Securing Web Applications against Structured Query Language Injection Attacks using a Hybrid Approach: Input Filtering and Web Application Firewall

Francis Kyalo Muia
Jomo Kenyatta University of Agriculture and Technology, Juja Main Campus
Institute of Computer science and information Technology

Calvins Otieno, PhD
Jomo Kenyatta University of Agriculture and Technology, Eldoret Campus
Institute of Computer science and information Technology

Dennis Njagi, PhD
Jomo Kenyatta University of Agriculture and Technology, Karen Campus
Institute of Computer Science and information Technology

ABSTRACT
SQL injection is a type of attack used to gain, manipulate, or delete information in any data-driven system regardless of whether the system is online or offline and whether this system is a web or non-web based. A common approach for an attacker to launch SQLIA is by modifying the user input to contain partial SQL queries and trick the server into executing them. In this paper, a literature review of the SQL injection attacks and their mitigation is presented. It shows that the study of SQL injection in general has been conducted in diverse range of areas. The main objective of this paper is to give an elaborate study on different types of SQL injection, their mitigation strategies, critiques of past approaches and finally the knowledge gap. It seeks to create knowledge on work done by others in the area of SQL injection attacks in web applications which remains a threat up-to-date despite the numerous studies done on the same field.

Keywords
Structured Query Language, Structured Query Language Injection Attacks, Web Application Keywords.

1. INTRODUCTION
Over the last two decades there is tremendous growth of websites. From government departments to different types of organizations, agencies, banks and even small to medium enterprises rely on web based applications for smooth running of their processes and transactions. Websites have become the important information release centers that manage large amount of data for sharing among billions of users over the Internet.

During the last few years most organizations preferred to have web based applications to have a global market access which in turn means more people accessing the web applications and hence increased vulnerability scope. Despite these web based applications having many advantages, there is also numerous risks associated with them. They have to face many input vulnerabilities including the SQL injection attacks (SQLIA). SQLIA have become more popular among intruders due to improvements in its techniques over the years. Over the last few years SQLIA attacks have emerged as one of the serious threats to the web based data driven applications. In fact, the open Web Application Security Project has placed SQLIA in top ten vulnerabilities for a web based application [13].

2. LITERATURE REVIEW
2.1 SQL Injection Concepts and Definition
When web server receives web user's page request from web browser, it interacts with application server. Application server relays the page request to either a file system or database where data is stored. The result of this interaction is to create a dynamic web page that displays relative information that is retrieved from the database or file system in a web page, as shown in Figure 1 [1]. Most of relational database management systems adopt Structured Query Language (SQL) as their programming language [3]. The computer security firm Imperva calls it the "most pernicious vulnerability in human computer history" and says that between 2005 and 2011, SQL attacks accounted for 83 percent of data breaches during that period [23].

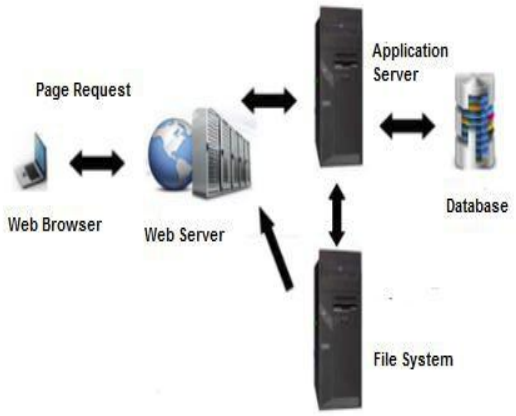

Figure 1: Typical Web Application Architecture (Buehrer, Weide, & Sivilotti, 2015)

Web application is the software program installed in web server of a website. Web application usually has tree-tiers.

1) Presentation Tier: this is where the web browser captures user input and displays the processed data using HTML, JavaScript, Flash, etc. through Graphical User Interface (GUI).

2) Common Gateway Interface (CGI) Tier: This tier lies between presentation tier and database tier as the Server Script Process (SSP) that encapsulates the business logic to support web application. User's data is processed and stored into the database. Retrieved data is presented in presentation tier through CGI tier from database according to web users' requests. CGI tier processes web application data with PHP, ASP, JSP, etc. and server script programming languages.

3) Database Tier: it is used to store data and also responsible to authenticate access and provides data storage services. [4].

2.1.1 Definition of SQL Injection Attack (SQLIA):
SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application.

The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. It is in fact an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

SQL queries that hackers deliberately craft may be interpreted as user input if SQL query keywords are not filtered out properly. Most of IDSs focus on monitoring IP and Network layer of Internet protocol and do not effectively detect SQLIA. Besides, SQLIA is difficult to detect and prevent as it has many types, approaches and various evading SQLIA detection and prevention techniques [11]. Victims of SQLIA sometimes are not even aware of their information leakage until the time after SQLIA has been successfully executed.

