using static code analysis tools has inherit limitation on identifying vulnerabilities of other language other than language that scanner developed to test against.

SQL Injection Vulnerability Scanner (SQLIVS) was proposed (Djuric,2013)to simulate different type of SQLI attacks type in attack component which enable the scanner to use this attack to lunch attacks against target applications using injection parameters. The author modified plagiarism application developed by (Cook and Rai, 2005) to analyze the content of the page in order to predict whether application is vulnerable to SQL injection vulnerability. Once page or forms is attacked the application has parser that takes the whole HTML page content to enable application performs analysis very easily As result of lack of number of attacks pattern required to trigger vulnerabilities the scanner failed to detect most of the second order SQL injection vulnerability in tested applications.

Black Box Testing Scanner (BBTS) was developed in (Chen and Wu, 2010) to find SQL injection vulnerabilities using dynamic approach. The BBST uses state aware crawler to identify forms and links with injection parameters to In this approach the authors' uses state aware scanner that will able to recognized webpage that contain injection parameter. This improvement enable scanner to save time by not downloading pages that does not contain injection parameter and also avoid false positive that result in from attacking pages that does not have injection parameter.

Author experimental evaluation show the scanner achieved 100% accuracy on tested application in short period of time. However author did not evaluate his approach with available existing technique to evaluate both accuracy and efficiency of the proposed scanner.

Viper is black box firewall proposed in (Ciampa et al., 2010) that detects SQLI vulnerability through dynamic testing. Scanners that implement dynamic approach are required to have different number of attacks type as well as way of analysing server response. Viper uses different number of predefined attack pattern to trigger hidden vulnerability in application but it is capable of performing single analysis on server response to attacks. Experiment shows that Viper was able to carry successful attacks that trigger blind SQL injection vulnerability but Viper was not able to report it, this is because analysis was not address to analyzed blind SQL injection vulnerability.

Static firewal was proposed in (Shar and Tan, 2012) that detect vulnerable point by characterizing input function into pattern of code attributes. Static code attributes are collected from backward static program slices of sensitive program points, so that to mine both input sanitization code patterns and input validation code patterns, from such static code attributes. This firewal uses vulnerabilities prediction model to enable scanner to predict vulnerable code for SQLI and cross site scripting (XSS) vulnerabilities. Authors evaluated their scanner with different PHP web-based application source code, which shows the effectiveness of their approach. However their approach can only be effective in PHP web-based applications.

4SQLi attacks that combine both static and dynamic approach to detect SQLI vulnerabilities (Appelt et al., 2014). 4SQLi uses a single or multiple mutation operators of different types that can be used as a single input parameter to generate desired inputs which will use latter for detecting subtle vulnerabilities that can only be triggered with an input generated by combining multiple mutation operators. For example, consider an application that alters inputs by searching for known attack patterns that can be generated using one of the behaviour-changing operators. To form a successful attack, it is necessary to apply a behaviour-changing operator and then apply one or more obfuscation operators.

SecuBat as proposed in (Kals et al., 2006) to detect XSS and SQLI vulnerabilities. SecuBat implement four stage of operation in trying to detect and report vulnerabilities. SecuBat begin by crawling all pages in a given seed URL, injection malicious request to injection point so that to make database server to respond with default configuration error messages. SecuBatdetect vulnerabilities by analyzing the error messages return by database server. SecuBat have similar limitation as other scanner by relaying on return error messages to detect vulnerabilities. Madhane proposed R-WASP tool to detect and prevent SQL injection attack.It intercepted dynamic queries entered by the user and breaks them into tokens of SQL keywords, operators and characters in order to track existing malicious input in the query (Madhene 2013). If all input tokens are found to be trusted then the query is considered to be safe and allowed processing by database engine. Otherwise action is performed as defined by the programmer. Using SQL keywords, operators, and characters to find the malicious input in a dynamic query is problematic in nature, as it is possible to have a valid query with delete or drop keywords.

