IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 March 2023, accepted 3 April 2023, date of publication 11 April 2023, date of current version 27 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3266385

SURVEY

Input Validation Vulnerabilities in Web Applications: Systematic Review, Classification, and Analysis of the Current State-of-the-Art

FARIS FAISAL FADLALLA AND HUWAIDA T. ELSHOUSH
Department of Computer Science, Faculty of Mathematical Sciences and Informatics, University of Khartoum, Khartoum 11115, Sudan
Corresponding author: Faris Faisal Fadlalla (frsfaisall@gmail.com)

ABSTRACT In recent years, huge increase in attacks and data breaches is noticed. Most of the attacks are performed and focused on the vulnerabilities related to web applications. Hence, nowadays the mitigation of application vulnerabilities is an ignited research area. Thus, due to the potential high severity impacts of web application, many different approaches have been proposed in the past decades to mitigate the damages of application vulnerabilities. Static and dynamic analysis are the two main techniques used. In this paper, a new classification for web application input validation vulnerabilities is proffered. In addition, various techniques/tools that are used to detect them are analyzed and evaluated to apprehend their strengths and weaknesses. Thus, this paper provides both technical as well as literature countermeasures to input validation vulnerabilities. Moreover, various statistical distributions of the reviewed techniques were manifested and scrutinize in different aspects to reveal the perception of the prevailing techniques and the gaps in the literature. In addition, the most widespread metrics are also propounded.

INDEX TERMS Web security, static analysis, dynamic analysis, input validation vulnerabilities, source code review.

I. INTRODUCTION

Since world wide web (WWW) arrives in early 1990 and joined the internet, it becomes ubiquitous and very quickly it hosts every aspect ranging from simple static text page, e.g. news, article, to frameworks running complex web applications such as banks applications, Facebook, Twitter, Google, Amazon [1], [2], [3]. Every day, new vulnerabilities evolve in web applications due to the complexity and continuously new technologies being introduced as well as integrated processes to build web applications such as back-end language, front end language, traditional and NoSQL databases, Web cache and application programming interface (APIs) data interchange language, etcetera [4], [5]. Each time new classes of vulnerabilities in web applications evolve and it can be seen that the open web application security project (OWASP) and sysadmin, audit, network and security (SANS) constantly release new reports containing recent top vulnerabilities in the web [6], [7], [8], [9], [10], [11]. Each report contains new evolved vulnerabilities with its rank in terms of severity and complexity as well they mention and modify ranks of the previous vulnerabilities.

Many reviews [12], [13], [14], [15], [16], [17], [18] exist in the area of web vulnerabilities but they focus on general kind of vulnerabilities, nonetheless this paper focused on input validation issues, that occur in cloud-based or in house based application because these issues result from programmers that write software and do not have background about security functions nor use functions that will validate user input. Moreover, the existing review articles suffer from poor categorization and overlapping. Furthermore, up to our knowledge, there is no deep study for defending against them.

Our paper aims to proffer a systematic review on most popular vulnerabilities that have high risk values and can occur due to lack of user input validation. We organized the input validation vulnerabilities into a classification that differ from OWASP top 10 as well as SANS top 25 category,

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

as these are related to broad kind of vulnerabilities that have high risk values [6], [19].

Input validation vulnerabilities represent each vulnerability that arises because there is an external data received (source) and processed as well as arrived at a sensitive function (sink) in the web application without validation and sanitation or using validation functions in the wrong way. Examples of external data can come from different sources such as user input, file name input, log files, databases, file systems, HTTP headers, integration with other systems, APIs, auto-generated data, etc. Noteworthy, validation depends on the context in which the data will be used; hence, different types of validations are needed.

Input validation vulnerabilities can be detected with developed security framework more easily than other types of vulnerabilities such as session fixation, certificate validation, weak encryption, misconfiguration, default credential, verbose error messages or vulnerabilities related to the patch and insufficient logging as well as monitoring [20]. Therefore, our proposed classification of input validation vulnerabilities provides big picture and guide to researchers to determine which vulnerabilities to mitigate, detect/prevent especially when they work on static analysis tools. Hence, a new classification is inaugurated. Moreover, a comprehensive review of the input validation vulnerabilities research revealing the strengths and weaknesses of each method is propounded according to this new classification.

The remainder of this paper is organized as follows: Section II presents background and motivation. The research methodology is explained in section III. We introduced different kinds of techniques to reduce web application vulnerabilities, which includes static and dynamic techniques in section IV. Section V classifies the vulnerabilities related to input validation according to the proposed classification and provides a concise explanation of their solutions mentioned in the literature. The common evaluation metrics are presented in section VI The vulnerabilities are scrutinized and analyzed in section VII,. The limitations and future works are propounded in section VIII. Finally, section IX concludes the paper.

II. BACKGROUND AND MOTIVATION

The main issue of the web application vulnerabilities arises when it receives input from out of its control such as user input coming through entry points (where the external data is received and entered into the web application) such as $_GET in the PHP language without proper validation and processed by sensitive sink (function that will process external data, and if external data is not sanitized, may exploit the system.) such as mysqli_query [21], [22], [23], [24], [25]. Most of the attacks exploit improper validation and enter malicious metadata such as (‘, ‘‘, OR) to corrupt applications logic. Therefore, to protect the web app often sanitization or validation functions need to be added that handles any input entering web application before arriving sensitive sink [26], [27], [28]. Figure 1 demonstrates the source and sink idea.

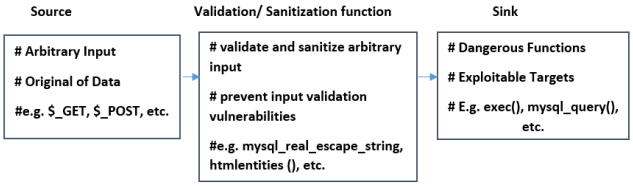

FIGURE 1. Source and sink concepts in source code review.

The motivation of this paper is that most of the studies in web application vulnerabilities discuss all kind of vulnerabilities or randomly selected issues but there are few papers written specially for input validation vulnerabilities [29], [30], [31], [32], [33], [34]. These kinds of vulnerabilities happen because users enter malicious inputs and servers have no validation or sanitization of these users inputs. OWASP [6], as well SANS [19], have categories for web application vulnerabilities but they are not grouped and thus have wide ranges. Additionally, previous review works, such as those in [12], [13], [14], [15], [16], [17], and [18], have their limitations in terms of broad and poor categorization, overlapping, and furthermore sometimes pay no attention to some of the vulnerabilities. Moreover, there is no deep study for defending against them. Ergo, this present work continues the efforts of the current reviews and aims to present a better ambit for comprehending and distinguishing the different methods of input validation vulnerabilities.

III. METHODOLOGY

Our methodology involved data collection, vulnerability analysis, and classification. To write this survey a total of 720 papers were downloaded during the whole steps using the keywords (‘‘WEB security’’ OR ‘‘WEB Attack’’ OR ‘‘WEB vulnerability’’ OR ‘‘Input validation vulnerabilities’’ OR ‘‘Source code review’’ OR ‘‘File Inclusion’’ OR ‘‘Directory Listing’’ OR ‘‘SQL Injection’’ OR ‘‘SQLI’’ ‘‘XSS attack’’ OR ‘‘XPath Injection’’ OR ‘‘LDAP Injection’’ OR ‘‘NoSQL Injection’’ OR ‘‘XSS’’ OR ‘‘Cross-Site Scripting’’ OR ‘‘Header Injection’’ OR ‘‘Email Injection’’ OR ‘‘Path Traversal’’ OR ‘‘CSRF’’ OR ‘‘Cross-Site Request Forgery’’ OR ‘‘SSRF’’ OR ‘‘Server side request forgery’’ OR ‘‘CORS’’ OR ‘‘Cross site request forgery’’ OR ‘‘XXE Injection’’ OR ‘‘XML external entity injection’’ OR ‘‘type juggling’’ OR ‘‘Deserialization’’ OR ‘‘Prototype Pollution’’ OR ‘‘command injection’’ OR ‘‘template injection’’) AND (‘‘source and sink’’ OR ‘‘Static analysis’’ OR ‘‘Dynamic analysis’’ OR ‘‘Machine Learning’’ OR ‘‘Deep Learning’’ OR ‘‘Source code review’’).

First collection of research articles were published between 2015 and 2022 related to input vulnerabilities from Engineering Village database. In specific, mostly research in SQL, XPATH, and XSS injection vulnerabilities were found in this database. Other research articles were obtained from trustworthy databases such as IEEE, ACM, Springer and OWASP. In particular, 506 research articles were downloaded from

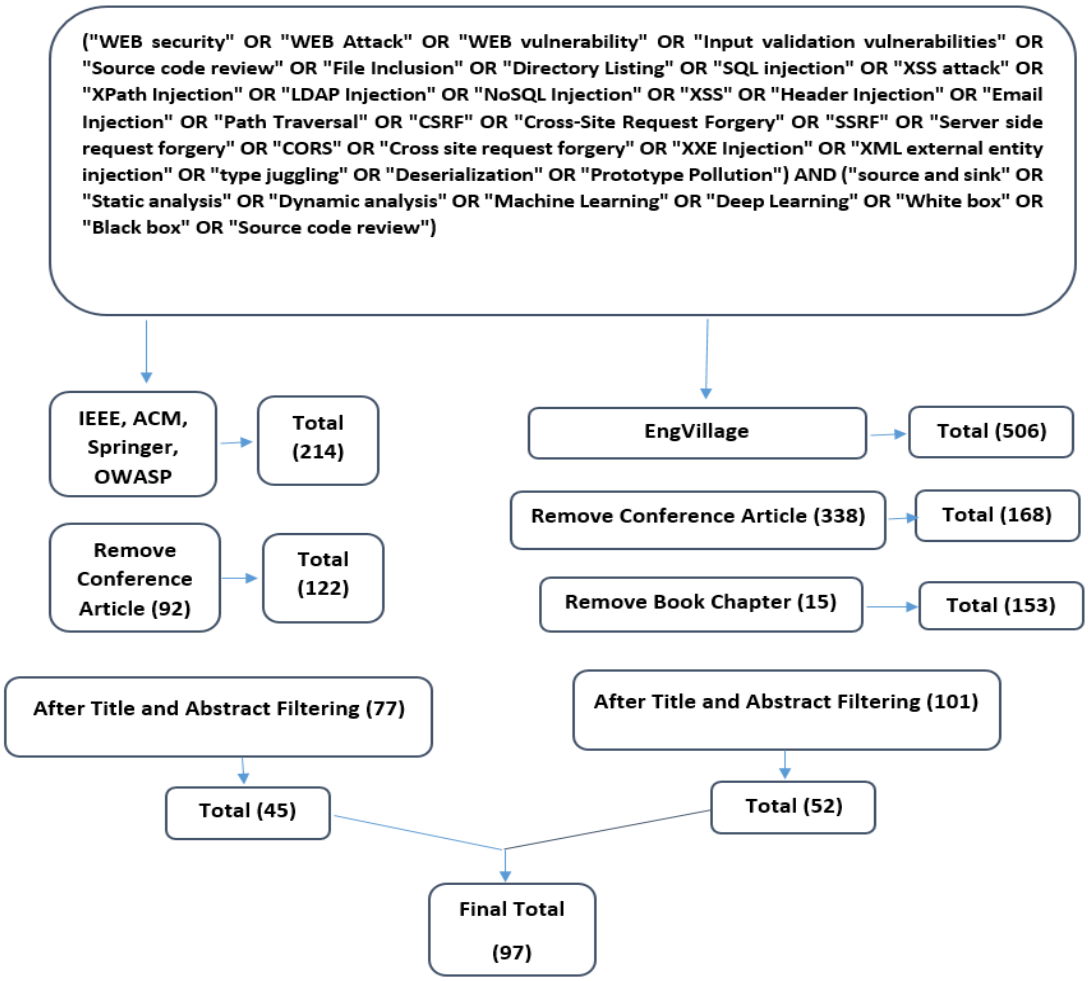

FIGURE 2. Data collection and filtering process.

Engineering Village database and filtered to 168 after removing conference articles. Then finally filtered to 153 research articles after removing book chapters. Finally, we came up with 52 research articles from Engineering Village. This last filtration was conducted after title and abstract filtering. These 52 research articles from Engineering Village were combined with 45 research articles from IEEE, ACM, Springer, and OWASP yielded 97 research articles as the final collection of data as depicted in Figure 2.

IV. TECHNIQUES TO REDUCE WEB APPLICATION VULNERABILITIES

There are numerous techniques developed to detect and prevent web appilcation vulnerabilities, some of them are widespread quick solutions that work as an addition security layer in web security. Security features implemented in the browser are considered client-side prevention, e.g., CSP, CORS, X-XSS-Protection header and HSTS [35], [36], [37], [38]. Other solutions based on web application firewall (WAF) are considered server-side prevention, where their mechanisms are analyzing traffic and preventing or sanitizing any request/input considered potentially malicious [39], [40], [41]. However, these kinds of widespread prevention techniques are not enough to make web applications completely secure because they lack completeness as they detect few kinds of vulnerabilities, not all of them. Furthermore, they can be bypassed because there is a variant method that can be used in the exploitation such as Cross-Site Scripting (XSS) attacks. Client-side protection viz X-XSS-Protection header is not supported by all browsers, for instance Firefox browser is an example. Therefore, they can be easily bypassed when browsing through Firefox [42].

Defensive programming is another technique used and advised by security consortia [43]. It consists of a set of habits and good practices that need to be followed when developing web applications. The core idea is to consider each supplied users input as malicious and not from a trusted source, and check all inputs and restrict them solely to intended recipients [44], [45], [46], [47], [48]. In specific, there are frameworks evolved nowadays used to build web application e.g., Rails, Django and Laravel that support defensive programming by providing built-in security features that assist in

input validation, authentication and authorization. Additionally, the OWASP provides guidance to write secure code [49]. However, defensive programming cannot be adopted to be a final solution for making each web application free from vulnerabilities because it is prone to human fault which is its nature and inevitable. Moreover, complexity of code with large technology integrated in web application harden the task even more.

Basically, there are two main classes of techniques to reduce web application vulnerabilities namely, static and dynamic analysis [50]. However, recently researchers utilized data mining and machine learning (ML) together with these techniques to yield more efficacious results. Hereafter, these techniques are scrutinized.

A. STATIC ANALYSIS

Static analysis can be carried out at the implementation phase of a security development lifecycle (SDL), where it looks for vulnerabilities in source codes and trying to flag them without executing the applications [51], [52], [53], [54], [55]. Former researches in static analysis were focused on older vulnerabilities such as heap and buffer overflow as well as race conditions [56]. Static analysis techniques are often derived from compiler technologies and actually there are more than one technique to detect vulnerabilities inside source code that can be combined into one solution [57]. These techniques are:

- *Control Flow Graph (CFG)*: extracts and slice source code. Then the sliced code is parsed and an abstract syntax tree (AST) is built. To jump and tie paths between blocks, a directed edge is used [58].
- *Taint Analysis*: is the most famous technique in software testing. It monitors at which entry point (source) user input is received and trace it until arriving to a vulnerable function (sink). If there is no sanitization function or validation implemented from entry point to the vulnerable function, it will be flagged as vulnerability [59].
- *Lexical Analysis*: extracts and slice block of code. Then it converts the syntax into group of tokens in an attempt to abstract source code. Each function, represented with a token, will be compared with a vulnerable function (sink) that is stored in a database. Some tools that implement this technique are Flawfinder, ITS4 and RATS for C and C++ [51], [60]. However, this technique may generate false positives. For instance, there may exist a variable in a source code having the same name as a vulnerable function that is stored in the database.