SQLIA is effective for all databases adopting SQL language as programming language, e.g. MySQL, MS SQL Server, DB2, Oracle, Sybase, etc. In the worst case, SQLIA also can lead to the operating system of website being hijacked. The symptoms of such attack may simultaneously affect multiple portions of the system or some portion of the system at different times [9].

SQLIA normally has three attack phases:
Reconnaissance phase: it reconnoiters that there is a vulnerability in web application via iteratively attempting to inject malicious input to a web application and carefully observe the web application response. Besides, hackers may utilize the diversity of databases to detect the database schema information. Malicious SQLIA queries are launched into the target web application to attack the Database Management System (DMS) if any vulnerability is found.

Hackers will attempt to attack the operating system of the web application after they have compromised the back-end database.

2.2 Types of SQLIA

In this section, different kinds of SQLIAs which are known to date are presented and discussed. For each attack type, a descriptive name is provided, one or more attack intents, a description of the attack, an attack example, and a set of references to publications and Websites that discuss the attack technique and its variations in greater detail. The different types of attacks are generally not performed in isolation; many of them are used together or sequentially, depending on the specific goals of the attacker.

2.2.1 Tautologies

In logic, a tautology is a formula which is true in every possible interpretation. In a tautology-based attack, the code is injected using the conditional OR operator such that the query always evaluates to TRUE [14]. Tautology-based SQL injection attacks usually bypass user authentication and extract data by inserting a tautology in the WHERE clause of a SQL query. The query transforms the original condition into a tautology, causes all the rows in the database table to be open to an unauthorized user.

2.2.2 Illegal/Logically Incorrect Queries

This type of SQLIA is one of the manipulation categories of attacks [12]. It is the preliminary step to gather important information of the back-end database server type and structure. Hackers deliberately submit illegitimate SQL queries, i.e. logical incorrect in order to let the database server to reject the queries and display error feedback message, e.g. database server type, table and column name or syntax or logical or type mismatches errors, etc. that aim to debug very helpful information if the database has not been designed to anti-SQLIA prevention [26]. E.g. if a hacker inserts a single quotation in end of URL, the website returns error message revealing some sever or database information. The attacker is then definitely sure that the web application is vulnerable, and they can use other types of SQLIA technologies to exploit the back-end database and extract data from the back-end database.

2.2.3 Union Query

This type of SQLIA lies under manipulation and code Injection category. It is usually used for bypassing authentication and unauthorized retrieval of confidential information from back-end database. By inserting SQL keyword "Union" and another SQL query that is proposed to authorize and retrieve confidential data into one legal SQL query so that the inserted SQLIA query bypasses the authentication to retrieve both tables' data. e.g. Original SQL queries:
Query = "SELECT * FROM employee;"
Malicious insert another query concatenated by "union" SQL keyword:
Modified Query = "SELECT * FROM employee union SELECT * FROM salary;" [14]

2.2.4 Piggy-Backed Queries

In this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that "piggy-back" on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first [26]. This type of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including stored procedures, into the additional queries and have them executed along with the original query. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.
Example: If the attacker inputs ["; drop table users - -] into the password field, the application generates the query:

SELECT accounts FROM users WHERE login='doe' AND pass=''; drop table users -- '

2.2.5 Stored Procedures

Stored procedure is a series of multiple executing commands procedures. This type of SQLIA is a function call injection category and can be deliberately crafted to execute malicious codes so as to attack the operating system [17]. Furthermore, stored procedure may create other type of vulnerabilities that hackers may arbitrarily upload malicious codes to the server or escalate their privileges. Stored procedures are set by database programmers as an extra abstraction layer, meanwhile it becomes as vulnerability of web application for SQLIA [18].

2.2.6 Inference

In this attack, the query is modified to recast in the form of an action that is executed based on the answer to a true/false question about data values in the database. In this type of

injection, attackers are generally trying to attack a site that has been secured enough so that, when an injection has succeeded, there is no usable feedback via database error messages. Since database error messages are unavailable to provide the attacker with feedback, attackers must use a different method of obtaining a response from the database. In this situation, the attacker injects commands into the site and then observes how the function/response of the website changes. By carefully noting when the site behaves the same and when its behavior changes, the attacker can deduce not only whether certain parameters are vulnerable, but also additional information about the values in the database. There are two well-known attack techniques that are based on inference. They allow an attacker to extract data from a database and detect vulnerable parameters [14].