Two similar techniques was proposed SQL DOM and Safe Query Objects in (McClure and Kruger, 2005) and in (Cook and Rai, 2005) respectively. In these approaches queries to the database are decoded so as to prevent attacker from gaining unauthorized access to the database. Changing the procedure of queries building is one of the efficient ways to prevent injection attack. However, approach is problematic in nature as always it produced false positive and encoding key can be compromised if not properly handle.

Scott and colleague proposed Security Gateway to detect and prevent SQLIA. Security gateway was designed to implement queries policy in which each query is annotated with message authentication code (MAC) (Scott and Sharp, 2002).Security Gateway act as a web firewall, it monitor HTTP request and response. A query is said to be malicious if the requested query does not match static query produce by MAC at runtime request. This method is very effective in identifying modified dynamic queries, however this approach is problematic in nature as it requires programmer to know each and every query in application and when every new query is added it requires programmer to update queries log.

Liu and colleagues proposed SQLProb that uses static and dynamic approach to detect and prevent SQLIA. In static phase query a collector was used to generate parse tree structure of legitimate queries from query repository as defined by the programmer which will be used to compare semantic structure of dynamic query (Liu et al., 2009). However, in dynamic phase when user inputs queries, the queries are compared against

semantic tree structure of legitimate queries created in phase and if the structure of dynamic query matches with the structure in a generated tree like structure query, queries are allowed; otherwise they are prevented and consider as malicious. This technique employs a similar approach used by (Shin et al. 2006)
and the accuracy of this approach depends on how accurate was the parser tree model that was developed.
Literature (Buehrer et al., 2005) proposed SQLGuard and in this approach input queries dynamically generate, through concatenation, a string representing an SQL statement and incorporating user input which generates and returns a new key by the database. SQLGuard validates dynamic queries by building two parse tree structures of dynamic query. First tree structure has unpopulated user tokens for dynamic query the second tree is the result of parsing the string with these nodes filled in with user input. The two trees are then compared for matching structure. If the structure marched, the query is allowed for execution; otherwise it is blocked.
This approach tends to be slow as data comparison takes much time to process in tree structure model as each node must be processed. The accuracy of this approach depends on whether or not the attacker discovered the key.
Literature (Cheon et al., 2013) proposed machine learning method using Bayesian algorithm to detect and prevent SQL injection attack. In this approach monitor capture dynamic SQL query HTTP POST and GET, send it to converter which breaks SQL statement into a number of keywords based on black space in statement and calculate the length of dynamic SQL query. It also calculates the number of keywords present in such a query and sends a numeric value of length and keyword of dynamic query to the classifier. The classifier then calculates the probability of SQL injection in dynamic query based on results received from the converter, and then compares the probability of SQL injection calculated with one defined by user threshold as training dataset which consists of the probability of legitimate query and probability of malicious query.
When the probability of dynamic SQL query calculated by classifier matches the probability of legitimate query in training dataset the query is allowed; otherwise it is blocked. One important thing in this method is that it simulates a high number of attack patterns in training data including blind SQL injection attack which is very difficult to address. However this method requires programmers to fully define and carefully train data set because the accuracy of this approach depends on how accurate was the trained data.
Literature (Joshi and Geetha, 2014) proposed method that uses machine learning to detect and prevent SQL injection attack. In this method training dataset was constructed by analyzing source code program of the application and calculating the entropy of static SQL query. The main purpose of entropy is to count the average amount of information needed to identify the class model of a training dataset. In this case entropy of all static queries that are implemented in a website was calculated which was used to construct training dataset which will be used later for comparison. When a user issues SQL query the entropy of dynamic SQL query is calculated and compared with entropy in training data. If a match is found the query is allowed to execute in database engine; otherwise it is blocked and prevented from parsing to the database engine. Using entropy in machine learning to classify queries has advantages over using probability as used because it produced better results. When data are categorized instead of using continuous-valued, small changes in SQL query will yield a great effect when the entropy of that query is calculated. The disadvantage of this method is that it requires analysis of the application of the source code.
In (Shahriar and Zulkernine, 2012) similar method used in (Cheon et al 2013) was proposed to detect and prevent SQL injection attack. In this approach black space method was used in breaking SQL stamen into keywords but here length of the query was not considered. After tokenizing SQL statement user action was then considered in generating training dataset in which system user was categorized into three namely visiting user, normal user and admin and a different role was assigned to each.
This user classification helps classifier to determine which model to use in training data to compute and compare probability of SQL injection in user query. For example in normal user query keywords considered as malicious are dropped so when visiting users issue SQL statements with drop keywords this query will be automatically considered as malicious before computing probability which allows the method to use two probabilities in computing user queries. The first is prior probability which assumes the query is malicious if it contains some malicious keywords and posterior

probability which can be obtained after comparing calculated probability of dynamic query against its model in training. Advantage of using prior probability and posterior probability is that they help to reduce false positive result.