The advantages of static analysis may be summarized as:

- Cover 100% of the source code
- detect more vulnerabilities than dynamic analysis
- detect vulnerabilities from developing phase
- solve a vulnerability from its root by adding a sensitization function

Nevertheless, static analysis has disadvantages viz:

- not applicable when there is no source code available
- prone to more false positives
- cannot detect logic vulnerabilities
- need a good understanding of programming languages

B. DYNAMIC ANALYSIS

On the other hand, dynamic analysis is a technique that tests applications without searching in the source code. Instead it is performed at run time with direct interaction with an application [43], [61], [62], [63], [64], [65], [66], [67], [68]. This technique tends to be simple to apply because it does not require knowledge about the program to test, furthermore its interaction with the program is limited to the program's entry points [69].

The advantages of dynamic analysis are hereafter outlined:

- covers only parts of the source code of current execution paths and depends on test cases used
- can detect vulnerabilities outside the code and in third-party interfaces
- more precise than static analysis
- mimic external attacks executed by attackers
- provide cost-effective detection for certain types of important vulnerabilities
- simple to apply because no need for programming language knowledge

Nonetheless, dynamic analysis has some disadvantages such as:

- prone to more false positive because it does not cover all source code
- predict vulnerabilities based on the received responses without knowledge of the source code.
- therefore, it suffers from false positive/negative, and
- often requires human investigation as well as it detects fewer range of vulnerabilities than static analysis.

C. HYBRID ANALYSIS

Static and dynamic analysis have complementing advantages, and this has led researchers to create amalgamation to attain the best of the two techniques. Actually, each technique detects distinct sets of vulnerabilities with some overlap [70], [71].

D. DATA MINING (DM) AND MACHINE LEARNING (ML)

Recently DM and ML are used frequently and prove better results when combined with the basic techniques, where they automatically obtain and learn knowledge through ML algorithms. On the contrary, in static and dynamic techniques, human provides signature of the vulnerabilities [72], [73], [74], [75], [76], [77], [78].

V. INPUT VALIDATION VULNERABILITIES IN WEB APPLICATIONS

In this section, most of the user injection vulnerabilities and their exploitations are examined as well as how to protect vulnerable codes from user input vulnerabilities is scrutinized. We propose a new classification for the input validation vulnerabilities in web applications. The vulnerabilities

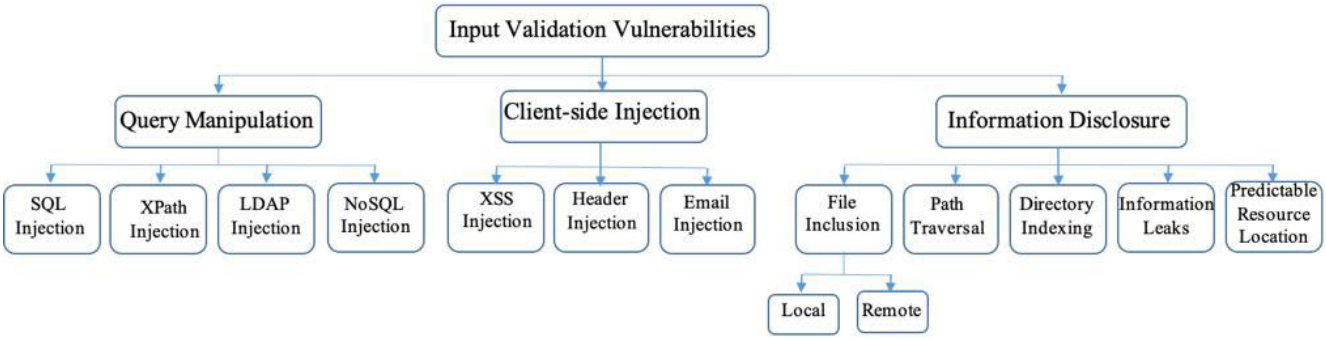

FIGURE 3. Proposed classification for the input validation vulnerabilities.

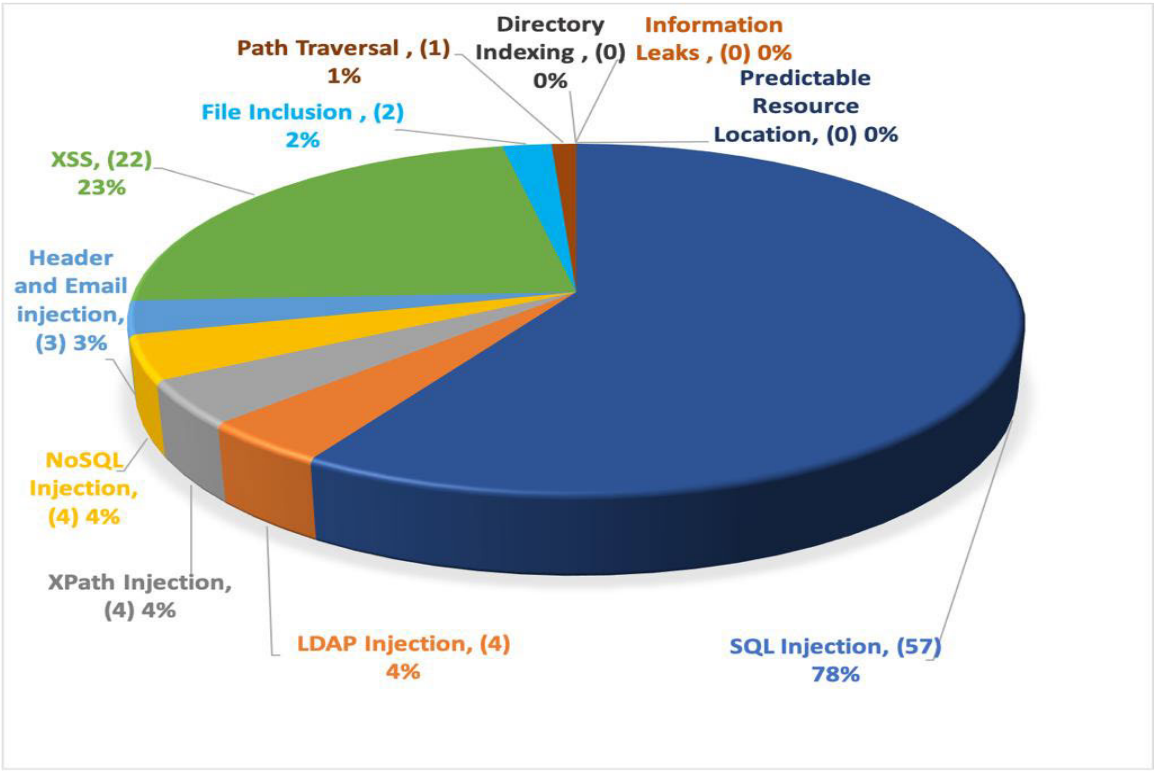

FIGURE 4. Distribution of the three proposed categories to reduce web application vulnerabilities.

were classified into three categories namely, query manipulation, client-side injection and information disclosure as illustrated in Figure 3. Each proposed category is scrutinized, exploring all its possible exploited scenarios as well as its ranking released by OWASP in 2007, 2010, 2013 and 2017, as demonstrated in Table 1. Furthermore, Table 2 presents sensitive functions written by PHP language that make web application vulnerable if there is no filtering or best practices used. Moreover, the countermeasures are further expounded. Sensitive functions, which are also called sinks, need to be noted and reviewed in the source code [79], [80]. Figure 4 depicts the distribution of the three proposed categories for reducing web application vulnerabilities and further shows the subclasses of each category, together with the number of reviewed articles (quoted between brackets) and the percentage of distribution. Hereafter, each category is further elucidated, and state-of-the-art solutions are presented.

A. QUERY MANIPULATION

This category includes attacks that take advantage of the query and change its intended action from normal to malicious. Such attacks execute SQLI, LDAP injection (LDAPI), XPath injection (XPathI) and NoSQL Injection (NoSQLI). In the following, we explore each type, give examples, and present literature solutions to avoid it.

1) SQL INJECTION

Structured Query Language Injection (SQLI) vulnerabilities occur when web application enquires users for input such as an authentication page that enquires a username and password

TABLE 1. Proposed classification for the input validation vulnerabilities, their attacks types, goals and ranking.

Category	Overview	Attack	Attack Types	Attack Vector	Attack Goals	OWASP and SANS Rank
Query manipulation	Arise when manipulating query interacts with the server.	SQL Injection	•Error based •Boolean based •Time based •Blind SQLI •Second order •Union based •Out-of-band	•User input •Cookies •Integrated system •Local file on server	•Gain full control of the server •Read or write file system •Remote code execution •Violate integrity, availability and confidentiality •Steal valuable information •Bypass authentication •Privilege escalation	OWASP: 2007 Rank 2 2010 Rank 1 2013 Rank 1 2017 Rank 1 2021 Rank 3 SANS: Rank 6
		XPath Injection	•Normal injection •Blind injection	•User input •Authentication page	•Bypass authentication •Access administrative account •User can access entire XML document	OWASP: 2007 Rank 2 2010 Rank 1 2013 Rank 1 2017 Rank 1 2021 Rank 3
		LDAP Injection	•Normal and blind injection	•Login page •Parameter for lookup	•Retrieve potentially sensitive data from LDAP directory •Login bypass •Information Alteration •Privilege escalation	OWASP: 2007 Rank 2 2010 Rank 1 2013 Rank 1 2017 Rank 1 2021 Rank 3
		NoSQL Injection	•Time based •Blind based •Boolean based	•Different method parameter such as GET, posat and etc. •Input that interact with nosql databases	•Allow to access administrator without supplying the correct password •Modify stored data	OWASP: 2013 Rank 1 2017 Rank 1 2021 Rank 3
Client-side injection	Vulnerabilities that target and effect users.	XSS	•Reflected •Stored	•User input •Upload image	•Steal cookies •Hijack users browser •Damage reputation	OWASP: 2007 Rank 1 2010 Rank 2 2013 Rank 1 2017 Rank 7 2021 Rank 3 SANS: N/A
		Header Injection	N/A	HTTP header	•Response splitting •Cross-site scripting •Session fixation •Malicious redirection	OWASP: 2013 Rank 1 2017 Rank 6 2021 Rank 5 SANS: N/A
		Email Injection	N/A	Email header	•Manipulate email components (such as content, sender, receiver, etc.).	OWASP: 2013 Rank 1 2017 Rank 1 2021 Rank 3 SANS: N/A
Information Disclosure	That help to disclose/ read operating system files or critical web site info.	Files Inclusion	•Local file inclusion •Remote file inclusion	•User input •Malicious file uploaded on server •Read/ include file from remotely hacker server	•Reading sensitive files as well configuration from the server •Disclose/read source code •Gain full control of the server	OWASP: 2013 N/A 2017 N/A 2021 N/A SANS: N/A
		Path Traversal	N/A	User input	•Read arbitrary files such as sensitive operating system files, configuration and application code on the back-end system	OWASP: 2013 N/A 2017 N/A 2021 N/A SANS: N/A
		Information Leaks	N/A	Developer comment •Public uploaded code •Wayback machine(web pages saved over time)	•Can allow attacker to cause damage to the application or control it. Attack level depend on the data revealed to the user	OWASP: 2013 N/A 2017 N/A 2021 N/A SANS: N/A
		Directory Listing	N/A	User input	•Allow attacker to read sensitive files such database configuration or third-party services that assist attacker to escalate attack.	OWASP: 2013 N/A 2017 N/A 2021 N/A SANS: N/A
		Predictable Resource Location	N/A	•Wayback machine •User input	•Disclose sensitive and critical information related to the web application database, passwords, admin panel, machine names that host web application and paths to other sensitive areas.	OWASP: 2013 N/A 2017 N/A 2021 N/A SANS: N/A

TABLE 2. Vulnerable sink and countermeasures.

Attack	Vulnerable Sink	Countermeasures/PHP function
SQL Injection	Mysql_query Mysqli_query Mysqldb_query	Sanitization or prepared statement mysql_real_escape_string()
XPath Injection	Xpath()	Sanitization or prevent meta-character
LDAP Injection	Ldap_search()	Encoding and input validation Prevent meta-characters at least at least ( ) ! & *
NoSQL Injection	Find()	Escape from these characters: & ; / : ' * as well as sanitization by using mysql_real_escape_string function
XSS	Echo()	Sanitization using function such as htmlentities() and/or by encoding output
Header Injection	Header() Setcookie()	Avoid returning user input into HTTP headers or by sanitizing user input for line-feed characters and carriage-return
Email Injection		Protection against this vulnerability is the same as HI.
Local File Inclusion	Include() Require() Include_once() Require_once()	Developer needs to explicitly build white list for required files or use if statement as well to not allow users to control file include feature
Remote File Inclusion	Same as LFI	Disable these two features to eliminate remote file inclusion: -Allow_url_fopen -Allow_url_include
Path Traversal		-Avoid passing user input to the file system. -Validate user input -White list
Directory Listing	Related to web server	Prevent server from listing directories through editing htaccess or server configuration file.
Information Leaks		Harden server and application by restrict data transmitted to the user in the comments, and disable debug mode as well handle exception.
Predictable Resource Location		Prevent developer form upload backup into web application directory that accessible by user. Or make credential to access sensitive files URL.

or any related input data such as hypertext transfer protocol (HTTP) headers that hence interact with databases through SQL language without validation or sanitization. A successful SQLI can lead to access and modify (insert/ update/ delete) and hence storing of confidential data to the database. It can as well lead to read and write file system and execute commands on the server that cause full control.

Figure 5 shows PHP authentication code interacting with a database and vulnerable to SQLI. Line 1 starts connection with the database. Line 2 and 3 are received data through the users calling source. Line 4 is a constructed query that executes at the database when send in line 5 through mysqli_query() function called sink. If the user inserts malicious data in line 2 username field such as admin' - -, this will cause script to return admin info and allows to bypass the login form without actually knowing a valid password and valid username. The constructed query will be:

*SELECT * FROM users WHERE username='admin' - -' AND password='foo'*

whereas - - character interpreted as a comment in SQL language. Therefore, any supplied data after username field will not be considered as part of the query and the password will never be checked.

This code is vulnerable because there is no sanitization or prepared statement (a.k.a. parameterized queries) are used. The concept of prepared statement is that an SQL statement is sent to the database server and parsed separately for any parameters to eliminate SQLI. Whereas sanitization is performed by using function that escapes special characters such as ', '', and \ that can lead to SQLI, where *mysql_real_escape_string()* function is a real example.

```
$con = mysqli_connect("localhost", "username", "password");
$user = $_POST['user'];
$pass = $_POST['password'];
$q = "SELECT * FROM users WHERE username='$user' AND password='$pass'";
$result = mysqli_query($q);
```

FIGURE 5. PHP login script vulnerable to SQLI [20].

Therefore, username at line 2 in Figure 5 should by sanitized to make the code secure:

$user = mysql_real_escape_string($_POST['user']);

Note that also password filled input at line 3 needs to be sanitized. Noteworthy, the mysql_real_escape_string is sufficient for the SQL statement in the context of Figure 5. In other situations it is not enough (e.g., ...Where id = $id. Clause without quotes. $id = −1 OR 1 = 1). so prepared statement is better in SQL Injection attack.