2.2.7 Blind SQL injection

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker. The page with the vulnerability may not be the one that displays data but will display differently depending on the results of a logical statement injected into the legitimate SQL statement called for that page. This type of attack has traditionally been considered time-intensive because a new statement needed to be crafted for each bit recovered, and depending on its structure, the attack may consist of many unsuccessful requests. Recent advancements have allowed each request to recover multiple bits, with no unsuccessful requests, allowing for more consistent and efficient extraction [14]. There are several tools that can automate these attacks once the location of the vulnerability and the target information has been established.

2.2.8 Second order SQL injection

Second order SQL injection occurs when submitted values contain malicious commands that are stored rather than executed immediately. In some cases, the application may correctly encode an SQL statement and store it as valid SQL. Then, another part of that application without controls to protect against SQL injection might execute that stored SQL statement. This attack requires more knowledge of how submitted values are later used. Automated web application security scanners would not easily detect this type of SQL injection and may need to be manually instructed where to check for evidence that it is being attempted.

2.3 SQL Injection Counter Measures.

After having successfully detected a vulnerability or any kind of attack that exploits the vulnerability, other schemes could be applied to cure the system. In usual case, there are mainly two types of schemes; some are for prevention and others are for curing the system once it is under attack. In case of SQL Injection, those schemes which work for preventing SQL injection also do the curing of the system (or application) in early stage. Hence, in plain term, we could call the schemes 'countermeasures'.

Below are some of the countermeasures.

2.3.1 Amnesia

AMNESIA is a model-based technique that combines static analysis and run-time monitoring [6]. In its static phase, AMNESIA uses static analysis to build models of the different types of queries an application can legally generate at each point of access to the database. In its dynamic phase, AMNESIA intercepts all queries before they are sent to the database and checks each query against the statically built models. Queries that violate the model are identified as SQLIAs and prevented from executing on the database. In their evaluation, the authors have shown that this technique performs well against SQLIAs.

2.3.2 SQLrand Scheme

SQLrand provides a framework that allows developers to create SQL queries using randomized keywords instead of the normal SQL keywords. A proxy between the web application and the database intercepts SQL queries and de-randomizes the keywords. The SQL keywords injected by an attacker would not have been constructed by the randomized keywords, and thus the injected commands would result in a syntactically incorrect query. Since SQLrand uses a secret key to modify keywords, its security relies on attackers not being able to discover this key. SQLrand requires the application developer to rewrite code [27].

2.3.3 SQL DOM Scheme

SQL DOM (a set of classes that are strongly-typed to a database schema) framework [7]. They closely consider the existing flaws while accessing relational databases from the OOP Language's point of view. They mainly focus on identifying the obstacles in the interaction with the database via CLIs. SQL DOM object model is the proposed solution to tackle these issues through building a secure environment (i.e., creation of SQL statement through object manipulation) for Communication. The qualitative evaluation of this approach has shown many advantages and benefits in terms of: error detection during compile time, reliability, testability, and maintainability.

2.3.4 SQLIA Prevention Using Stored Procedures

Stored procedures are subroutines in the database which the applications can make a call to [15]. The prevention in these stored procedures is implemented by a combination of static analysis and runtime analysis. The static analysis used for commands identification is achieved through stored procedure parser and the runtime analysis by using a SQLChecker for input identification. Webs SARI (Web application Security by Static Analysis and Runtime Inspection) was used and implemented on 230 open source applications on SourceForge.net. The approach was effective, however it failed to remove the SQLIVs (SQL Injection Vulnerabilities). It was only able to list the input either white or black.

2.3.5 Parse Tree Validation Approach

[6] adopted the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement. They stopped the execution of Statement unless there is a match. This method was tested on a student Web application using SQLGuard. Although this approach is efficient, it has two major drawbacks: additional overheard computation and listing of input only (black or white).

2.3.6 Dynamic Candidate Evaluations Approach

[13] Propose CANDID (Candidate evaluation for Discovering Intent Dynamically). It is a Dynamic Candidate Evaluations method for automatic prevention of SQL Injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer). Hence, it solves the issue of manually modifying the application to create the prepared statements. Though this tool is shown to be efficient for some cases, it fails in many other cases. For example, it is inefficient when dealing with external functions and when applied at a wrong level. Besides that, sometimes it also fails

due to the limited capability of the scheme.

2.3.7 Ali et al.'s Scheme

[2] Adopt the hash value approach to further improve the user authentication mechanism. They use the user name and password hash values. SQLIPA (SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The username and password hash values are created and calculated at runtime for the first time the particular user account is created. Hash values are stored in the user account table. Though the proposed framework was tested on few sample data and had an overhead of 1.3 mms, it requires further improvement to reduce the overhead time. It also requires to be tested with larger amount of data.