However the issue of stacked query was not addressed in this method which allowed attackers to perform piggy backend query attack.

Kumar in (Kumar, 2013) proposed a technique to detect and prevent crosssite scripting (XSS) and SQL injection attack. In this approach programmer created files which contain attack patterns of both XSS and SQL injection attack. HTTP request to database will be intercepted and compared to dynamic query with set of attack patterns in programmer define threshold. If the query is found to contain attack patterns defined by the programmer the query will be blocked and a report is generated. This technique was found to be effective after evaluation; however it cannot guarantee protection for attack patterns that were not included in the programmer predefined threshold.

Symbolic code execution was proposed by Huang and colleagues in (Huang et al., 2013) that used symbolic code execution to detect SQLIA and XSS vulnerabilities in a web application. Symbolic code execution refers to the randomly generating malicious queries and trying to simultaneously inject such queries into symbolic socket (HTTP POST, GET, cookies and forms). The purpose is to ensure that malicious request is reaching directly inside database engine without any sanitization; by doing this the database engine may perhaps respond with return error messages which indicate that the application is exploitable by SQL injection attack.

However the approach does not address the problem of inference attacks where application is vulnerable but configure not return to default configuration error messages.

Qu and colleagues proposed Java-based technique in (Qu et al., 2013) for detection of SQL injection vulnerability in Java-based web application. In this approach java codes are transformed into it intermediate representation by lexical and grammar analysis, this enable to collect sensitive point by matching the sensitive application entry point call while traversing the intermediate representation. Then perform taint dependency analysis for each fragile sensitive point based on the constructed data dependency graph and function call. Then, it generate the taint dependency graph from the fragile sensitive point to the pollution source and vulnerability is detected by measuring intersection between the value of fragile sensitive point and the attack mode. This method is language specific and accuracy of this approach depend logic design of language recognition, code conversion attack knowledge base that are used in vulnerability detection.

A techniques that combine both static and dynamic approach to detect SQL injection vulnerability called WAVES was proposed in (Huang et al., 2005) to overcome problem of damage or changing state of web application by penetration testing scanners. WAVES is capable of detecting entry points that are vulnerable to injection and XSS attacks without modifying or course any damage to web applications. However, experimental evaluation show that WAVES failed to detect all vulnerabilities detected by static code analyzer. According to the author the failure was result of limited functionalities of WAVES to observe HTML output.

3.3.1.2. Evaluation of Proposed Techniques and firewall

According to Antunes in (Antunes and Vieira, 2010) there is no systematic approach to evaluate vulnerability scanners this is because one scanner can be accurate in one language and failed to achieved any detection in other programming language. Thus, they proposed costume approach compare the effectiveness of static against dynamic vulnerability assessment tools based on coverage number of known vulnerabilities in target applications. The approach adopted in this study is known as F-measure proposed by (Van Rijsbergen 1979), which largely independent of the way vulnerabilities are counted. In fact, it represents the attribute mean of two very popular measures" that is precision and recall.

Experimental evaluation shows the average F measure produced by static scanner is 80% of the total vulnerabilities while dynamic scanner produced average of 36%. This evaluation shows there is much gap between the two different approaches, there is need to improve the effectiveness of these scanners of different approaches.