State-of-the-art research of SQL Injection Solutions

Figure 6 demonstrates the distribution of SQL Injection techniques used in reducing web application vulnerabilities. Multifarious researchers [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94] implement a pattern matching algorithm to detect and prevent SQL Injection attacks. All applied dynamic analysis except [89], [95], [96], [97] whom used static analysis. They check user SQL input and find whether they are SQL injected or not using various approaches viz. scanner [90], [92], [94], regular expression [84], string pattern [86], [97], keyword matching [93], no sanitization [82], sanitizing or blocking [85], [98], access control methods [88], and with the cooperation of intrusion detection systems (IDS) [81], [87]. Figure 7 depicts the distribution of the different pattern matching techniques used in SQLI.

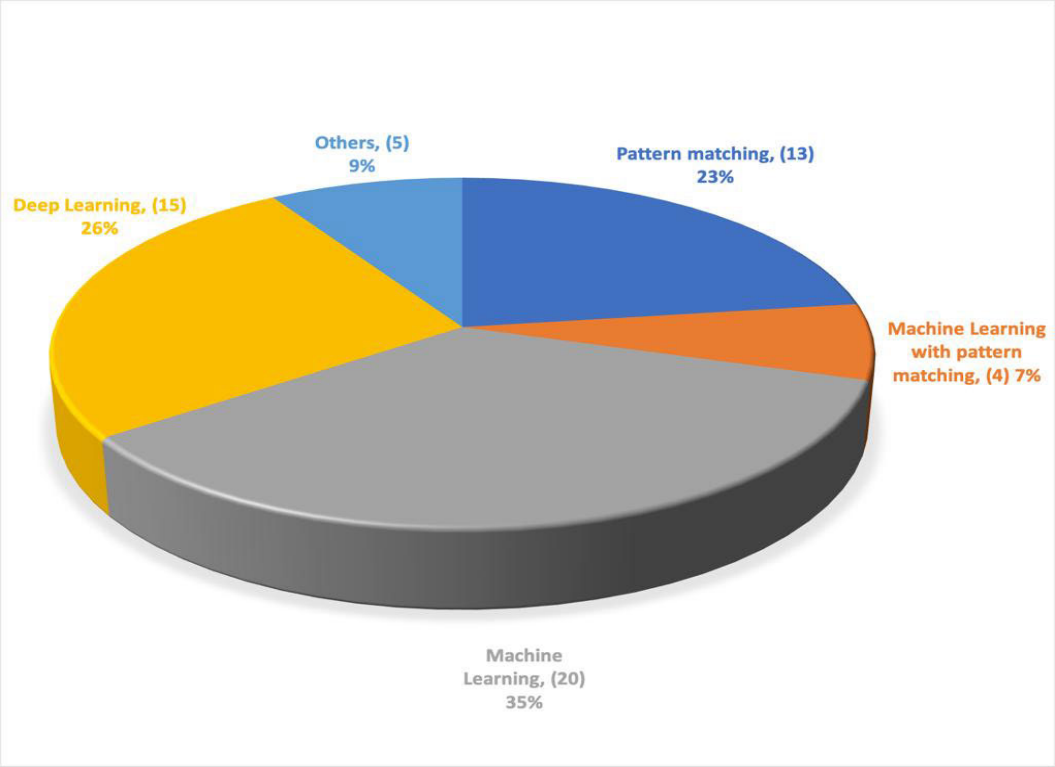

FIGURE 6. SQL Injection techniques to reduce web application vulnerabilities.

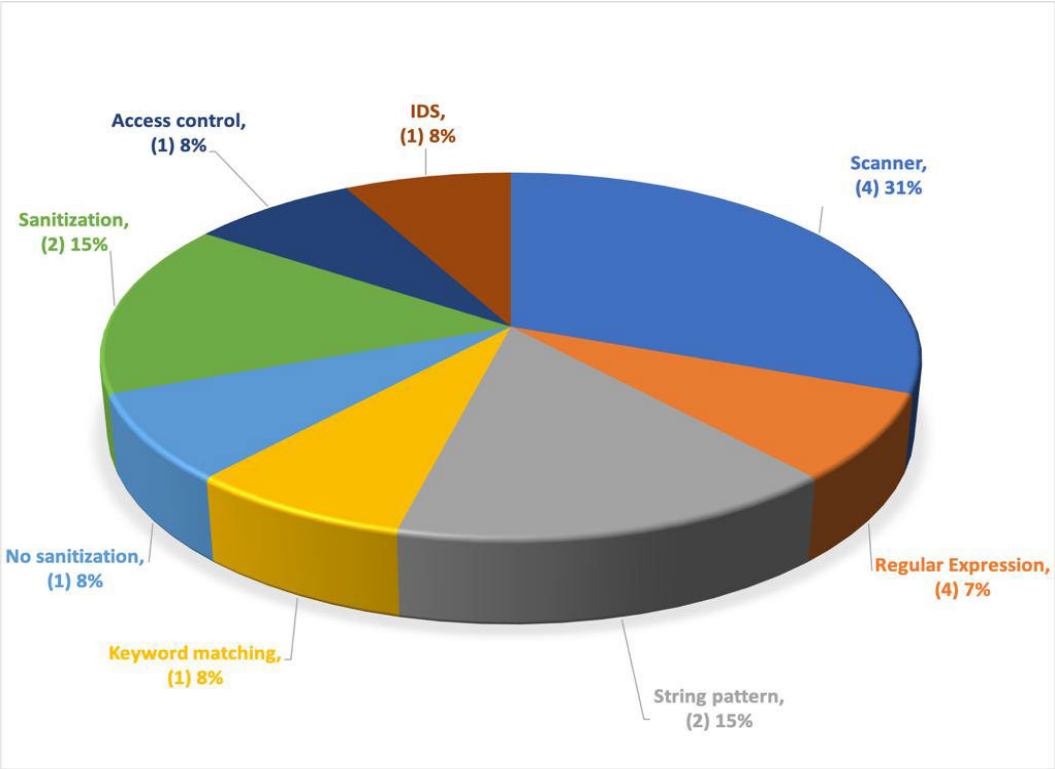

FIGURE 7. SQL Injection - Pattern matching techniques.

Using static analysis, Umar et al. [89] developed a tool called EPSQLIFix, which uses grammar reachability analysis to detect as well as remove SQL Injection. Yet, they need to evaluate their method. Whilst working on a different dimension, researcher D'silva et al. [95] built lightweight static technique to detect and prevent SQLI in authentication page

by using hash concept. Their solution generates hash value for the SQL authentication query concatenated with user's credential and then checks this value each time a user tries to authenticate. This solution will allow to check if there are any modification or SQL Injection attack in the query also validate correct credentials. It can be implemented in most languages. However, this method only protects from SQL Injection in authentication page, yet SQL vulnerability can be found in various pages such as search feature or file inclusion.

Other researchers utilized scanner in pattern matching. In particular, Saoudi et al. [90] provided a scanner tool called SQLIVD, which detects injection by comparing HTTP response results. The author tested SQLIVD tool with other three well-known scanners named W3af, ZAP, and Acunetix. It shows better results for detecting blind as well as error-based SQL Injection. Alike, Aliero et al. [92] developed a black box scanner called SQLIVS that fuzz web server. It has anti crawling feature that performs data preprocessing such as eliminating duplicates and filtering stored URLs more accurately as well as analyzing attacked page response. However, they perform experiments on vulnerable web applications they created, and not public open source vulnerable application that have many different vulnerable scenario. Later the same authors, Aliero et al. [92] developed an automatic scanner tool based on an object-oriented approach to detect SQL Injection in black box testing without scanning source code. Their method lacks comparison with more advanced scanners. Recently, Thombare and Soni [94] also presented a scanner consisting of four elements: crawling by visiting URLs that talk with database, attacking by sending malicious SQL request, analysis by analyzing response page to determine if there are vulnerabilities or not, and their final component is report generation.

Using regular expression, Chenyu and Fan [84] proffered an intention-oriented detection approach for submitted queries and checks if it is malicious or not.

From a different facet using string pattern, Ceccato et al. [86] presents a security oracle for SQL-injection vulnerabilities (SOFIA) that works through making classification for requests. This tool intercepts request specially SQL statement and makes some processes on it: parsing, prunning, and eventually classifying as safe or malicious. On the other hand, Abikoye et al. [97] starts by: preparing, parsing, then identifying and extracting SQLI type patterns and finally preventing identified SQL Injection attacks. Different actions may be performed such as blocking user, reseting HTTP request and displaying warning message. Yet, their method needs more processing and time.

Using keyword matching, Kumar et al. [93] used a two-level restricted application prevention (TRAP) technique, which works at the middle tier using pattern passed keyword filtering and the DB tier using SQL rewriting.

Some such as Li et al. [82] analyze the source and determines injection points that have no sanitization.

Working differently, Karuparthi and Zhou [85] proposed an enhanced dynamic approach to detect SQL attacks that works through efficient matching techniques and sanitizing or blocking data before it arrives at the database server. Their method lacks evaluation. Likewise using sanitization, Jahanshahi et al. [98] submitted a hybrid static-dynamic tool, called SQLBlock, which works in PHP language by limiting each function for accessing the database. SQLBlock works as a plugin for PHP and MySQL and does not require any modification to the web app.

Using access control methods, Zhu et al. [88] developed a new technique based on two-tiers, where the first tier is a fine-grained role-based access control model and the second tier is an extended AC multi-pattern matching algorithm.

With the aid of IDSs, Patel and Shekoka [81] used AIIDA-SQL techniques and SQLMAP tool, and their algorithm yields better accuracy and memory consumption results. In a similar fashion, Lodeiro-Santiago et al. [87] presents an improvement of current IDSs based on the use of a frequency analysis and the previous behavior of one of the most used database audit software, SQL Map. They use training data and achieve positive detection close to 99%.

Alike research [87], other researchers [83], [91], [99] combine pattern matching with ML techniques. Specifically, Gao et al. [91] presented a model, called ATTAR, which uses access behavior mining and grammar pattern recognition. The author extracts features to detect injection from a custom web access log file and trains it with support vector machine (SVM), Naive Bayesian, random Forest, ID3, and K-means algorithms. Similarly, Latchoumi et al. [99] employed SVM trained with malicious SQL Injection syntax to predict bad input also can detect new malicious syntax by matching it with a minimum amount of syntax. This technique can work in big data environment. However, there are no comparison with other techniques to measure its efficiency and results properly. In addition, it detects limited types of SQL Injection. In a like manner, Uwagbole et al. [83] provide API service called SQLIA that works as a proxy which captures requests before arriving at the back-end database and decrypts the web traffic. Then the intercepted request traffic will be parsed for pattern matching.

From a different aspect, multitudinous researchers [96], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111], [112], [113], [114], [115], [116], [117] deployed ML techniques to detect/prevent SQLI. Working in a static way, Zhang [96] developed classifier model based on ML, which works through scanning source code and deciding if there is SQL Injection or input is sanitized properly. This model shows highly accurate results, yet they need to increase the dataset.

However, most researchers use dynamic analysis. Concerning web applications, in particular Luo [114] studied the limitations of web fuzzing and generated an ML solution called SQLI-Fuzzer that overcomes traditional fuzzing limitations. Rahul et al. [115] developed WAF that receives a request before it arrives at a web application and if it detects a malicious request will redirect the attacker to a honeypot rather than an actual web application. Whilst,

researchers [101] suggest API and web services respectively that work as proxy in .NET application to detect and prevent malicious requests from reaching the back-end database using ML. The same authors, in [102] present an application context pattern-driven corpus to train a supervised learning model. They use ML to train their model using algorithms of Two-Class Logistic Regression (TC LR) and Two-Class Support Vector Machine (TC SVM) implemented on Microsoft Azure ML (MAML) studio to mitigate SQL Injection attack.

Using also classification SVM, Li and Zhang [106] improve the Term Frequency-Inverse Document Frequency (TFIDF) algorithm through distribution of feature words in the same kind of statement. They also combine TFIDF algorithm with SVM, yielding better accuracy results. Another researcher uses SVM, Chen et al. [103] process the text data of an HTTP request and effectively determine malicious SQL payload. Other researchers implement different ML techniques. In particular, Sivasangari et al. [116] used AdaBoost algorithm, while the algorithm of researchers Pathak and Yadav [109] was based on trained neural network mode.

Others presented distinct approaches, viz Parashar et al. [111] used text rank summarization with ML classification. Working by syntax analysis and emulation, Kuroki et al. [108] developed a method to detect the intention of SQL queries in HTTP requests. The author defines four types of the intention of SQLI (Reconnaissance, Leakage of system information, Leakage of database content, or Falsification).

A different approach presented by Tripathy et al. [110] developing an ML classifier to detect SQL Injection in cloud Software as a Service (SaaS) module.

With the assistance of IDSs, [104] and [105] presented different approaches. For instance, Ross et al. [104] work by capturing request data in two points: first in web application using snort IDS and save data in PCAP file. Second, using Datiphy appliance node that work as proxy between web application and remote MySQL server and save captured data in CSV format. These two datasets process using bash shell scripts and save into one file to create the correlated dataset. Volkova et al. [105] applied ML approaches for identifying SQLI in the HTTP query string. They compare results from SVMs, Rule-based IDS, Neural Network with Dropout layers, Multilayer Perceptron (MLP), and Deep Sequential Models (Gated Recurrent Units, and Long Short-Term Memory) using bag-of-word techniques, word embedding for query string vectorization, and multiple string analysis.

Comparing different ML techniques, Hasan et al. [107] proposed a heuristic algorithm that compares best five classifiers having best accuracy results. Their results show that both Bagged Trees Ensemble Boosted classifiers provide the highest classification accuracy (93.8%). Choudhary et al. [113], after comparing many algorithms, concludes that the Decision Tree, Neural Network, and Naive Bayes-based method provide better accuracy results. On the other hand, Adebiyi et al. [117] discovered that decision tree shows better accuracy results than Naïve Bayes and K Nearest Neighbour classifiers.

In the last few years, researchers [118], [119], [120], [121], [122], [123], [124], [125], [126], [127], [128], [129], [130], [131] began to consider using deep learning (DL) techniques to detect and prevent SQL Injection.

Researchers [120] and [118] extracted the features values in the HTTP traffic. Using deep belief network (DBN), the model of Zhang et al. [120] works at network layer through sniffing traffic, specifically HTTP requests and determining if it contains malicious SQL request or not based on training data. They further compare between four models (long short-term memory (LSTM), MLP, CNN, and DBN) and the DBN model shows better accuracy results. Deploying also LSTM, Tang et al. [118], using MLP and LSTM networks, extract the feature values in the HTTP traffic to detect user behavior that contains SQLI. Conversely, Li et al. [119] presented an LSTM based SQLI detection method, and uses injection sample generation that is based on data transmission channel from the perspective of penetration. This method can model SQLI and generate valid positive samples. To detect SQLI in transportation system, Li et al. [119] likewise use the LSTM to complete feature extraction automatically. Li et al. [124] proffered an QL-LSTM model based on both traffic features and text features. Their model also utilizes a multi-layer LSTM structure. Deploying two LSTMs, researchers [127], [128], [129] all uses a bidirectional LSTM (BiLSTM). For instance, Gandhi et al. [127] presented a hybrid module based on CNN combined with BiLSTM. Wen et al. [128] enhanced the BiLSTM model for SQL attack detection by adding an attention mechanism. Farea et al. [129] also uses BiLSTM for SQL and XSS attack detection. However, another researcher Tang et al. [123] uses artificial neural network (ANN) to model training data in MLP and LSTM.