2.3.8 SQLCHECKER Approach

It checks whether the input queries conform to the expected ones defined by the programmer. [2] A secret key is applied for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages. It is a very efficient approach; however, once an attacker discovers the key, it becomes vulnerable. Furthermore, it also needs to be tested with online Web applications.

2.3.9 Detecting Intrusions in Web Databases (DIWeDa) Approach

[24] propose IDS (Intrusion Detection Systems) for the backend databases. They use DIWeDa, a prototype which acts at the session level rather than the SQL statement or transaction stage, to detect the intrusions in Web applications. DIWeDa profiles the normal behavior of different roles in terms of the set of SQL queries issued in a session, and then compares a session with the profile to identify intrusions. The proposed framework is efficient and could identify SQL injections and business logic violations too. However, with a threshold of 0.07, the True Positive Rate (TPR) was found to be 92.5% and the False Positive Rate (FPR) was 5%. Hence, there is a great need of accuracy improvement (Increase of TPR and decrease of FPR). It also needs to be tested against new types of Web attacks [3].

2.3.10 Manual Approaches

[6] highlights the use of manual approaches in order to prevent SQLI input manipulation flaws. In manual approaches, defensive programming and code review are applied. In defensive programming: an input filter is implemented to disallow users to input malicious keywords or characters. This is achieved by using white lists or black lists. As regards to the code review, it is a low cost mechanism in detecting bugs; however, it requires deep knowledge on SQLIAs.

2.3.11 Automated Approaches

Besides using manual approaches, also highlights the use of automated approaches. [2] The author notes that the two main schemes are: Static analysis Find Bugs and Web vulnerability scanning. Static analysis Find Bugs approach detects bugs on SQLIAs, gives warning when an SQL query is made of variable. However, for the Web vulnerability scanning, it uses software agents to crawl, scans Web applications, and detects the vulnerabilities by observing their behavior to the attacks.

2.3.12 Parameterized statements

With most development platforms, parameterized statements that work with parameters can be used (sometimes called placeholders or bind variables) instead of embedding user input in the statement [].A placeholder can only store a value of the given type and not an arbitrary SQL fragment. Hence the SQL injection would simply be treated as a strange (and probably invalid) parameter value. In many cases, the SQL statement is fixed, and each parameter is a scalar, not a table. The user input is then assigned (bound) to a parameter. Unfortunately, prepared statements can also be vulnerable to SQLIAs unless developers rigorously apply defensive coding guidelines.

2.3.13 Escaping

A straightforward, though error-prone way to prevent injections is to escape characters that have a special meaning in SQL. The manual for an SQL DBMS explains which characters have a special meaning, which allows creating a comprehensive blacklist of characters that need translation. For instance, every occurrence of a single quote (') in a parameter must be replaced by two single quotes (") to form a valid SQL string literal.

For example, in PHP it is usual to escape parameters using the function mySQLi_real_escape_string(); before sending the SQL query:

```
$mySQLi = new mySQLi('hostname', 'db_username', 'db_password', 'db_name');
$query = sprintf("SELECT * FROM `Users` WHERE UserName='%s' AND Password='%s'",
 $mySQLi->real_escape_string($username),

 $mySQLi->real_escape_string($password));

$mySQLi->query($query); [14]
```

2.3.14 Pattern check

Integer, float or Boolean, string parameters can be checked if their value is valid representation for the given type. Strings that must follow some strict pattern (date, UUID, alphanumeric only, etc.) can be checked if they match this pattern.

2.3.15 SecuBat: A Web Vulnerability Scanner

Author developed a scanner named as "SecuBat" that use white box testing for identification of possible vulnerabilities. This technique relies on three components named crawling, attack and analysis components. This technique is implemented in C and MS SQL server database. [16]. As the popularity of the web increases and web applications become tools of everyday use, the role of web security has been gaining importance as well. The last years have shown a significant increase in the number of web-based attacks. For example, there has been extensive press coverage of recent security incidences involving the loss of sensitive credit card information belonging to millions of customers. Typical web application security vulnerabilities result from generic input validation problems. Examples of such vulnerabilities are SQL injection and Cross-Site Scripting (XSS).

2.3.16 Automatic Revised Tool for Anti-Malicious Injection

Writer believes that input validations are the main source of vulnerabilities for SQL and XSS attacks. This technique first checks for threats input areas in the HMTL form, cookies etc. This technique then generates automatic validation technique.

This technique based on four components named spider, analyzer, function producer and tester [17].