Experimental method was proposed in (Khoury et al., 2011) to challenge the capability of three different scanners. The purpose is to ensure whether these

scanners are able to perform automatic testing in effective way. The test bed applications (PCI, WackoPicko and MatchIt) used has the following feature;. PCI with three (3) stored SQL injection with two (2) login requirements, WackoPicko with one (1) stored SQLI one login requirement and MatchIt with one stored SQLI with no login required. Wireshark ware used to monitor activities between scanner and testing application. Experiment shows that all of the scanner successfully create username but failed to login using SQLIA. By analysing the captured traffic it was found that all of the scanner ware able to trigger default error configuration message from the server but failed to recognize that as vulnerability because that return error message from database server was not addressed in the predefined error messages library.

In (Antunes and Vieira, 2009a) method is proposed to test and evaluate effectiveness four popular commercial scanners from different vendors. Authors claim that most of the vendor/individual adjust the effectiveness of their product based on specific custom application that which cannot predict effectiveness of these scanners when other applications. Thus, authors tested four scanners (Acunetix, AppScan, WebInspect and updated version of one of the mentioned scanner) on 300 public website. When these scanners was evaluated using accuracy metric, the result shows that best scanner detect 40% out of total vulnerabilities with 18% of false positive. Worse among the scanners provide the coverage of 25% with 7% of false
positive.

Similar approach was proposed in (Antunes and Vieira, 2009b) but in this approach authors choose to developed custom application with number of vulnerability which enable to evaluate effectiveness of static against dynamic scanners. Authors choose eight (8) popular scanners four implementing static approach and four implementing dynamic approach. Each scanner was tested on same 8 websites four public and four private with total number of sixty one (61) known vulnerabilities. Result of evaluation shows that best among dynamic scanner achieved coverage of 50.8% of total vulnerabilities with 14% false positive. The worse among dynamic scanner achieved coverage of 9.8% out of total vulnerability with no false positive In case of static scanners the best scanner achieved 100% coverage of all vulnerabilities with 23.6% false negative and worse among static scanners achieved coverage of 39.3% of total vulnerability with 26.6% false positive.

Despite the fact that this evaluation cannot be generalized to compare the effectiveness of scanner with different implementation, however it shows significant difference different scanner implementing same approach reporting different vulnerabilities on same tested applications. Hence, there is need to improve the coverage detection of dynamic and static scanner as well as reduce the number of false positive in both side.

3.3.2. Commercial and open source of web application firewall

Unlike academic vulnerabilities scanners, open source tools such as Vega, Zap, Wa3p, Wapit and Nikitoetc are available for public use inform of source code application under copyright for free of charge. However, architecture, algorithm or development approach is not available to public. Individuals or researchers are permitted to study and improve open source tool with consent of the owner. Beside, Open source and academic Web vulnerability scanner, there are also commercial tools such as AppScan, Acunetix, Bugblast, Netsparker etc. These tools are totally different from academic and open source tools in the sense that users can only utilize the full functionalities of these tools by purchase, also architecture, algorithms or method used by development of these tools are not available to public and no vendor allows improvements of their tool (Djuric, 2013), (Acunetix, 2013), (OWSAP, 2013)

The advantage of commercial tool over other tools is that; they provide user with extensive help and functionalities that are not available in academic and commercial tools (OWSAP, 2013).

Acunetix Web Vulnerability assessment tool that is equipped with much functionality. Acunetix is capable of detecting much vulnerability in web application.

Acunetix uses technology called AcuSensor which allowed Acunetix to run in different mode and this technology are expected to enable fast vulnerabilities assessment and maintain law false positive. However, study shows that acunetix take 3hr to scanned page with 100 web pages. Acunetix company provide free software with is capable of detecting only XSS vulnerability and licensed software injection, file execution, session