Researchers [126] and [125] based their models on a deep learning MLP algorithm. Particularly, Jothi et al. [126] model shows good accuracy results and can also scale with other type of injection issues easily. However, it needs more payload types to assess it well. Whilst the model of Chen et al. [125], which is based on CNN and MLP algorithms, combines lexical analysis and data preprocessing techniques to achieve higher accuracy results. Their model is able to detect some 0-day attacks. However, it cannot detect second-order SQL attacks. Using CNN also, Xie et al. [121] utilizes elastic-pooling CNN (EP-CNN) algorithm to detect SQL Injection. It is harder to bypass and can identify new attacks.

Recently, Zhang et al. [130] presented an SQLNN deep neural network model for SQL Injection attack detection. The author compares this model with LSTM, KNN, and DT algorithms and SQLNN shows better accuracy, precision, recall, and F1-score results. Lately, Falor et al. [131] compares different ML and DL algorithms for SQL Injection attack detection, and came out with the result that CNN outperforms other algorithms in precision, accuracy, and recall. In another research [132], Li et al. suggested an algorithm

TABLE 3. State-of-the-art of the research on **Query Manipulation - SQL Injection.**

Author/ Year	Technique	Aim	Target App.	Prog. Lang.	Pros	Drawback	Metrics
Static Techniques							
Abikoye et al [98] 2020	Pattern matching (String)	Detection and prevention	Custom web app	PHP	•Many characteristic for detection •Detect/prevent various kinds of SQLI	•Need more processing and time	•TP; TN
K. Zhang [97] 2019	ML	Detection	Real web app	PHP	•Highly precise classification model	•Need to increase dataset	•Precision 95.4% • Accuracy •Recall
Umar et al [90] 2018	Grammar reachability analysis	Detection	Custom web app	JAVA	•Detect and remove injection point	•No evaluations	N/A
D'silva et al [96] 2017	Hash technique	Detection and prevention	Custom web app	.NET	•Easy to implement •Hashing technique is reliable	•Uses week hashing •Only protects in authentication page	•TP; TN
Dynamic Techniques							
Saoudi et al [91] 2019	Pattern matching (Scanner)	Detection	Real web app	N/A	•Shows better results when compared with W3af, ZAP scanners	•Unable to bypass all existing filters	•TP
Aliero et al [93] 2019	Pattern matching (Scanner)	Detection	Custom web app	N/A	•Anti-crawling •Data preprocessing •Uses more than one analysis technique	•Is not tested in public open source	• Precision • Recall • Accuracy 100%
Aliero et al [93] 2020	Pattern matching (Scanner)	Detection	Custom web app	N/A	•No false alarms •Uses many techniques for detection	•Lacks comparison	• TP; FP • Precision • Recall
Thombare and Soni [95] 2022	Pattern matching (Scanner)	Detection	N/A	N/A	•More component in the scanner	•Not compared with other scanners	N/A
Chenyu and Fan [85] 2016	Pattern matching (Regular expression)	Detection	Real web app	N/A	•Detect various types of SQL Injection	•Could not distinguish the intention for different users	•TP
Ceccato et al [87] 2016	Pattern matching (String)	Detection	Real web app	N/A	• recall 100%	•Lacks comparison	•TP; TN; FP; FN
Kumar et al. [94] 2021	Pattern matching (keyword)	Detection and prevention	N/A	N/A	• Robust •High effective detection	•More time and resources	• Time
Karuparthi and Zhou [86] 2016	Pattern matching (Sanitization)	Detection	N/A	N/A	•Additional layer for detection	•No evaluations	N/A
Zhu et al [89] 2017	Pattern matching (access control)	Detection and Prevention	Real web app	N/A	• Reduce the SQL detection time	• Lacks comparison	• Accuracy
Patel and Shekokar [82] 2015	Pattern matching (IDS)	Detection and Prevention	N/A	N/A	•Better accuracy and memory consumption	In some scenarioS give inaccurate results	•Memory consumption •Accuracy

TABLE 3. *(Continued.)* **State-of-the-art of the research on Query Manipulation - SQL Injection.**

Lodeiro-Santiago et al [88] 2017	ML with pattern matching (IDS)	Detection and prevention	N/A	N/A	• Positive detection close to 99%	•Lacks comparison	N/A
Gao et al [92] 2019	ML with pattern matching	Detection	N/A	N/A	•Detect multi SQL Injection types	•Need to use standard dataset	•FP; FN
Latchoumi et al. [100] 2020	ML (SVM) with pattern matching	Detection and prevention	Custom web app	N/A	• Works in big data environment • Detect new payload of SQL	• Lacks comparison •Detect limited types of SQL Injection	N/A
Uwagbole et al [84] 2016	ML with pattern matching	Detection and prevention	N/A	N/A	•Uses large dataset • High accuracy	•Need to compare their tool with others	• Accuracy 92.9% •Precision • Recall
Kamtuo and Soomlek [101] 2016	ML	Detection	N/A	N/A	• Comprehensive dataset used	• Lacks comparison	•Detection rate •FP • Accuracy
Uwagbole et al [102] 2017	ML	Detection and prevention	Custom Web app	.NET	• Works better with big data • Good data preprocessing • Fast and scalable	Uses few classifier	• Accuracy • Precision • Recall
Ross et al [105] 2018	ML (IDSs)	Detection	N/A	N/A	• Collects log from different layers • High accuracy	• Needs more time to capture all requested data	• TP; TN; FP; FN • Accuracy 98.055%
Volkova et al. [106] 2019	ML (IDSs)	Detection	N/A	N/A	• Compares various ML technique	• Needs to increase dataset	• Accuracy • FP •Predication rate
Kuroki et al. [109] 2020	ML	Detection	N/A	N/A	• Good accuracy result in artificial dataset • High recall value	• Less accurate in real-world dataset	• Precision • Accuracy •Recall 98.3%
Pathak & Yadav [110] 2020	ML	Detection	N/A	N/A	•Good comparisons between algorithms • High accuracy	• Does not mention which other vulnerabilities can detect.	• TP, TN, FP, FN • Accuracy 97.897%
Tripathy et al. [111] 2020	ML	Detection	N/A	N/A	•High accuracy •Good comparison	• It is not tested in big dataset	• Accuracy 98% •Precision • Recall
Parashar et al. [112] 2021	ML	Detection	Real web app	N/A	• High accuracy	•Need more payload types	•Accuracy 78% •Precision •Recall
Gowtham and Pramod [113] 2021	ML	Detection	N/A	N/A	• High accuracy	•Need more measurements	•Accuracy 98%

TABLE 3. *(Continued.)* **State-of-the-art of the research on Query Manipulation - SQL Injection.**

Rahul et al. [116] 2021	ML	Detection and Prevention	N/A	N/A	•Multi-layer for detection	•Need more resource and configuration	•Accuracy •Precision • Recall
Hasan et al. [108] 2019	ML	Detection and prevention	N/A	N/A	•Easily scalable •High accuracy	•Need to add more features to give better accuracy result.	•TP;TN •Accuracy 93.8%
Raj et al. [114] 2021	ML	Detection	N/A	N/A	• High accuracy	•Lacks comparison	•Accuracy
Adebiyi et al. [118] 2021	ML	Detection	N/A	N/A	• High accuracy	•Need to increase dataset size	•TP; TN; FP; FN •Accuracy 98.11%
Chen et al [104] 2018	ML (SVM)	Detection	N/A	N/A	• Overcomes the defects of the existing rule matching method	•Requires more memory	• +ve/-ve classification • Recall
Li and Zhang [107] 2019	ML (TFIDF with SVM)	Detection	N/A	N/A	• High F-score •Better accuracy	•Needs to work with more dataset	• Precision • Recall •F-score 99.21% • Accuracy
Uwagbole et al [103] 2017	ML (context pattern-driven)	Detection	N/A	N/A	• Uses many algorithms in model training	• Need to identify the type of SQL Injection	• TP; TN; FP; FN
Sivasangari et al. [117] 2021	ML (Adaboost)	Detection	N/A	N/A	•High accuracy and precision results.	•Need to increase dataset size	•Accuracy • Precision
Y Luo [115] 2021	ML (Fuzzing)	Detection	N/A	N/A	•Extensibility •Interactivity	•Need more measurements	•Payload ratio •Vulnerability discovery ability
Zhang et al [121] 2019	DL (DBN)	Detection	N/A	N/A	•Detect attacks traffic in real time	•Need to add more measurements	•Accuracy
Tang et al [119] 2018	DL (MLP and LSTM)	Detection	N/A	N/A	•High predictive capacity •High detection accuracy	•Lacks comparison	•Accuracy • Precision • Recall •Predictive capacity over 99%
Li et al [120] 2019	DL (LSTM)	Detection	N/A	N/A	•High F1-score •Solves the overfitting problem	•Lacks comparison	•Accuracy •Precision •Recall •F1-score 99.58%
Qi Li et al [120] 2019	DL (LSTM)	Detection	Real web app	N/A	•Detects in complex high-dimensional massive data •High accuracy	•Did not mention if it can work efficiently in traditional system environment	• Accuracy 91.53% • Precision •Recall

TABLE 3. *(Continued.)* **State-of-the-art of the research on Query Manipulation - SQL Injection.**

M. Li et al. [126] 2020	DL (QL-LSTM based on text & traffic features)	Detection	N/A	N/A	•High accuracy •High detection rate	•Is not tested in big dataset	•TP •Accuracy 97% •Detection rate >97%
Gandhi et al. [129] 2021	DL (CNN and BiLSTM)	Detection	N/A	N/A	•High accuracy	•Need to Increase dataset size	•Accuracy 96% •Precision •Recall
Wen et al [130] 2021	DL (BiLSTM)	Detection	N/A	N/A	•Good comparison with ML as well DL	•Need to increase dataset with other injection types	•Precision, •Accuracy, •F1-score •Recall
Farea et al. [131] 2021	DL (BiLSTM)	Detection	N/A	N/A	•Uses multi measurements •Good compassion with other algorithms	•Need to check with balanced dataset	•TP, TN, FP, FN •Accuracy •Precision •F1-score •Recall
Tang et al [125] 2020	DL (MLP and LSTM)	Detection	N/A	N/A	•High accuracy results	•Need to train more algorithms and compare them	•Accuracy over 99% •Recall •Precision
Jothi et al. [128] 2021	DL (MLP)	Detection	N/A	N/A	•Scalable •Detect new injection pattern •High accuracy	•Need more payloads to be assessed well	•Accuracy 98% •Precision •Recall
Chen et al. [127] 2021	DL (CNN and MLP)	Detection	N/A	N/A	•Detect some 0-day attacks •High accuracy	•Cannot detect second-order SQL Injection	•TP, TN, FP and FN •Accuracy 98%
Xie et al [122] 2019	DL (Elastic-Pooling CNN)	Detection	N/A	N/A	•Can identify new attacks	•Need more log data	•TP; TN; FP; FN
Zhang et al [132] 2022	DL (Deep neural network)	Detection	N/A	N/A	•No need to manually extract features	•Lacks comparison	•Accuracy •Precision •F1-score •Recall
Falor et al. [133] 2022	DL	Detection	N/A	N/A	•Exhaustive dataset that includes all SQLI types	•Lacks comparison	•Accuracy •Precision •Recall
Li et al [134] 2019	DL (AdaBoost based on DF model)	Detection	N/A	N/A	•Solves degraded problem that faces deep forests	•Needs to compare with larger dataset	•Accuracy •Precision •Recall •F1-score
M Liu et al. [124] 2020	DL (deep natural language processing)	Detection	Real web app	N/A	•Fast running •Better accuracy •More SQLI are identified by using a less # of test cases	• Does not handle other vulnerabilities, e.g., XSS. •Few metrics are tested	•No. of vulnerabilities •No. of reductions
Xiao et al [135] 2017	Analysis user behavior	Detection and prevention	Custom web app	Java	•Occupying less system resources	•More false alarm	•TP; FP; FN

TABLE 3. *(Continued.)* State-of-the-art of the research on Query Manipulation - SQL Injection.

Qi and Dai [136] 2020	Sequence alignment algorithm	Detection	Real web app	N/A	•Minimum response delay •Superb accuracy	•Need to check more types & payloads of SQLI	•Accuracy 100% •Response delay
H y b r i d T e c h n i q u e s							
Li et al [83] 2015	Pattern matching (No sanitization)	Detection	Real web app	N/A	• High accuracy	•Needs more evaluation with more metrics	•Accuracy
Jahanshahi et al. [99] 2020	Pattern matching (Sanitization)	Detection and prevention	Real web app	PHP	•Does not require any modification to the web app.	•Does not handle functions and class definitions inside eval function	• TP and FP
Singh et al [137] 2015	Data mining clustering technique	Detection	Custom web app	N/A	• Detects using audit record • High accuracy	• Detection requires more time and high resources	• TP • Accuracy 100%

[1]TP=true positive; TN true negative; FP=false positive; FN=false negative; ML=machine learning; SVM=Support Vector Machine; DL = deep learning; Deep Belief Network (DBN); LSTM=Long Short-Term Memory; BiLSTM=bidirectional LSTM; MLP=multilayer perceptron; CNN=Convolutional Neural Network; OOP =Object oriented programming; TFIDF=Term Frequency-Inverse Document Frequency; DF=deep forest

called AdaBoost based on structure of deep forest model. In the training stage, their algorithm assigns different features with different weights based on their influence on the results and uses error rate to update the weights of features on each layer. They claimed that their algorithm shows better performance results than ML and DL.

Other researchers work dynamically on different aspects, such as DeepSQLI tool of M. Liu et al. [122] is based on deep natural language processing, and shows better results than SQLMAP. Likewise in a dynamic fashion, Xiao et al. [133] model is based on analyzing user behavior such as numbers of requests sent, length of SQL character inserted in the request and conducted operation in database. When behavior is recorded as malicious, the attacker will be added to the block-list, hence banning attacker from accessing web application. Also dynamically, Qi and Dai [134] presented a method to detect SQL Injection attacks based on an alignment algorithm. Operating on a different dimension using both static and dynamic analysis, Singh et al. [135] detects unauthorized users utilizing auditing database records.

Table 3 summarizes the state-of-the-art of the SQL Injection techniques, and manifests their pros and drawbacks, together with target application, languages and metrics used.

2) XPath INJECTION (XPathI)

XPath stands for XML Path Language which is a language that is used to query the Extensible Markup Language (XML) file like SQL for Database Management System (DBMS). XML files are usually used to store configuration related to the application or to store user data such as information related to authentication page, viz. roles, credentials and privileges. XPath injection is similar to SQL Injection scenario and when used without sanitization can allow unauthorized users to supply and manipulate query that is used to access and read stored confidential data. However, XPath language unlike SQL does not support comment character, so attackers need to build complete and successful malicious query to run their exploits. Figure 8 shows PHP authentication code that uses XPath language which is vulnerable to XPathI. Lines 2 and 3 receive user supplied data, whilst line 4 builds XPath query from the users input that is stored in lines 2 and 3 variables. Therefore, malicious user can send malformed payload in the username field line 2, for example:

```
1 $xml = simplexml_load_file("addresses.xml");
2 $user = $_POST['user'];
3 $pass = $_POST['password'];
4 $query = "//addresses[susername/text()='".$user."' and
    password/text()='".$pass."']/creditCard/text()";
5 $result = $xml->xpath($query);
```

FIGURE 8. PHP login script vulnerable to XPathI [20].

admin' or 2=2 or 'b'='c.