2.3.17 Eliminating SQL Injection Attacks - A Transparent Defense Mechanism

This technique uses validation and run time checks to safeguard the application against different types of attacks. This technique has advantage that it can be merged with existing application and also do not require any modification in source code. This technique relies on string analysis for static analysis and builds the SQL Graph. At the run time checks the input is validated against the SQL graph built at the static analysis phase. [18].

2.3.18 Defending Against Injection Attacks through Context-Sensitive String Evaluation

The technique named as Context-Sensitive String Evaluation (CSSE) use metadata information and context sensitive string evaluation function. This technique also does not require source code modification and programmer attention. This technique is implemented in PHP and use context sensitive. [19]. CSSE works by addressing the root cause why such attacks can succeed, namely the ad-hoc serialization of user-provided input. It provides a platform-enforced separation of channels, using a combination of assignment of metadata to user-provided input, metadata-preserving string operations and context-sensitive string evaluation. CSSE requires neither application developer interaction nor application source code modifications. Since only changes to the underlying platform are needed, it effectively shifts the burden of implementing countermeasures against injection attacks from the many application developers to the small team of security-savvy platform developers.

2.3.19 D-WAV: A Web Application Vulnerabilities Detection Tool using Characteristics of Web Forms

This method is an automated testing methodology which detect web vulnerabilities, for example, SQLIA and XSS. It gets a target web structure with the assistance of given URL. It makes test suites which consider the confidence of each one test with evaluation. At last, these test suites are executed and compared in order to make conclusion for HTML code investigations. A Web Application Vulnerabilities Detection Knowledge Repository is utilized to figure out if the vulnerabilities exist or not. This technique implemented into D-WAV. [20].

2.3.20 X-LOG Authentication Technique to Prevent SQL Injection Attacks

In these technique three filtrations schemas are used named as vulnerability guards, X Log authentication and stored procedures. This technique has been used against many types of attackers and it has proved to be an excellent one. [21]. This technique monitors the dynamically generated queries with the Data model which is generated by X- Log Generator at runtime and checks them for compliance. If the Data Comparison violates the model, then it represents potential SQLIA' s and its prevented from executing on the database and then reported.

2.3.21 Swaddler: An Approach for the Anomaly based Detection of State Violations in Web Applications

This technique is also used to protect stored procedures against the SQLIA named "Saddler". This technique also uses static analysis and run time checks of validations. This technique parses the SQL query and compares the user input query with the original SQL query to identify any problem with the input query. [14]. Swaddler analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state. By doing this, Swaddler is able to identify attacks that attempt to bring an application in an inconsistent, anomalous state, such as violations of the intended workflow of a web application.

2.3.23 Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection

This is a novel technique that tries to identify the possible vulnerabilities for SQLIA at development and testing phase. This technique uses syntactic and semantics of the queries for possible detection of vulnerability for SQLIA attack. This technique identifies the point where a malicious user can exploit for SQLIA. This technique is implemented in a tool named "sania". [23]. Sania intercepts the SQL queries between a web application and a database, and automatically generates elaborate attacks according to the syntax and semantics of the potentially vulnerable spots in the SQL queries. In addition, Sania compares the parse trees of the intended SQL query and those resulting after an attack to assess the safety of these spots

2.3.24 SMask: Preventing Injection Attacks in Web Applications by Approximating Automatic Data/Code Separation

SMask is a technique that is used for detection of SQLIA and XSS. This technique uses string masking for syntactical analysis for differentiating the legal query and malicious one. This technique uses pre and post processor for query validation. [22]. By using string masking to persistently mark legitimate code in string values, SMask is able to identify code that was injected during the processing of an http request. SMask works transparently to the application and is implementable either by integration in the application server or by source-to-source translation using code instrumentation.

2.3.25 Automated Protection of PHP Applications against SQL-injection Attacks

This technique is based on static and dynamic analysis for identification of SQLIA in a PHP code. This technique also relies on code reengineering to protect legacy applications. This technique tested in phpBB (PHP Bulletin board) and produced amazing results. [24].

2.3.26 Using Automated Fix Generation to Secure SQL Statements

This technique automatically eliminates the SQLIA vulnerabilities from the java code. This technique uses the prepared statement and changes the vulnerable part of the query of the prepared statement. The Programmer can change the vulnerable part of the code with the automatic generated code. [5].