management, and manual buffer overflow attacks (Acunetix, 2013).
Grendel-Scan is free scanner that is capable of detecting web applications vulnerabilities such as SQL injection, XSS and session management, but do not detect complicated vulnerabilities such design and logic defects (Grendel).
HP's WebInspect is the one of the most popular web application vulnerabilities assessment tool. HP claim WebInspect is capable of detecting vulnerabilities in a complex application that cannot be detected by traditional scanners. It uses technology that enable parallel crawling, multiple payload injections. WebInspect is capable of detecting session management, broken SQL injection and XSS vulnerabilities. Many literature review in this thesis evaluate the effectiveness of WebInspect (Inspect, 2012).
Falcove is web vulnerabilities assessment tool that is available for sell in Buy Server limited. Ltd Falcove is capable of identifying web vulnerabilities as well as exploiting them, it uses intelligent crawler that is capable of recognizing form with password fields, shopping cards and present report indicating the security state of tested application. The trial version of this software can detect detects SQL injection, XSS, and file execution attacks (Falcove, 2007).
N-Stalke is vulnerability scanning tool that provide over 39,000 infrastructure and signature check. User can set the custom assessment policies depending on the need of user requirements. N-Stalker provide 7 day trial version which enable user to check for SQL injection, XSS attacks, buffer overflows, and session management attacks (N-Stalker, 27 Feb. 2014,).
Rational AppScan is licensed software tool that powered by IBM, it provide unlimited edition to the user. However, AppScan can only perform security assessment on the application provided by IBM Company. It is capable of detecting SQL injection, XSS attacks, buffer overflows, and other popular web application vulnerabilities. AppScan have been evaluated by many literature review in this thesis (IBM, 2013).

3.4. Research Findings

This research has used analytical approach to evaluate current methods and approaches used by web applications as firewall to detect and prevent SQL injection attacks. We did not perform any experiments, it is based on our experience and therefore our findings is as follows:
Our study reveals that blacklist approach in developing a web firewall has been traditionally deployed as a key element typically in the form of a malicious database of known digital signatures, heuristics or behavior characteristics associated with SQL injection attacks that have been identified in the wild.

Since blacklist relay on only known SQL injection attacks signatures and experienced SQL injection attack vectors, exploits, vulnerabilities, and for which counter-measures are already known or developed. Therefore it is limited against unwanted attacks unknown menaces like zero-day threats (which have yet to be discovered and isolated by security professionals), blacklisting is of very limited.
However the advantage of blacklist it's traditionally been a low-maintenance option, as responsibility for compiling and updating a blacklist of applications or entities falls to the software itself and its related databases, or to some form of third-party threat intelligence/service provider.
On other hand whitelisting turns the blacklist logic on its head: You draw up a list of acceptable entities (software applications, email addresses, users, processes, devices, etc.) that are allowed access to a system or network, and block everything else. It's based on a "zero trust" principle which essentially denies all, and allows only what's necessary.

The simplest whitelisting techniques used for systems and networks identify applications based on their file name, size, and directory paths. But the U.S. National Institute of Standards and Technology or NIST, a division of the Commerce Department, recommends a stricter approach, with a combination of cryptographic hash techniques and digital signatures linked to the manufacturer or developer of each component or piece of software.
At the network level, compiling a whitelist begins by constructing a detailed view of all the tasks that users need to perform, and the applications or processes they need, to perform them. The whitelist might include network infrastructure, sites and locations, all valid applications, authorized users, trusted partners, contractors, services, and ports. Finer-grained details

may drill down to the level of application dependencies and software libraries (DLLs, etc.), plugins, extensions, and configuration files.

Whitelisting for user-level applications could include email (filtering for spam and unapproved contacts), programs and files, and approved commercial or non-commercial organizations registered with Internet Service Providers (ISPs).

In all cases, whitelists must be kept up to date, and administrators must give consideration both to user activity (e.g., what applications they're allowed to install or run) and user privileges (i.e., making sure that users aren't granted inappropriate combinations of access rights).

Third-party whitelisting services exist and are sometimes employed by enterprises seeking to ease the management burden that's associated with the process. These services are often reputation-based, using technology to give ratings to software and network processes based on their age, digital signatures, and rate of occurrence.

4. CONCLUSION

This study presented background of web application threat and firewall, explored different methods and approaches for detection and prevention of SQL injection attacks as web firewall. Our study identifies half of the study uses blacklist approach while there no much studies proposing whiletlisting approach with only three studies proposed combination of the two approaches. Furthermore the study reveals that fact that, blacklists are restricted to known variables (documented malware, etc.), and that attacks variants are continually being designed to evade behavior or signature-based modes of detection, there's a feeling in many circles that whitelisting represents the more sensible approach to information security.