Then this malicious payload will build Xpath query that will allow attackers to access admin profile without need to provide password value:

//addresses[susername/text()='admin' or 2=2 or 'b'='c' and password/text()='']/creditCard/text()

This code is vulnerable to XPathI because there is no parameterized XPath or sanitization in user supplied data. Therefore to prevent XPathI the following meta characters: ( ) = ' : [ ] , / . * need to be prevented.

State-of-the-art research of XPath Injection Solutions

Table 4 summarizes the XPath injection solutions in the literature. Few research [136], [137], [138], [139] is found regarding XPath. Researchers [136] and [137] both use static

analysis. Thomé et al. [136] proposed a method to reduce the slice that is extracted by a tool in source code analysis. The authors reduce slice length to only extract valuable and relevant information needed to predict if there was a vulnerability or not in the sink. However, their method only enhances slice but not verify vulnerabilities. Later, Thomé et al extended this work in [137] where they suggested a tool called JoanAudit that works by data flow analysis to detect and fix common injection vulnerabilities such as XSS, SQLI, XMLi, XPathI, and LDAPi in Java web system. This tool slices sensitive lines and sink of code that needs to audit regarding security checks. Then the code is followed from source to sink using context analysis and vulnerable sink that missed security or validation functions are reported. This tool has many advantages such as detect vulnerabilities in early stage, security auditor only needs 1% to inspect source code manually as well as locate which line of source code has vulnerability. However, it periodically needs update when any new source or sink are released and add them to the configuration file. Furthermore, it also needs comparison with prevailing tools to measure its effectiveness properly.

On the other hand, [138] and [139] based their solutions on dynamic analysis. Specifically, Clincy and Shahriar [138] developed IDS by leverage generic algorithm to protect web services from common web attack such as XPath injection, XML bomb, remote file inclusion and SQL Injection. This IDS is signature-based and work by analyzing SOAP messages against attack signature. It has the ability to generate new attack signature and detect complex and simple forms of variety injection attacks. However, it needs to be compared with other IDS types such as anomaly based and increase datasets for best evaluation

Deshpande and Kulkarni [139] worked from a different perception using neural network to classify and identify user input behavior for three groups valid, invalid and malicious. After identification, if the user input is classified as valid, then access to legitimate file is allowed, otherwise if the input is classified as a malicious request, then user is redirected to other counterfeit resources instead of original one. Finally, if it was classified as invalid will provide a custom error message. Their experimental results shows accuracy value over 90% in classification of input vectors, misleading attacker and redirect him to fake resources and custom error messages and access normal input if it contains special characters.

3) LDAP INJECTION

Lightweight Directory Access Protocol (LDAP) is an internet open protocol used to store and retrieve data from a hierarchical structure called LDAP information tree. It works like computer's folders that stores data inside directory in a hierarchical way. It is commonly used in organizations for authentication/authorization as well as when releasing web applications. Hence, web application can integrate with LDAP server. A malicious user could query LDAP server

[1]OCL =Object Constraint Language.

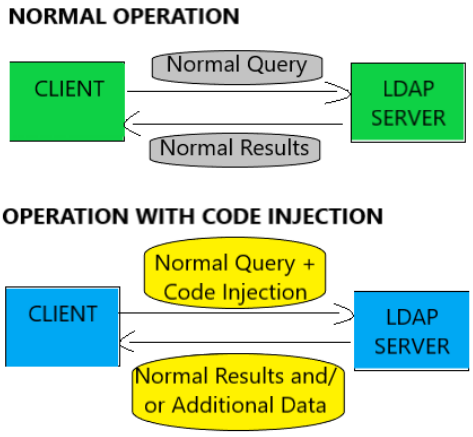

FIGURE 9. LDAPI attack [143].

```
1 $ds = ldap_connect("ldap.server.com");
2 $r = ldap_bind($ds);
3 $dn = "ou=Bank foo of city XXX,o=Bank foo,c=PT";
4 $user = $_POST['user'];
5 $pass = $_POST['password'];
6 $filter = "(&(username=$user)(password=$pass))";
7 $fields = array("userID", "name", "mail", "creditCard");
8 $result = ldap_search($ds, $dn, $filter, $fields);
```

FIGURE 10. PHP login code vulnerable to LDAPI [20].

with meta-characters in an unexpected way and retrieve potentially sensitive data from LDAP directory if there is no sanitization or validation and this is called LDAPI. This is illustrated in Figure 9, where a normal operation versus another operation containing code injection is presented. This attack is conducted on the server side and works by manipulating LDAP query logic such as SQLI and XPathI. However, LDAP does not support comment character like SQL. Therefore, attackers need to insert query that will cause intended filter to be ignored.

Figure 10 presents authentication script that is vulnerable to LDAPI. Lines 1 and 2 are used to connect web application with LDAP server. Line 6 uses filter to authenticate users login info that is supplied in lines 4 and 5.

If a malicious user enters as username *Alice)(&))* and as password any Password, he will cause unintended behavior and bypass login validation by accessing Alice profile without sending correct password. Therefore, the resulting filter will be *(&(username=Bob)(&))*. Consequently, the password value will be ignored and substituted by & character. Therefore to prevent LDAPI, a validation in user input is required and these characters ( ) ;, * | & = need to be blocked.

State-of-the-art research of LDAP Injection Solutions

*To our knowledge, very few approaches [72], [141], [142] solving LDAP injection were found, as presented in Table 5. In particular, Jawalkar et al. [142] utilized a hybrid approach using both static and dynamic technique to test web application vulnerabilities. Their approach focuses on Java language specially input injection vulnerabilities such as SQL

TABLE 4. State-of-the-art research on Query Manipulation - XPath Injection.

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
Static Techniques							
Thomé et al [138] 2015	Pattern matching	Detection	Real web app	N/A	•Security slice is 80% smaller than previous works	•Do not verify vulnerabilities	•Accuracy
Thomé et al. [139] 2017	Data flow analysis	Detection (fix in some cases)	Custom web app	Java	•Security auditor only needs 1% to inspect source code manually •Detects XSS, SQLI, XMLi, XPathI, and LDAPi	•Need to periodically update configuration file •Lacks comparison	•Precision •Accuracy •Scalability
Dynamic Techniques							
Clincy and Shahriar [140] 2017	Pattern matching (Signature based IDS)	Detection	Custom web service	N/A	•Generates new attack signatures •Detect simple and complex forms of injection attack	•Lacks comparison •Use small dataset	•Accuracy
Deshpande & Kulkarni [141] 2019	Neural Networks	Detection and prevention	Custom web service	N/A	•Misleading attackers and redirect to fake resources	•Accuracy value is not very high	•TP;FN;TN; FP •Response time •Accuracy 90%

TABLE 5. State-of-the-art research on Query Manipulation - LDAP Injection.

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
Static Techniques							
P. Bulusu 2015 [142]	OCL fault-injection	Detection	Custom web app	PHP	•High detection rate	•Does not extract design level information from source code	Mutation Score
Shahriar et al. [143] 2016	OCL Fault Injection	Detection	Real web app	N/A	•Less FP and FN	•Need to implement it automatically	•TP; TN
Medeiros et al [73] 2016	Data flow analysis	Detect (and correct)	Real web app	PHP and WordPress	•Solves extensibility issues in WAP •Detects SQLI in WordPress •Detects NoSQL and comment spamming injection •Less FP than WAP	•Attributes need to update regularly •Need knowledge to define entry points and sinks for new class of vulnerabilities	TP; FP
Hybrid Techniques							
Jawalkar et al. [144] 2015	Findbug and Tomcat server	Detection	Custom web app	Java	•High confidence results •Less FP	Produce more FN	TP; TN

Injection, XSS, LDAP injection, XML injection, HTTP verb tampering and HTTP parameter pollution. Their approach first uses Findbug tool to perform static analysis and reports vulnerabilities, followed by dynamic analysis using Tomcat server to test reported vulnerabilities. Dynamic analysis also will report vulnerabilities, but final report only contains vulnerabilities that are reported in both static and dynamic analysis. Therefore, it will produce a final report with high confidence results and less false positives. However, this will produce more false negatives because the vulnerabilities if are not discovered by dynamic analysis will not be reported in the final report and some vulnerabilities will not be discovered by

dynamic analysis or may be blind and need further work to detect.

On the other hand, researches [72], [140], [141] use static techniques. In particular, P. Bulusu [140] applied OCL fault-injection based testing approach for detecting LDAP injection in source code. It first identifies LDAP function used in source code and analyses it by applying fault adequate test case generation algorithm. It has has high detection rate, yet it does not extract design level information from source code

Working manually, Shahriar et al. [141] proposed a new method to detect LDAP injection manually. Their method works by reading source code and identifying lines that are related to LDAP query and building a flow chart for the function they want to test and expressing various paths related to make injection succeed. This technique has less false positives and negatives yet needs to be automated.

From a different perspective using data mining techniques to classify the false positives, Medeiros et al. [72] developed a WAP tool by introducing WAPe (extensions). Extensions are weapons that solved extensibility issues and allow WAPe to detect 15 classes of vulnerabilities. As well user can configure it to handle new classes of vulnerabilities without programming knowledge by defining entry points, sensitive sinks and sanitization functions. It is an open source and first static tool to detect NoSQL and comment spamming injection and programmed with new weapon that handle $wpdb class to detect SQLI in WordPress. In addition, it detects LDAP injection. Moreover, WAPe produces less false positives than WAP because it is developed with more symptoms aka attributes that handle false positives as well as dealing with dynamic attributes such as symptoms defined by a user. However, a user does not have enough knowledge to define most entry points, sensitive sinks and sanitization functions to detect new classes of vulnerabilities. Moreover, attributes need to be updated regularly to help predicting FPs.

4) NoSQL INJECTION

NoSQL is an approach to untraditional, unstructured database that uses to store and retrieve data without using SQL query. NoSQL databases are commonly used with large-scale web applications. There are varieties of NoSQL databases that are created for specific needs. MongoDB is the most common used one that implements document store model, similar to key-value model but in the value it stores all information in the document. MongoDB uses Javascript object notation (JSON) format to execute queries, which is well-defined and natively implemented with most web application languages.

Figure 11 shows login code vulnerable to NoSQL Injection. In lines 4 and 5, two variables receive user input then embed them into MongoDB query to lookup user in its database without using validation or sanitization. Therefore, this code is vulnerable and could allow attacker to bypass login page if he supplies malicious payload such as in username field: *user=administrator* and in the password field: *password= [$ne]=1*. This payload will compile in this array:

```
1 $conn = new MongoClient("mongodb.server.com");
2 $db = $conn->selectDB('foo');
3 $collection = new MongoCollection($db, 'users');
4 $user = $_POST['user'];
5 $pass = $_POST['password'];
6 $query = array("username" => $user, "password" => $pass);
7 // line 9 does the following codification implicitly:
8 // $query = "{username: '" + $user + "', password: '" + $pass + "'}";
9 $result = $collection->find($query);
```

FIGURE 11. PHP login code vulnerable to NoSQL Injection [20].

array("username" => "administrator", "password" => array("$ne" = > 1)) and encoded in JSON format to execute at MongoDB:

username: 'administrator', password: $ne: 1.

Therefore, break intended logic when send $ne that means not equal and allow to access administrator without supplying the correct password. To secure code from this vulnerability, a proper validation by checking user input is required to escape from these characters: <> & ; / : ' * as well as sanitization by using mysql_real_escape_string function, which will escape from malicious characters as SQLI.

State-of-the-art research of NoSQL Injection Solutions

It is an injection vulnerabilities that appeared in 2013 mentioned in OWASP report under injection attack. Table 6 recaps the solutions investigated in this category, where all use dynamic techniques.

Ma et al. [144] detection approach, called dynamic NOSQL injection attacks detection (DND), relies on parse tree. DND focuses on MongoDB database. When receiving HTTP request from the client, a parse tree is built according to the client's request and compared with the old record of parse tree. If they are equal, that means no NoSQL Injection is detected otherwise injection is detected and stored as text in repository. DND approach shows less false positives, fewer response time and high accuracy rates, but only detects injection in MongoDB database.

Researches [145], [146], [147] use pattern matching techniques in detecting NoSQL Injection. For instance, Joseph and Jevitha [145] proposed a dynamic solution based on regular expression to detect and prevent injection attack in MongoDB when deployed with Java language. Their solution works by matching user input and decide if it contains injection or not. If entered input flagged as invalid the query will not proceed to the database even if it was not attack intended. They claim there is no false negatives in their solution but this solution only detects blind-based boolean and time-based NoSQL attacks. Moreover, some not intended attacks input will be rejected. Eassa et al. [146] present independent RESTful web service solution named DNIARS to detect and prevent NoSQL Injection attack. DNIARS is built using PHP language and has the ability to response to variety of request formats viz. XML and JSON. It works by checking if there is no injection, then returns 200 status code and continue executing the NoSQL query. Otherwise if injection is detected,

it will return 400 status code and display error message as well as stop executing NoSQL query. DNIARS tool has less error rate but the request needs more time because another request will be generated from web server to web service DNIARS tools. Furthermore, if DNIARS web service is down there will not be any detection. A different approach for detecting NoSQL Injections using supervised learning technique was suggested by Rafid et al. [147] by creating two categories benign and injection. Their tool detects injection of both MongoDB and CouchDB but can extend to other NoSQL databases with minor modifications. Their tool works as server plugin, which automatically opens a port and listen in the web server. Then, it intercept requests when there is a communication between a web server and NoSQL database. After interception, it only sends benign query to the back-end NoSQL database. This tool shows better results in detecting NoSQL Injection, and has the ability to detect most NoSQL Injection attacks, but needs to increase its dataset.

B. CLIENT-SIDE INJECTION

This category differs from the others in that here the target is users not the application itself, through executing malicious client-side script (e.g., JavaScript) in the victim's browser. There are three vulnerability classes, namely cross-site scripting (XSS), header injection (HI) and email injection (EI), which are hereafter explained.

1) XSS INJECTION

Cross-Site Script (XSS) is one of the most vulnerabilities that affects web applications and websites because XSS occur in environment that can parse and understand Javascript. Additionally, it is considered the most reported vulnerability in hacker one bug bounty platform. It is considered a type of injection attack because it occurs when an attacker injects malicious code in the web application and sends it to other users. This flaw is succeeding because it is quite widespread in most web applications and occurs when web application utilizes users input and generate output based on it without encoding or validation [148], [149]. When exploiting XSS, attacker can inject JavaScript code that allows to steal user cookies, session or redirect users into fishing web sites as displayed in Figure 12.

XSS can be classified into three types:

- *Reflected XSS*: is non-persistent. Attacker needs to send malformed URL with injected Java-script code to other users.
- *Stored XSS*: is persistent and attackers inject malicious JavaScript code in the database. Therefore, there is no need to send malformed URL or to interact with the victim where users request web application page that retrieves its content from the injected database.
- *DOM Based XSS*: is a less well-known kind of XSS and occurs in the DOM environment not in the response code from the server.

To prevent XSS attack, a sanitization is required by using function such as htmlentities() and/or by encoding output.

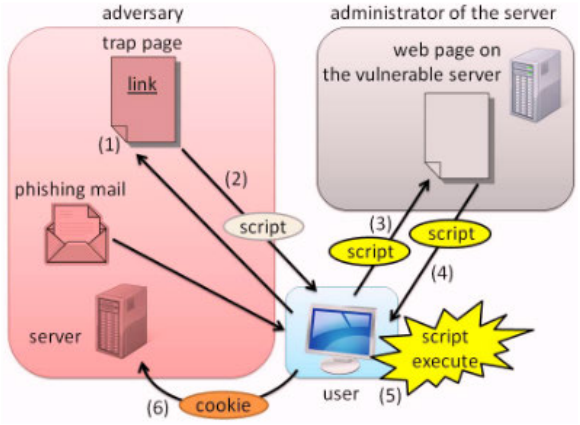

FIGURE 12. Stored XSS attack to steal user cookies [150].