2.3.27 Web application Firewall

Since 2006, web application firewall research seems to have been growing steadily and will probably continue its growth. Internet security has been a rising trend and WAFs play a big part in mitigating cyber threats. According to Symantec's 2016 Security Report [32] crypto-ransomware was up 35% in 2015 from 2014, there were 36% more new malware variants and in 2015 there were nine breaches where more than 10 million identities were exposed. Zero-day vulnerabilities more

than doubled (+125%) between 2014 and 2015 with 54 new vulnerabilities. Akamai Technologies has similar figures on their Q1 of 2016 "akamai's [state of the internet] / security" report [29]. Comparing the first quarter of 2015 and 2016, DDoS attacks have risen 125% and mega attacks (greater than 100 Gbps) 137%.

2.3.28.1 Blacklist and whitelist algorithms

The majority of research work in WAFs is concentrated on creating and improving algorithms to identify and prevent cyber-attacks. They are further divided into two categories: improving attack signature patterns to be more comprehensive and/or efficient and self-learning algorithms. The largest research motivation in this category is to prevent SQL injections and quite justifiably so. Injections have been the number one threat according to OWASP Top 10 in both 2010 and 2013 surveys. The next survey will be published sometime in late 2016 or early 2017. A good example of new algorithms for detecting injection attacks can be found in "SQL Injection Attack Detection Method Using the Approximation Function of Zeta Distribution" [30]. This learning algorithm creates a zeta distribution profile for user submitted strings and tries to determine if it is harmful or not. For example, in normal data the symbol "SP", also known as "space", is the most used symbol but in injection strings the most used symbol is the asterisk, or "*". Some training data is required to create distinct categories for profiles.

2.3.28.2 Comparison of existing WAFs

There are very few studies that compare WAFs or measure their effectiveness. Among them is this study [31], which compares the three most popular open source WAFs: Comodo, ModSecurity by Trustwave SpiderLabs, AQTRONIX WebKnigh [31 and Guardian@JUMPERZ.NET [31]. It concluded that Comodo was the best one of them, generating less false negatives than WebKnight and blocking more attacks than Guardian. The study was a bit limited because all the WAFs were used with their default settings except for ModSecurity where the "Base" OWASP rules were installed. WebKnight seemed to block all POST-requests affecting the results significantly. In retrospect the study measured the default configurations of three different WAFs more than it measured their capabilities. How a WAF is tuned has a significant effect on its performance as the study "Estimates on the effectiveness of web application firewalls against targeted attacks" [32] discusses. There were four countermeasures that increased the effectiveness of a WAF based on expert knowledge. Those were WAF operator experience, the effort spent on tuning the WAF, automated black box testing tools, and whether an operator was monitoring the WAF.

2.3.28.3 New WAF Implementations

Several new WAF implementations have been suggested by the research community. The first one [29] is implemented as a plugin for WebScarab, which is a java-based web application testing tool from OWASP that intercepts and can alter HTTP requests. This WAF blocks control flow tampering attacks targeted at a web application. Control flow tampering means that a URL is requested in the wrong sequence which might result in an exploitation. To prohibit attackers from control flow tampering the WAF has to build a dependency graph that represents the relation of web pages and later on when someone tries to access a page from the wrong location the request is blocked. The difficulty of building an accurate dependency graph increases as websites become more dynamic in their content and when there are changes to the structure of the web site.

2.3.29 URL Validation/Filtering Approach

In most security solutions, traffic dissection process is the first operation before applying any security control. Typical HTTP request: HTTP protocol is expressed in a human-readable ASCII text. Headers use text to describe a request form a client (browser) or a response from the server. An HTTP request begins usually with a GET or POST method, followed by the URL and the protocol version. The following headers provide various information about the client, connection, content, etc. These headers are separated by \r\n to distinguish each header.

HTTP Request Dissection: The dissection module is able to recognize request's components (headers and the body which are separated \r\n characters). However, before making the dissection, it has to get information about security rules. Indeed, users are obliged to declare security rules for the body and for each header. With the knowledge of headers involved in the inspection process, the dissector will only extract and parse these headers [27].

2.4 Critiques of past approaches of preventing SQL injection attacks

Though many approaches and frameworks have been identified and implemented in many interactive Web applications, security still remains a major issue. SQL Injection prevails as one of the top-10 vulnerabilities and threat to online businesses targeting the backend databases. Research of most of the above discusses methods, their research has not been done exhaustively, testing has not been carried out exhaustively or they are not effective in prevention of SQL injection attacks. Hackers are in reality very innovative and as the time is passing by, new attacks are being launched that may need new ways of thinking about the solutions we currently have at our hands.