This is despite the time, effort, and resources which must be spent in compiling, monitoring, and updating whitelists at enterprise level – and the need to guard against efforts by cybercriminals to compromise existing whitelisted applications (which would still have the go-ahead to run) or to design applications or network entities that have identical file names and sizes to approved ones.

As always such debate, there are also those who favor a best of both worlds scenario, with a blacklisted approach to security software for malware and intrusion detection and eradication, operating in tandem with a whitelisted policy governing access to the system or network as a whole.

REFERENCES

[1] Acunetix. (2013). Accunetix Vulnerability Scanner. Retrieved 29/06/2015, from https://www.acunetix.com/vulnerability-scanner/

[2] Agosta, Giovanni, Barenghi, Alessandro, Parata, Antonio, & Pelosi, Gerardo. (2012). Automated security analysis of dynamic web applications through symbolic code execution. Paper presented at the Information Technology: New Generations (ITNG), 2012 Ninth International Conference on.

[3] Aliero, M. S., Ghani, I., Zainudden, S., Khan, M. M. & Bello, M. 2015. review on sql injection protection methods and tools. Jurnal Teknologi, 77.

[4] Aliero MS, Ardo AA, Ghani I, Atiku M. "Classification of Sql Injection Detection And Prevention Measure". IOSR Journal of Engineering (IOSRJEN), ISSN (e). 2016:2250-3021

[5] Aliero MS, Ghani I. "A component based SQL injection vulnerability detection tool". InSoftware Engineering Conference (MySEC), 2015 9th Malaysian 2015 Dec 16 (pp. 224-229). IEEE.

[6] Antunes, Nuno, & Vieira, Marco. (2009a). Comparing the effectiveness of penetration testing and static code analysis on the detection of sql injection vulnerabilities in web services. Paper presented at the Dependable Computing, 2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on.

[7] Antunes, Nuno, & Vieira, Marco. (2009b). Detecting SQL injection vulnerabilities in web services. Paper presented at the Dependable Computing, 2009. LADC'09. Fourth Latin-American Symposium on.

[8] Antunes, Nuno, & Vieira, Marco. (2010). Benchmarking vulnerability detection tools for web services. Paper presented at the Web Services (ICWS), 2010 IEEE International Conference on.

[9] Antunes, Nuno, & Vieira, Marco. (2011). Enhancing penetration testing with attack signatures and interface monitoring for the detection of injection vulnerabilities in web services. Paper presented at the Services Computing (SCC), 2011 IEEE International Conference on.

[10] Antunes, Nuno, & Vieira, Marco. (2012). Evaluating and improving penetration testing in web services. Paper presented at the Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on.

[11] Antunes, Nuno, & Vieira, Marco. (2015). Assessing and Comparing Vulnerability Detection Tools for Web Services: Benchmarking Approach and Examples. Services Computing, IEEE Transactions on, 8(2), 269-283.

[12] Appelt, Dennis, Nguyen, Cu Duy, Briand, Lionel C, & Alshahwan, Nadia. (2014). Automated testing for SQL injection vulnerabilities: an input mutation approach. Paper presented at the Proceedings of the 2014 International Symposium on Software Testing and Analysis.

[13] Bandhakavi, Sruthi, Bisht, Prithvi, Madhusudan, P, & Venkatakrishnan, VN. (2007). CANDID: preventing sql injection attacks using dynamic candidate evaluations. Paper presented at the Proceedings of the 14th ACM conference on Computer and communications security.

[14] Bau, Jason, Bursztein, Elie, Gupta, Divij, & Mitchell, John. (2010). State of the art: Automated black-box web application vulnerability testing. Paper presented at the Security and Privacy (SP), 2010 IEEE Symposium on.

[15] Boyd, Stephen W, & Keromytis, Angelos D. (2004). SQLrand: Preventing SQL injection attacks. Paper presented at the Applied Cryptography and Network Security.

[16] Buehrer, Gregory, Weide, Bruce W, & Sivilotti, Paolo AG. (2005). Using parse tree validation to prevent SQL injection attacks. Paper presented at the Proceedings of the 5th international workshop on Software engineering and middleware.