Encoding techniques can prevent metacharacters such as $<$ *and* $>$ to interpret as HTML code instead of using it as normal character.

State-of-the-art research of XSS Solutions

Researchers [151], [152], [153], [154], [155], [156] use ML concept to detect XSS attack, as synopsizes in Table 7. For instance, Rathore et al. [151] proffered an approach to detect XSS attack on social network services (SNS). Ten different classifiers are used to classify webpages into XSS or not-XSS. This approach shows better accuracy and lowest FP in the SNS environment. However, its dataset need to be updated regularly to work better. On the other hand, Banerjee et al. [152] detect modus operandi of XSS attack via two features: URLs and JavaScript. They use four ML algorithms (SVM, KNN, Random forest and Logistic Regression), and hence classifying webpages as malicious or benign. They inferred that the Random Forest Classifier was the most accurate having the lowest false positive rate of 0.34. As well Mereani et al. [153] investigate SVM, KNN and Random Forests and achieved high accuracy and precision. Whilst Gogoi et al. [154] also compare ML techniques in detecting XSS attacks. Specifically, researchers [155], [156], [157] propounded other solutions using genetic algorithm. Gupta et al. [155] further compare their classification accuracy with NB, random forest (RF), logistic regression (LR), SVM, AdaBoost, and MLP. They achieve high accuracy of 98.5%. Whilst research [157] utilized genetic algorithm-based fuzzing scheme to sequence the attack vectors into genes, which are then repeatedly optimized using the grammatical structure features of XSS together with common bypass methods. They profess high precision and accuracy rate. However, their parallel detection performance may be further optimized and improved. Lu et al. [156], on the other hand, detects XSS attacks using a fusion verification method that amalgamates traffic detection with XSS payload detection. Their experiments have an increase in accuracy by 3.81%, in recall rate by 48%, and in F1-score by 27.94%.

Researchers [158], [159] work on both ML and DL in detecting XSS attacks. Using a fuzzing-based approach,

research [158] realized a black & white attack that enhances the confidence coefficient of malicious samples. Their approach is an adversarial attack model based on Soft Q-learning, which has an escape rate of over 85%. On the other hand, Zhou et al [159] proffered an ensemble learning approach to detect XSS attacks. Sorting the nodes, using the Bayesian network, could aid in real time attack detection. Howbeit, they need more testing to verify their method. Other researchers [160], [161], [162], [163], [164], [165], [166], [167], [168] present DL XSS attacks detection approaches. In particular, [161], [162], [163], [164] use LSTM in their detection methods. Specifically, the approach of Lei et al. [161] is based on the attention mechanism of LSTM recurrent neural network. They preprocess the data then utilize word2vec to extract XSS payload features and finally map them to feature vectors. The LSTM-Attention detection model, which is an improvement to the LSTM model, was developed to train and test the data. The context-related features for DL are extracted using LSTM, then the added attention mechanism is utilized to extract more effective features. Their model achieves a precision rate of 99.3% and a recall rate of 98.2%.

From a different aspect, to defend embedded devices deployed in intelligent IoT system against XSS attack, research work of [162] uses a fog-enabled approach that detects by comparing injected strings with the block-listed attack vectors. Further, they prevent by utilizing an optimized filtering method. They claim high accuracy up to 90%. On the other hand, Yong et al. [163] use LSTM RNN to train and test the detection model, achieving a precision rate of 99.5% and a recall rate of 97.9.%. Whilst [164] combined CNN with LSTM after decoding, generalizing and tokanizing, then next utilizing word2vec to change words into word vectors. Their method achieves excellent accuracy of 99.3%.

Researchers [165] and [166] use MLP in their methods. Research [165] detects XSS using a robust ANN-based MLP scheme, using a large real-world dataset. They achieve high accuracy, detection rate and AUC-ROC while maintaining low FP rate. Whereas, research [166] use MLP DL model in five phases namely extraction, feature engineering, datasets generation, then DL modeling, and classification filtering. Their experiment shows high accuracy of 99.47%.

Utilizing neural networks, [160] and [167] proffered solutions for XSS attacks. Research [160] utilized Convolutional Deep Neural Network (CDNN) in preprocessing, then they use noise filtering to encode and train the CDNN for removing SQL and XSS special symbols. Their method has a reduced processing time. Whereas, researchers [167] use Convolutional Gated-Recurrenr-Unit (CGRU) neural network. Instead of a pooling layer, a gate-recurrent unit is used to do feature acquisition on the time dimension, yielding high-accuracy multicategory results above 99.6%.

Other researchers [168] work differently using DL GAN technique to optimize the detection of XSS attacks. Their model is enhanced using Monte Carlo tree search (MCTS) algorithm, which is utilized to produce the adversarial model for training and testing.

Same authors Gupta et al. [169], [170], [171] work on mobile cloud computing application, virtual cloud server, and HTML scripting. Specifically, the work of [171] utilizes context-sensitive sanitization with HTTP requests. Research [169] uses PHP web applications with BlogIT, whilst the method of [170] works on virtual cloud server based on HTTP requests.

From a different facet, instead of using black box, Antonin et al. [172] uses gray box. Their method utilizes HTML output with HTTP request based context-sensitive XSS flaws.

2) HEADER AND EMAIL INJECTION

Header Injection (HI) occurs when web application receives input from a user without security check and includes that input in the HTTP response headers. This allows an attacker to break the normal response and inject it with the new line (\n) and carriage return (\r) characters to execute attacks such as response splitting, Cross-Site Scripting, session fixation and malicious redirection. To prevent this attack, it is highly recommended to avoid returning user input into HTTP headers or by sanitizing user input for line-feed characters and carriage-return.

Email injection (EI) is similar to HI and occurs when there is vulnerable web contact forms that allow users to send email. The malicious user can inject vulnerable forms with line termination characters, which allow attacker to manipulate email components (such as content, sender, receiver, etc.). Protection against this vulnerability is alike in HI.

State-of-the-art research of Header and Email Injection Solutions

Table 8 sums up the Header and Email injection solutions in the literature. Researchers Medeiros et al have two researches [173] and [174]. For instance, research [174] built a new tool called WAP to detect as well as go further by correcting vulnerabilities in source code. Their tool works by combining different approaches specifically Taint Analysis (human coded knowledge) with data mining supervised technique (automatically get knowledge). WAP is implemented in three phases. First, it performs taint analysis to flag vulnerabilities. Second, each candidate vulnerability enters data mining process to classify which one is a real vulnerability and which one is an FP. In the last phase, after being confirmed as real vulnerabilities, they will enter a correction step by adding proper code that will eliminate vulnerabilities i.e., validation and sanitization functions. It performs global analysis by scanning connected modules not only current file, and shows better accuracy and precision results when compared with Pixy as well as PhpMinerII tools. Actually, accuracy 45% better than Pixy's and 5% better than PhpMinerII. One more benefit that it can handle eight classes of input validation vulnerabilities. However, attributes used in data mining to classify FPs from real vulnerabilities need to be updated regularly because there is always new ones created

TABLE 6. State-of-the-art research on **Query Manipulation - NoSQL Injection - Dynamic techniques.**

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
H. Ma et al [146] 2017	Parse tree and HTTP request	Detection	Custom web app	N/A	•Less FP •Less response time •High accuracy	•Only detect injection in MongoDB	•Accuracy 100% •FP
Joseph & Jevitha [147] 2015	Pattern matching (Regular expression)	Detection and Prevention	Real web app	Java	Claims no FN	•Some not intended attack input will be rejected.	TP; FN
Eassa [148] 2018	Pattern matching (Sanitization web service tool)	Detection and prevention	Custom web service	N/A	•Less error rate •Supports different request format e.g. XML and JSON	•If DNIARS is down there will be no detection. •More time	N/A
Islam et al. [149] 2019	Pattern matching (Sanitization)	Detection and prevention	Custom web app	N/A	•High accuracy •Extendable to other NoSQL DB	Needs to increase dataset	•Accuracy 91.8% •Precision •Recall

and evolved. Moreover, WAP is only configured with static attributes not dynamic one such as user defined function, moreover source code used in WAP tool is hard to extend to new classes of vulnerabilities. Another static approach suggested by the same authors [173] inspired in natural language processing NLP, which makes static tools learn to detect vulnerabilities automatically without programing knowledge about how each vulnerability is discovered. They implemented this concept in a tool called DEKANT, which uses sequence model hidden Markov model (HMM) for learning to characterize vulnerabilities from a corpus (group of instructions 'not slices' converted to ISL) as vulnerable or not. ISL is an intermediate slicing language produced by the authors that translates source code into tokens to represent code. It is a crucial part in the approach, where it stores info about which group of instructions lead to vulnerabilities. Then utilizing this knowledge (namely which instructions may lead to a flaw) with remaining slices to classify them. Sequence model is differing from standard classifiers where it takes the order of source code elements and relation between them into consideration to reduce FPs.

Working dynamically from a different perspective, Chandramouli et al. [175] developed a tool for detecting email header injection through fuzzing web application to find email form then sending request to test it by predefined payloads and based on the response can determine if it vulnerable or not. Their tool detect email header injection in many programing language, nonetheless can not test email header injection if CAPTCHAs is used.

C. INFORMATION DISCLOSURE

This last proposed category considers vulnerabilities dealing with access to URL locations and unintended files to access or disclose/read operating system file from application file inclusion feature. The following vulnerabilities are related to this category: local file inclusion (LFI), remote file inclusion (RFI), Path Traversal (PT), Information leaks (IL), Directory Listing (DR) and Predictable Resource Location (PRL).

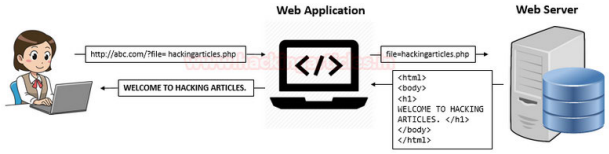

FIGURE 13. File inclusion feature [176].

1) FILE INCLUSION (FI)

There are few researches conducted on detecting FI vulnerability. It has two types local and remote inclusion.

- *Local File Inclusion(LFI)*
  * It is a vulnerability that allows an attacker to exploit the feature of file inclusion in web application through reading files from the server, as shown in Figure 13. LFI attack can lead to read sensitive files, configuration, XSS or even remote code execution (RCE). To secure from LFI, a developer needs to explicitly build white list for required files or use *if statements* besides not allowing users to control file include feature.
- *Remote File Inclusion (RFI)*
  RFI is similar to LFI, but differs in the required files. In the RFI, the web application requests files from remote location not in the local host. Exploiting RFI can lead to the same risks that are caused by LFI. Figure 14 displays RFI attack where an attacker controls file parameter via URL to retrieve and execute malicious payload from the server that he/she controls. Fortunately, most current PHP installation configured with *allow_url_include* feature to off, which prevents

TABLE 7. State-of-the-art research on **Client-side - XSS injection.**

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
S t a t i c T e c h n i q u e s							
Abaimov et al. [162] 2019	CDNN	Detection	Custom web app	N/A	•Reduced processing time	•Lacks comparison	•Accuracy 95% •Precision 99% •Recall 92%
D y n a m i c T e c h n i q u e s							
Rathore et al [153] 2017	ML	Detection	Custom web app	N/A	• High accuracy •Uses numerous ML classifier	•Dataset needs to be updated regularly	•Recall •Precision •Accuracy 97% •FP
Banerjee et al [154] 2020	ML (SVM, KNN, Random forest, & Logistic Regression)	Detection	Custom web app	Java	• High accuracy •Uses numerous ML classifier •Lowest false positive rate of 0.34	• No prevention scheme is suggested	•Precision •Accuracy •FP
Mereani et al. [155] 2018	ML (SVM, k-NN and Random Forests)	Detection	Custom web app	Java	•High accuracy and precision	•Lacks comparison	• Accuracy of 99.75% •Precision 99.88% • Specificity • Sensitivity
Gogoi et al. [156] 2021	ML (SVM)	Detection	Custom web app	N/A	•High precision, recall and F1-score	•Lacks comparison • Needs more evaluations	•Precision 99.88% • Recall • F1-score
Gupta et al. [157] 2022	ML (Genetic algorithm)	Detection	Custom web app	N/A	•High accuracy, precision and F1-score	•Does not detect other injection attacks	•Accuracy 98.5% •Precision • F1-score • Specificity • Sensitivity
Lu et al. [158] 2022	ML (Genetic algorithm)	Detection	Custom web app	N/A	•Increase in accuracy, recall rate and F1-score	•Cost of decreasing FN rate low incurs an increase in FP rate	•Accuracy 98.5% •Precision • F1-score • Specificity • Sensitivity
Liu et al [159] 2022	ML (Genetic algorithm)	Detection	Custom web app	Python	•Automatic detection • Better detection without a large # of test dictionaries •Reasonable time • High precision and accuracy	• Parallel detection performance may be optimized and improved	•Precision rate of 1.0 •Recall • Accuracy rate > 0.98 •Detection time •FP
Wang et al [160] 2022	ML/DL	Detection	Custom web app	N/A	• High escape rate	• Needs more evaluations	•Escape rate of over 85%
Zhou et al [161] 2019	ML/DL	Detection	Custom web app	N/A	• Sorting the nodes helps the attack detection in time	• Needs more evaluations	•Detection rate
Lei et al [163] 2020	DL (LSTM)	Detection	Custom web app	N/A	• High precision •High recall rate	• No prevention scheme is suggested	•Precision $\approx$ 99.3% •Recall $\approx$ 98.2%
Chaudhary et al [164] 2022	DL (LSTM)	Detection and prevention	Custom web app	N/A	• High accuracy	• Needs more evaluations	•Accuracy 90%

TABLE 7. *(Continued.)* **State-of-the-art research on Client-side - XSS injection.**

Yong et al. [165]	DL (LSTM and RNN)	Detection	Custom web app	N/A	•High precision and recall rates	• Lacks comparison •Needs more evaluations	•Precision 99.5% •Recall 97.9%
Kadhim et al. [166] 2021	DL (CNN and LSTM)	Detection	Custom web app	Python	•High accuracy • Low false positive rate	• Not applied in real-time system	• Accuracy 99.3 •Precision 99.9% •Recall 99.1% •F-score 99.5% •AUC-ROC 0.95%
Mokbal et al. [167] 2019	DL (ANN-based MLP)	Detection	Custom web app	Java	•High accuracy, detection rate and AUC-ROC • Low FP rate • Low complexity	• Not applied in real-time system	• Accuracy 99.32 •Precision 0.993 •Recall 98.35% •F-score 0.9932 •FP rate 0.3% •AUC-ROC 99.02%
Odun-Ayo et al. [168] 2021	DL (MLP)	Detection and prevention	Custom web app	Java	•Real-time detection •High accuracy	• Does not prevent other web-based attacks	•Accuracy 99.47% •Precision •Recall •F-score •AUC-ROC
Yang et al. [169] 2019	CGRU neural network	Detection	Custom web app	Python	•Efficient method	•Memory consumption needs reduction •Needs online update	•Accuracy 99.6% •Precision •Recall • F1-score
Zhang et al. [170] 2020	DL (GAN)	Detection	Custom web app	N/A	•Improved DR	•Suits only adversarial cases	•Precision •Recall
Gupta et al. [171] 2016	PHP web app with BlogIT	Detection	Real web app	PHP	• Acceptable runtime overhead	• FN needs improvement	•Precision 99.5% •Recall 97.9% •FN
Gupta et al. [172] 2016	Virtual cloud server based on HTTP	Detection	Real web app	PHP	•Low FP and FN	• Needs evaluations on more real world & OSN-based web apps	•Recall • FP;FN
Gupta et al. [173] 2018	Sanitization (context-sensitive)	Detection	Custom web app	Java	•Very low acceptable FP and FN	• Suffers from acceptable performance overhead	•Precision •Recall • F1-score • Response time • FP;FN
Antonin et al. [174] 2020	Gray box	Detection	Custom web app	N/A	• Intercept traffic to non-open source databases	• Needs more evaluations	•Severity Incorrect Sanitizations

malicious user from *include remote files*. Nonetheless, this prevents RFI but not LFI.