2.4.1 Overview of shortcomings of some of the existing approaches for mitigation of SQL injection attacks

Although defensive coding practices remain the best way to prevent SQL injection vulnerabilities, their application is problematic in practice. Defensive coding is prone to human error and is not as rigorously and completely applied as automated techniques Moreover, approaches based on defensive coding are weakened by the widespread promotion and acceptance of so-called "pseudo remedies" [29]. Intrusion Detection Systems (IDS's) which uses learning based techniques can provide no guarantees about their detection abilities because their success is dependent on the quality of the training set used. Amnesia-The primary limitation of this technique is that its success is dependent on the accuracy of its static analysis for building query models [30]. Proxy filters-This approach is human-based and, like defensive programming, requires developers to know not only which data needs to be filtered, but also what patterns and filters to apply to the data [31].

Taint based approach-The primary drawbacks of this technique are that it assumes that adequate preconditions for sensitive functions can be accurately expressed using their typing system and that having input passing through certain types of filters is sufficient to consider it not tainted. Instruction set randomization-While this technique can be very effective, it has several practical drawbacks: Firstly,

since it uses a secret key to modify instructions, security of the approach is dependent on attackers not being able to discover the key; Secondly, the approach imposes a significant infrastructure overhead because it require the integration of a proxy for the database in the system. New query development paradigm-By changing the development paradigm in which SQL queries are created, these techniques eliminate the coding practices that make most SQL Injection Attacks possible. Although effective, these techniques have the drawback that they require developers to learn and use a new programming paradigm or query-development process. Furthermore, because they focus on using a new development process, they do not provide any type of protection or improved security for existing legacy systems. White box texting-The primary drawbacks of this technique are the assumptions that preconditions for sensitive functions can be adequately and accurately expressed using their type system and forcing input to pass through certain types of filters is sufficient to consider it reliable. For many types of functions and applications, these assumptions do not hold

2.5. Knowledge Gap

After conducting the literature review it has been found that most of existing researchers are currently working on new ways to counter SQL injection attacks. Many of the suggested approaches are either not able to detect and prevent all SQL injection attacks or are resource intensive and therefore affects performance of the web databases. In an attempt to address these shortcomings, a hybrid approach that combines more than one approach is therefore adopted. The approach which is a combination of a web application firewall and a user input filter works by providing a two layered security check/mechanism for prevention of SQL injection. This technique therefore improves previous technique of URL filtering by reinforcing it using through the use of a web application firewall as a second security mechanism which therefore improves its effectiveness to a greater extend.

3. ACKNOLOWLEDGEMENT

I would like to acknowledge and thank my very supportive supervisors, Dr. Calvins Otieno and Dr. Dennis Njagi who have guided me in every step throughout the research period. I appreciate deeply everything you have done for me over this time of study and I will be forever grateful. Thanks for your guidance, advice, friendship, and enthusiasm all throughout the research duration.

4. REFERENCES

[1] [Angelo et al.] Angelo Ciampa, Corrado Aaron Visaggio, Massimiliano Di Penta:" A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications".

[2] "SQL Injection Tutorial," Oracle Corp., 2009. [Online]. Available:http://stcurriculum.oracle.com/tutorial/SQLInjection/index.htm. [Accessed: Mar. 11, 2010].

[3] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of sql injection attacks", in proceedings of the conference on detection of intrusions and Malware and vulnerability assessment (DIMVA), 2005.

[4] Lowe, D., and Henderson-Sellers, B. (2003b) Characterizing Web Systems: Merging Information and Functional Architectures. Architectural Issues of Web-Enabled Electronic Business. V. K. S. Murthy, N. Hershey, PA, USA, Idea Group Publishing.

[5] Using Automated Fix Generation to Secure SQL Statements. Stephen Thomas, Laurie Williams. IEEE Computer Society Washington, DC, USA: s.n., 2007. SESS '07 Proceedings of the Third International Workshop on Software Engineering for Secure Systems. p. 9.

[6] W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," in Proc. the International Symposium on Secure Software Engineering, 2006

[7] R.A. McClure, I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements", *Software Engineering 2005. ICSE 2005. Proceedings. 27th International Conference on*, pp. 88-96, 15–21 May 2005.

[8] Kasra Amirtahmasebi, Seyed Reza Jalalinia, and Saghar Khadem. A survey of SQL injection defense mechanisms. The 4th International Conference for Internet Technology and Secured Transactions (ICITST-2012), pages 1-8, Nov 2014.

[9] [Shanmughaneethi et al., 2009] Shanmughaneethi, S.V.; Shyni, S.C.E.; Swamynathan, S.; "SBSQLID: Securing Web Applications with Service Based SQL Injection Detection," Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT '09. International Conference on, vol., no., pp.702-704, 28-29 Dec. 2009

[10] Servlet and jsp filters. Online document, Sun Microsystems and Prentice Hall. http://www.moreservlets.com/.