[17] Cenzic's. (2014). Application Vulnerability Trends Report: 2014. Retrieved 29/09/2015, from https://www.info-point-security.com/sites/default/files/cenzic-vulnerability-report-2014.pdf

[18] Chen, Jan-Min, & Wu, Chia-Lun. (2010). An automated vulnerability scanner for injection attack based on injection point. Paper presented at the Computer Symposium (ICS), 2010 International.

[19] Cheon, Eun Hong, Huang, Zhongyue, & Lee, Yon Sik. (2013). Preventing SQL Injection Attack Based on Machine Learning. International Journal of Advancements in Computing Technology, 5(9).

[20] Cho, Ying-Chiang, & Pan, Jen-Yi. (2015). Design and Implementation of Website Information Disclosure Assessment System. PloS one, 10(3), e0117180.

[21] Ciampa, Angelo, Visaggio, Corrado Aaron, & Di Penta, Massimiliano. (2010). A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. Paper presented at the Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems.

[22] Cook, William R, & Rai, Siddhartha. (2005). Safe query objects: statically typed objects as remotely executable queries. Paper presented at the Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on.

[23] Djuric, Zoran. (2013). A black-box testing tool for detecting SQL injection vulnerabilities. Paper presented at the Informatics and Applications (ICIA), 2013 Second International Conference on.

[24] Falcove. (2007). Falcove Web Vulnerability Scanner and Penetration Testing. Retrieved 29/06/2015, from http://www.ramsayfalcove.com/htdocs/Welcome.html

[25] Fu, Xiang, Lu, Xin, Peltsverger, Boris, Chen, Shijun, Qian, Kai, & Tao, Lixin. (2007). A static analysis framework for detecting SQL injection vulnerabilities. Paper presented at the Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International.

[26] Gartner. (June 19, 2014). WEB APPLICATION ATTACK REPORT #5. Retrieved 29/06/2015,

from http://www.imperva.com/docs/hii_web_application_attack_report_ed5.pdf

[27] Grendel.). Grendel-Del Web vulnerability Scanner. Retrieved 29/06/2015, from http://sectools.org/tool/grendel-scan/

[28] Halfond, William GJ, & Orso, Alessandro. (2007). Detection and prevention of sql injection attacks Malware Detection (pp. 85-109): Springer.

[29] Huang, Shih-Kun, Lu, Han-Lin, Leong, Wai-Meng, & Liu, Huan. (2013). Craxweb: Automatic web application testing and attack generation. Paper presented at the Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on.

[30] Huang, Yao-Wen, Tsai, Chung-Hung, Lin, Tsung-Po, Huang, Shih-Kun, Lee, DT, & Kuo, Sy-Yen. (2005). A testing framework for Web application security assessment. Computer Networks, 48(5), 739-761.

[31] Huang, Yao-Wen, Yu, Fang, Hang, Christian, Tsai, Chung-Hung, Lee, Der-Tsai, & Kuo, Sy-Yen. (2004). Securing web application code by static analysis and runtime protection. Paper presented at the Proceedings of the 13th international conference on World Wide Web.

[32] IBM. (2013). IBM Web Application Scanner. Retrieved 29/06/2015, from http://www-03.ibm.com/software/products/en/appscan

[33] Inspect, HP. (2012). HP Inspect Vulnerability. Retrieved 29/06/2015, from http://sectools.org/tool/webinspect/

[34] Jnena, Rami MF. (2013). Modern Approach for WEB Applications Vulnerability Analysis. The Islamic University of Gaza.

[35] Joshi, Anamika, & Geetha, V. (2014). SQL Injection detection using machine learning. Paper presented at the Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on.

[36] Kals, Stefan, Kirda, Engin, Kruegel, Christopher, & Jovanovic, Nenad. (2006). Secubat: a web vulnerability scanner. Paper presented at the Proceedings of the 15th international conference on World Wide Web.

[37] Khoury, Nidal, Zavarsky, Pavol, Lindskog, Dale, & Ruhl, Ron. (2011). An analysis of black-box web application security scanners against stored SQL injection. Paper presented at the Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on.