State-of-the-art research of File Inclusion Solutions

A prevention technique presented by Tajbakhsh and Bagherzadeh [178], called AntiLFIer, prevents local file inclusion vulnerability in PHP language. This framework is written in Java and work by only allowing to include PHP scripts that are locally located in the root folder or subfolder related to the web site. These PHP scripts are labeled as trusted files and other files are labeled as untrusted file. This framework only includes PHP web site code as well as trusted list encrypted by administrator private key. However, attacker can delete trusted list file if he/she gets access to the web site directory. On the other hand, Hassan et al [179] proffered a model to detect local file inclusion (LFI) vulnerability. This model is implemented using Python language. The tool is developed with five steps:

- *URL validation*: identify host status through checking HTTP response code if matched with 200 then host is up and forward to next step otherwise the host is considered

TABLE 8. State-of-the-art research on **Client-side - Header and Email Injection - Static techniques.**

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
S t a t i c T e c h n i q u e s							
Medeiros et al [175] 2016	ML (Hidden Markov Model)	Detection	Real web app	PHP	•High acc & precision • Automatic detection •Obtains knowledge without coding	•Produce FN because some slices are not collected	•Accuracy & Precision ≈ 96% • FP; FN
Medeiros et al [176] 2016	Data flow analysis with data mining	Detection and correction	Real web app	PHP	•Performs global analysis •Reduced FP •Better accuracy & precision than Pixy's & PhpMinerII	•Attributes need to be updates regularly •Hard to extend to new classes of vulnerabilities.	TP; TN; FP; FN •Accuracy
D y n a m i c T e c h n i q u e s							
Chandramouli et al. [177] 2018	Fuzzing web application	Detection	Real web app	N/A	• Detects email header injection in many programming language	•Can not detect email header injection if CAPTCHAs is used	•Detection time

[1]ANN=artificial neural network; MLP=multi-layer perceptron; CNN=Convolution Neural Network; CDNN =convolutional deep neural network; CGRU =Convolutional Gated-Recurrenr-Unit; GAN= Generative Adversarial Network; RNN=recurrent neural networks; NB=Naive Bayes, RF= random forest, LR= logistic regression, AdaBoost, MLP= multilayer perceptron; AUC ROC=Area Under the Curve of ROC

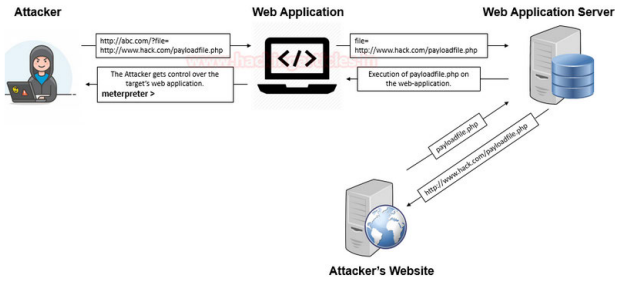

FIGURE 14. Exploit RFI and execute payload from attacker server [177].

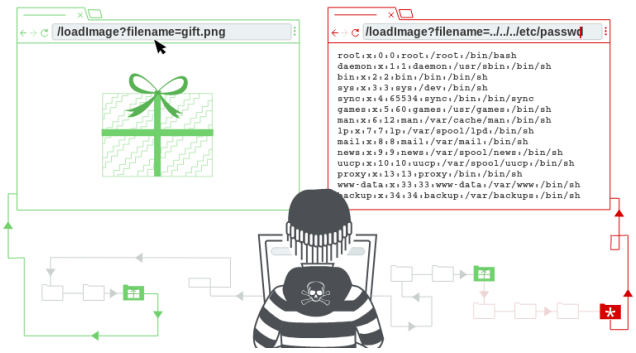

FIGURE 15. Manipulate filename variable to perform PT/DT attack [180].

down and the tool will display an error message "Host server is not available".

- *Crawling*: this step will send many requests to identify URL endpoint in the application. Then extract parameters from endpoint
- *Execution of the URLs*: sends LFI attack payload to crawled pages.
- *Collect and Matched Response*: receives response and matches it with predefined expressions to confirm if there is LFI or not.
- *Provide Output*: shows output result to the user.

This model has an advantage of comparing between manual and automated LFI detection and presents high accuracy value. However, it only detects LFI in $_GET method and also leaves many host without checking because it only considers host is up when receiving 200 status code. Therefore, it needs to dig dipper when receiving other meaningful status code such as 301, 302, 403, and 500.

These solutions are summarized in Table 9.

2) PATH TRAVERSAL (PT)

PT is an attack that works by manipulating sanitized variables that reference files aiming to access files/directories stored outside the web root folder. Figure 15 explains PT attack. When exploiting PT, an attacker can read arbitrary files such as sensitive operating system files, configuration and application code on the back-end system. This attack is also known as "directory traversal", "dot-dot-slash", "directory climbing" or "backtracking".

State-of-the-art research of Path traversal Solutions Flanders [181] presented new algorithm to prevent directory (DT) traversal attacks. This algorithm is written in C programming language and works by using both path string canonicalization and whitelisting technique to prevent DT. Their algorithm is easy to test, lightweight, easily extendable, cross-platform compatible as well as intuitive. However, they need to test this algorithm with more real application and compare

TABLE 9. State-of-the-art research on **Information disclosure - File Inclusion - Dynamic solutions.**

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
S t a t i c T e c h n i q u e s							
Tajbakhsh and Bagherzadeh [180] 2015	White list	Prevention	Real web app	PHP	•Only includes PHP web site code •Trusted list encrypted by administrator private key	Attacker can delete trusted list file if he got access to the web site directory	•TP
Hassan et al [181] 2018	Automated model using python	Detection	Real web app	Python	• Average accuracy •Compares between manual and automated LFI detection	•Only using 200 stats code to consider host is up •Only works with $_GET method	•Accuracy 88% •FP; FN

TABLE 10. State-of-the-art research on **Information disclosure - Path Traversal - Dynamic solution.**

Author/ Year	Technique	Aim	Target App	Prog. Lang.	Pros	Drawback	Metrics
S t a t i c T e c h n i q u e s							
Flanders [183] 2019	White listing technique with string canonicalization	Prevention	N/A	C programming language	•Easy to test •Lightweight •Easily extendable •Cross-platform compatible as well as intuitive	•Needs to be tested with more real apps •Lacks comparsion	N/A

it with more algorithms to measure its effectiveness properly. Their method is recapitulated in Table 10.

3) DIRECTORY INDEXING

Directory Indexing (also called Directory Listing or Browsing) occurs when user clicks URL to access web page but there is no index default page. Therefore, user can view besides download entire files located on the same directory/folder of the visited URL file. The danger of this misconfiguration allows attacker to read sensitive files such as database configuration or third-party services that assist attackers to escalate the attack. To protect from directory indexing the administrator of the web server has to configure web server properly, such as preventing viewing files located in the server when there is no index page.

4) INFORMATION LEAKS

This command occurs when web application reveals sensitive information, example a comment created by a developer in HTML, error messages or debug mod. This information may be critical such as credential, source code or unsecure endpoint, or not critical such as framework version or recommendations but can also help attackers to leverage attack and cause damage to the application. Therefore, revealed information needs to be restricted.

5) PREDICTABLE RESOURCE LOCATION

This attack is used to discover hidden function and content of the web application by sending guessing requests for unintended public content and brute force critical files and endpoint such as backup and configuration. These discovered files may disclose sensitive and critical information related to the web application database, passwords, admin panel, machine names that host web application and paths to other sensitive areas.

VI. EVALUATION METRICS

This section presents the most common metrics used to assess the performance of the reviewed suggested techniques. Every metric is represented by the number of its occurrence in the reviewed articles as depicted in graph 16. As can be seen from the graph, the most common evaluation metric as yet is *accuracy,* then *precision* and *DR or recall*, which were used by most of the total reviewed articles. Next in popularity comes FP and TP. In actual fact, other evaluation metrics viz. TN, FN and Fi-Score share great popularity amongst the reviewed techniques. However, others like response time, specificity, AUC-ROC, detection time, scalability, memory consumption, payload ratio, predictive capacity, no. of reductions, escape rate and mutation score were noticed with varying less popularity. Actually, regarding negative tests, researchers have to embed more to their metrics evaluations because TN shows real result of no vulnerability found. On the other hand, FN is most danger because it unveils no vulnerability in applications that really have.

Recalling the following metrics [125], [165]:

True Positive (TP): is the number of actual attacks that are correctly classified.

True Negative (TN): is the number of legitimate statements that are correctly classified as safe.

False Positive (FP): this is a false vulnerability reported that happens when the tool incorrectly reports vulnerability that does not exist.

False Negative (FN): this occurs when a scanner fails to recognize known vulnerability.

Accuracy is a measure of the number of correct predictions of the total number, as presented by equation 1 [165].

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{1}$$

Detection rate (DR) or Recall (also known as *Sensitivity*) is the ratio between the correctly detected attacks and all the actual attacks as shown by equation 2 [165]. This metric is also well known as True Positive Rate (TPR).

$$Detection\ rate\ (DR)\ or\ Recall = \frac{TP}{TP + FN} \tag{2}$$

The False Positive Rate (FPR), on the other hand, measures the ratio of false positives within the negative samples, which is presented by equation 3.

$$False\ Positive\ Rate\ or\ (FPR) = \frac{FP}{FP + TN} \tag{3}$$

Equation 4 shows *precision* (also known as Positive Predictive Value (PPV)) representing the proportion of predicted accuracy in the total number of predictions [165].

$$Precision\ (or\ Positive\ Predictive\ Value) = \frac{TP}{TP + FP} \tag{4}$$

The *Negative Prediction Value (NPV)* is presented by equation 5

$$Negative\ Prediction\ Value == \frac{TN}{TN + FN} \tag{5}$$

F1-Score is a measure combining both precision and recall as given by equation 6 [165].

$$F1 - Score = 2 \times \frac{Recall \times Precision}{Recall + Precision}. \tag{6}$$

Specificity, also known as *True Negative Rate (TNR)* is gauged using equation 7.

$$Specificity = \frac{TN}{TN + FP} \tag{7}$$

Misclassification Rate (Error Rate) is the number of false predictions of the total number, which is calculated as presented in equation 8 [165]:

$$Misclassification\ Rate = \frac{(FP + FN)}{(TP + TN + FP + FN)} \tag{8}$$

Area Under the Curve is gauged by equation 9 below [165]:

$$Area\ Under\ the\ Curve\ (AUC - ROC) = \frac{1}{2}\left(\frac{TP}{(TP + FN)} + \frac{TN}{(TN + FP)}\right) \tag{9}$$

Other metrics varying in popularity viz.:

Memory consumption, which reveals the memory (RAM) resource utilization [81].

Response time is the taken time in HTTP response [98].

Scalability is ability to increase features and work with other platforms [136].

Payload ratio is the size of the payload in the test [114].

Mutation Score [140] is the ratio between the number of test cases included in the test set to the total number of test cases generated.

VII. ANALYSIS

Security is unarguably the most key concern for web applications, to which SQL Injection (SQLI) and XSS attacks are the most ruinous attacks. The pie chart of Figure 4 affirmed that injection vulnerabilities are most danger specifically SQL Injection and there are more research papers than other issues in web application. SQL Injection has more papers because it is ranked number 1 in injection vulnerabilities. Regarding our proposed classification the query manipulation has more papers followed by client side injection. Specifically, XSS attacks are the most popular in client side injection category. Information disclosure category has fewest solution papers. Moreover, directory indexing, information leaks and predictable resource location have no solution articles from sounded journals. Therefore there is a demand for more research solutions in these topics.

As proclaimed in Figures 17 and 18, dynamic is the most implemented one in this survey because dynamic is more easily to develop than static. Dynamic work is similar to fuzzing concept that send large number of requests and notice responses for some characteristics, if found that means there is a vulnerability. Regarding the Information disclosure category, all reviewed articles use dynamic techniques. Nevertheless, static technique needs to specify target language to protect and know its critical sink and code structure but it suffers from more FPs. Howbeit, static techniques are more complex yet accurate because it scans source code not just sends a request.

Whilst the works summarized in the previous sections are of obvious value to input validation vulnerabilities in web applications, there is, in our opinion, a scarcity in hybrid solutions in all categories. We noticed few solutions take advantage of both static and dynamic techniques and merged them together. Therefore, there is a need for more study in hybrid techniques and also more accurate static techniques.

Figure 19 illustrates that the most used target applications for presented solutions are real and custom web application because they are most and old form of web application, whereas web service is less used. Furthermore, to our knowledge, there is no test in real web service. Therefore, in future researchers have to focus to test their solutions in real web services because web services are becoming more important and more frequently used nowadays. Web service used to allow various applications to communicate such as web application with desktop application or mobile app, escetra.

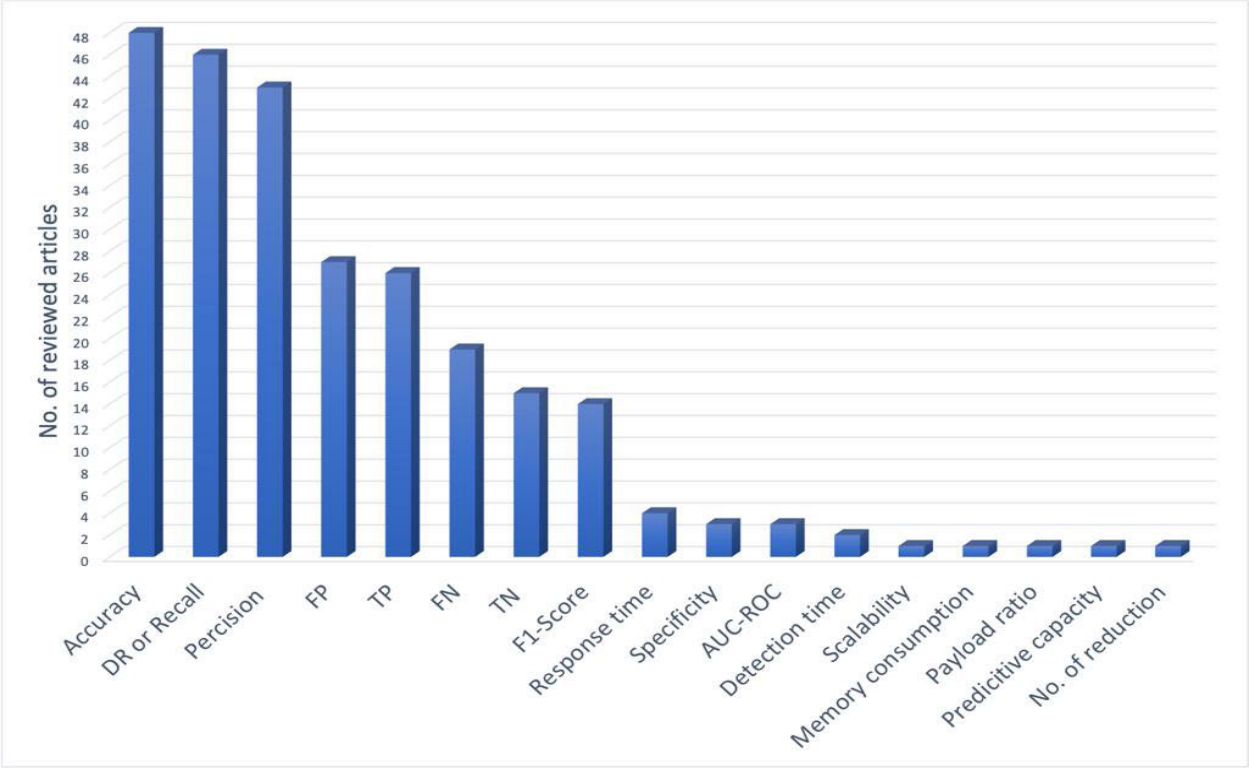

FIGURE 16. Evaluation metrics distribution.