[11] Junjin, M., An Approach for SQL Injection Vulnerability Detection. Proc. of the 6th International Conference on Information Technology: New Generations, Las Vegas, Nevada, April 2009, pp. 1411- 1414.

[12] A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications. Angelo Ciampa, Corrado Aaron Visaggio , Massimiliano Di Penta. New York : s.n., 2010. Proceeding SESS '10 Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems. pp. 43-49

[13] CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. Prithvi Bisht, P. Madhusudan , V. N.Venkatakrishnan. 2, February 2010, ACM Transactions on Information and System Security (TISSEC), Vol. 13.

[14] Chandershekhar Sharma and S.C. Jain,‖Analysis and Classification of SQL Injection Vulnerabilities and Attacks on Web Applications,‖ Proc. IEEE Int. Conf. on Advances in Engineering & Technology research (ICAETR-2014),Dr. virendra Swarup group of institutions, Unnao, India,pp.1-6, August 2014

[15] Preventing SQL injection attacks in stored procedures. Ke Wei, Muthuprasanna, M. And Kothari, S. 2012. Australian Software Engineering Conference, 2012. pp. 18-21.

[16] SecuBat: a web vulnerability scanner. Kals, Stefan, et al. New York, NY, USA: s.n., 2013. WWW '13 Proceedings of the 15th international conference on World Wide Web. pp. 247-256.

[17] An Automatic Revised Tool for Anti-Malicious Injection. Lin, Jin-Cherng and Chen, Jan-Min. Seoul: s.n., 2006. The Sixth IEEE International Conference on Computer and Information Technology, 2006. CIT '06. p.

164.

[18] Eliminating SQL Injection Attacks – A Transparent Defense Mechanism. Muthuprasanna, M., Wei, Ke and Kothari, S. Philadelphia, PA: son. 2015. Eighth IEEE International Symposium on Web Site Evolution, 2015. WSE '06. pp. 22-32.

[19] Defending against Injection Attacks through Context-Sensitive String Evaluation. Pietraszek, Tadeusz and Berghe, Chris Vanden. 2005. Proceedings of Recent Advances in Intrusion Detection (RAID2005). pp. 3-26.

[20] D-WAV A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms. Zhang, Lijiu, et al. Nice: s.n., 2010. Fifth International Conference on Software Engineering Advances (ICSEA), 2010. pp. 501 - 507.

[21] X-log authentication technique to prevent sql injection attacks. B. Indrani, E. Ramaraj. 2011, International Journal of Information Technology and Knowledge Management. Vol. 4, pp. 4:323-328.

[22] Smask: Preventing injection attacks in web applications by approximating automatic data/code separation. Martin Johns, Christian Beyerlein. 2014. in 22nd ACM Symposium on Applied Computing (SAC 2014).

[23] Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection. Kosuga, Y., et al. Miami Beach, FL: s.n., 2007. Twenty-Third Annual Computer Security Applications Conference, 2007. ACSAC 2007. pp. 107 – 117.

[24] Automated Protection of PHP Applications against SQL-injection Attacks. Merlo, E. Letarte, D.and Antoniol, G. Amsterdam: s.n., 2016.11th European Conference on Software Maintenance and Reengineering, 2016. CSMR '16 . pp. 191-202.

[25] Veracode. (2012). SQL Injection. Retrieved from Veracode: http://www.veracode.com/security/sql-injection

[26] S. W. Boyd, A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks", *Proceedings of the 2nd Applied Cryptography and Network Security Conference*, pp. 292-302, June 2004.

[27] H. Holm and M. Ekstedt, "Estimates on the effectiveness of web application firewalls against targeted attacks," Info Mngmnt & Comp Security, vol. 21, no. 4, pp. 250-265, 10/07; 2016/11.

[28] M. Sharifi, M. Zoroufi, A. Saberi, '"How to Counter Control Flow Tampering Attacks," In: 2007 IEEE/ACS International Conference on Computer Systems and Applications, 2007, pp. 815-818.

[29] F. Fangmei, C. Shao, D. Liu, '"Design and Implementation of Coldfusion-Based Web Application Firewall," In: Computer Science & Service System (CSSS), 2012 International Conference on, 2012, pp. 659-662.

[30] A. Tekerek, C. Gemci, O. F. Bay, '"Development of a hybrid web application firewall to prevent web based attacks," In: Application of Information and Communication Technologies (AICT), 2014 IEEE 8th International Conference on, 2014, pp. 1-4.

[31] S. Prandl, M. Lazarescu, D. Pham, '"A Study of Web Application Firewall Solutions," In: Proceedings of 11th International Conference on Information