[38] Kumar, Praveen. (2013). The multi-tier architecture for developing secure website with detection and prevention of sql-injection attacks. International Journal of Computer Applications, 62(9), 30-35.

[39] Lawal, MA, Sultan, Abu Bakar Md, & Shakiru, Ayanloye O. (2016). Systematic Literature Review on SQL Injection Attack. International Journal of Soft Computing, 11(1), 26-35.

[40] Liban, Abdilahi, & Hilles, Shadi. (2014). Enhancing Mysql Injector vulnerability checker tool (Mysql Injector) using inference binary search algorithm for blind timing-based attack. Paper presented at the Control and System Graduate Research Colloquium (ICSGRC), 2014 IEEE 5th.

[41] Liu, Anyi, Yuan, Yi, Wijesekera, Duminda, & Stavrou, Angelos. (2009). SQLProb: a proxy-based architecture towards preventing SQL injection attacks. Paper presented at the Proceedings of the 2009 ACM symposium on Applied Computing.

[42] Livshits, V Benjamin, & Lam, Monica S. (2005). Finding Security Vulnerabilities in Java Applications with Static Analysis. Paper presented at the Usenix Security.

[43] McClure, Russell A, & Kruger, Ingolf H. (2005). SQL DOM: compile time checking of dynamic SQL statements. Paper presented at the Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on.

[44] Medhane, Munqath H Alattar SP. R-WASP: Real Time-Web Application SQL Injection Detector and Preventer.

[45] N-Stalker. (27 Feb. 2014,). N-Stalker Web Vulerability Scanner. Retrieved 29/06/2015, from http://www.windowsecurity.com/software/Web-Application-Security/N-Stalker-Web-Application-Security-Scanner.html

[46] Nguyen-Tuong, Anh, Guarnieri, Salvatore, Greene, Doug, Shirley, Jeff, & Evans, David. (2005). Automatically hardening web applications using precise tainting: Springer.

[47] OWSAP. (2013). Top 10 Vulnerability 2013 by Open Web Security Project. Retrieved 29/06/2015, from https://www.owasp.org/index.php/Top_10_2013-Top_10

[48] Qu, Binbin, Liang, Beihai, Jiang, Sheng, & Chutian, Ye. (2013). Design of automatic vulnerability detection system for Web application program. Paper presented at the Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on.

[49] Scott, David, & Sharp, Richard. (2002). Abstracting application-level web security. Paper presented at the Proceedings of the 11th international conference on World Wide Web.

[50] Shahriar, Hossain, & Zulkernine, Mohammad. (2012). Information-theoretic detection of sql injection attacks. Paper presented at the High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on.

[51] Shar, Lwin Khin, & Tan, Hee Beng Kuan. (2012). Predicting common web application vulnerabilities from input validation and sanitization code patterns. Paper presented at the Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on.

[52] Shin, Yonghee, Williams, Laurie, & Xie, Tao. (2006). Sqlunitgen: Sql injection testing using static and dynamic analysis. Paper presented at the 17th IEEE International Conference on Software Reliability Engineering, ISSRE.

[53] Singh, Avinash Kumar, & Roy, Sangita. (2012). A network based vulnerability scanner for detecting SQLI attacks in web applications. Paper presented at the Recent Advances in Information Technology (RAIT), 2012 1st International Conference on.

[54] Thiyagarajan, A, Uma, S, Vipin, Ambat, & Dheen, Najeem. (2015). METHODS FOR DETECTION AND PREVENTION OF SQL ATTACKS IN ANALYSIS OF WEB FIELD DATA.

[55] Tiwari, Yash, & Tiwari, Mallika. (2015). A Study of SQL of Injections Techniques and their Prevention Methods. International Journal of Computer Applications, 114(17).

[56] Valeur, Fredrik, Mutz, Darren, & Vigna, Giovanni. (2005). A learning-based approach to the detection of SQL attacks Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 123-140): Springer.

[57] Zhang, Xin-hua, & Wang, Zhi-jian. (2010). Notice of Retraction A Static Analysis Tool for Detecting Web Application Injection Vulnerabilities for ASP Program. Paper presented at the e-Business and Information System Security (EBISS), 2010 2nd International Conference on.