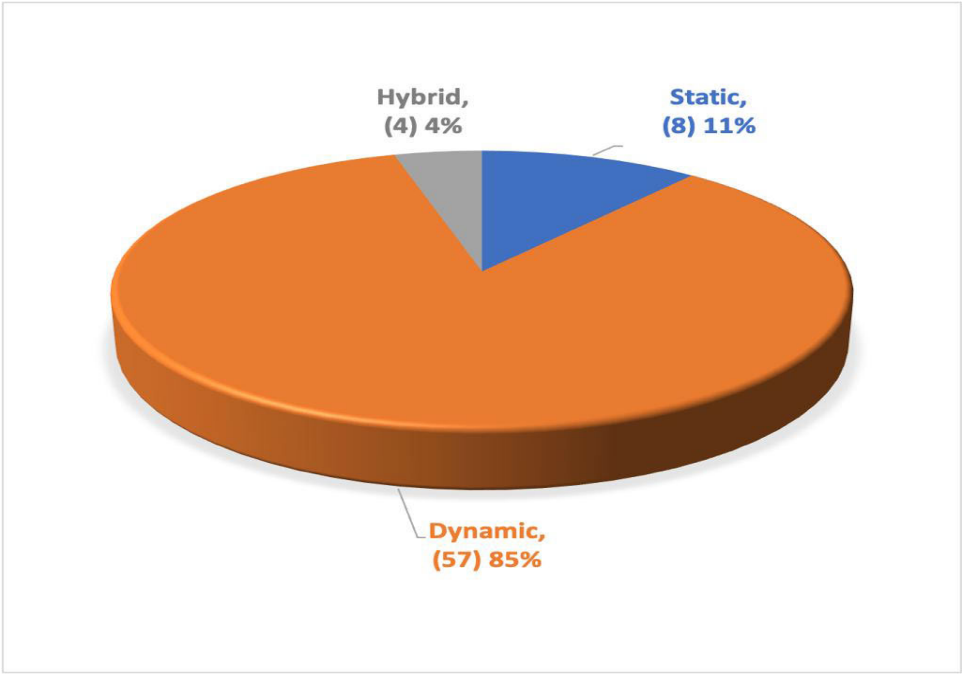

FIGURE 17. Distribution of query manipulation static/dynamic techniques.

When static technique is used, target language have to be specified because it will find vulnerability based on the syntax of the language. Also there are few solutions that implement dynamic technique require specific language. Figure 20 demonstrates the distribution of target language used in the literature solutions and shows that PHP language

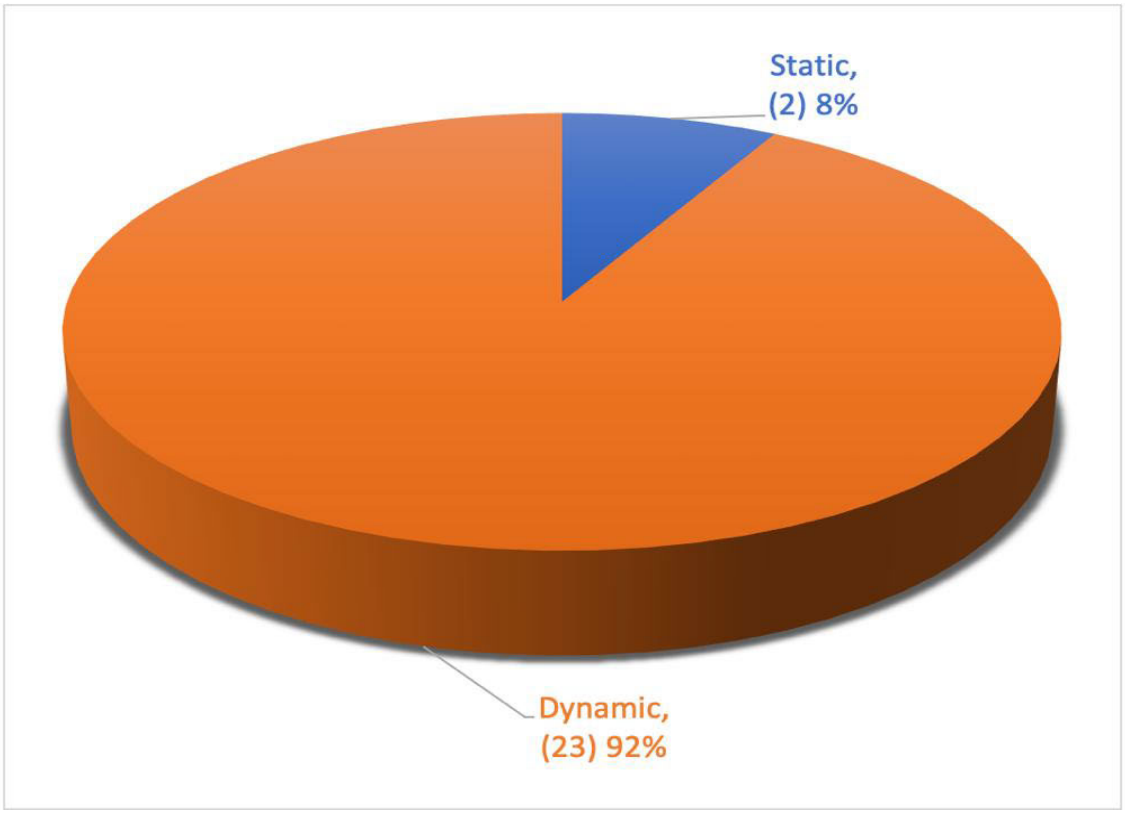

FIGURE 18. Distribution of client side static/dynamic techniques.

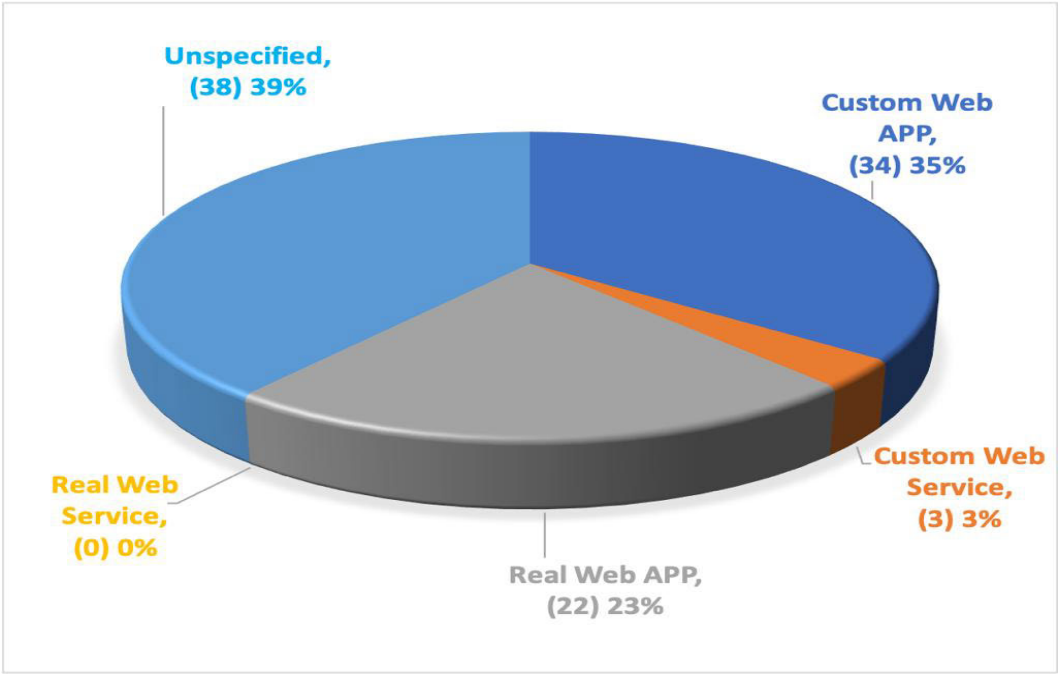

FIGURE 19. Distribution of the target application commonly used.

is the most target language which researchers focus on for protection. This is because it is the most used back-end language in internet and most developer start with this language. Therefore, there is a need to develop more static security solutions that focus on JAVA and .NET language. Furthermore, other back-end languages and framework such as node.JS,

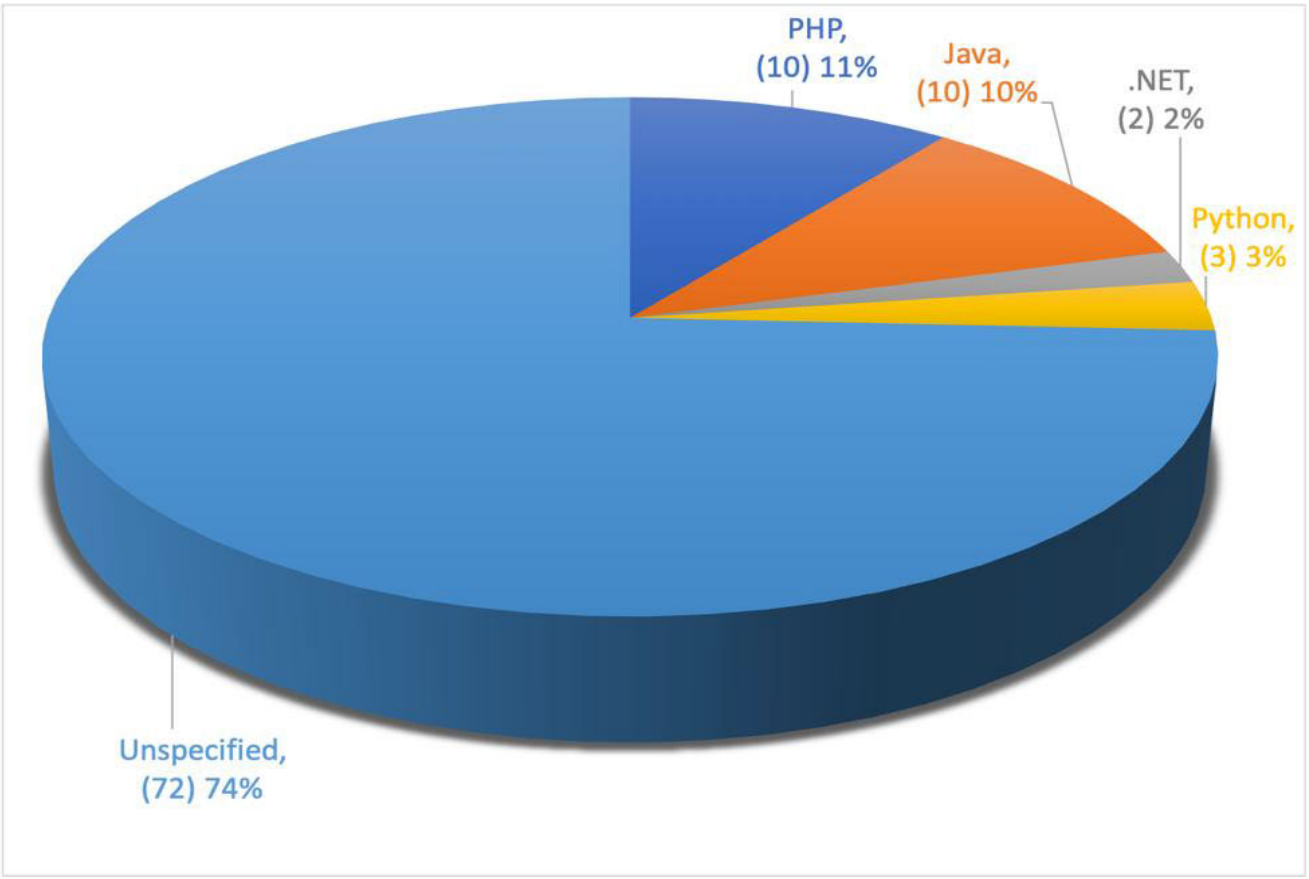

FIGURE 20. Distribution of the language commonly used.

Python, Ruby, Laravel, Django etc, have few studies in the literature.

Based on the aforementioned review, the deep learning-based XSS detection has commenced from 2018 till up-to-date. Multifarious DL techniques have been amalgamated to develop efficient and effective schemes to detect/prevent various attacks from different categories viz. MLP, LSTM, DBN, ANN, CNN, CDNN, and RNN. Thus, this offers a profound insights into the need of in depth study of DL techniques.

VIII. LIMITATIONS AND FUTURE WORK

This section highlights the limitations of the prevailing work. The main aspect that can be covered further in this review is a comparison in real environment of the proposed tools/solutions in previous study and notice their effectiveness for detecting vulnerabilities. Future work will focus on add more vulnerabilities to each category as well as more research papers that focus on data mining and machine learning techniques. In addition, a unified evaluation environment that includes controlled parameters can be designed to enable a fair comparison for each mentioned solution.

IX. CONCLUSION

Security is indisputably the most serious concern for Web applications. Input validation vulnerabilities happens because user enters malicious input and servers have no validation or sanitization of user input. In this paper, a review has been conducted on the solutions of various input validation vulnerabilities. The solutions have been collected in a systematic manner on four major digital databases. The proposed review aims to shed light on the solutions and ideas proposed in input validation vulnerabilities. Existing reviews focus on general kind of vulnerabilities, yet this paper focused on input validation issues. Moreover, the existing review articles suffer from poor categorization and overlapping. Furthermore, up to our knowledge, there is no deep study for defending against them. This review proposes a new classification of input validation vulnerabilities namely query manipulation, client side injection and information disclosure.

The contribution of this paper is summarized hereafter:

- It reviewed existing web vulnerabilities and their types from 2015 - up-to-date.
- Provided technical as well as literature solutions that reduce web application vulnerabilities. The goal is to elucidate the present methods, unveil and discuss their

pros and drawbacks, moreover investigate the gaps and hence give insight for future research.
- A new classification for input validation vulnerabilities is proffered.
- Displayed statistical distributions of the various techniques in the different proposed categories.
- Demonstrated the distribution of the static, dynamic and hybrid solutions.
- Presented the most common metrics used to measure the performance of the reviewed suggested techniques, and their distributions in the literation solutions.
- Manifested the statistical distribution of the target apps and the programming languages used in the literature.

ACKNOWLEDGMENT

(Faris Faisal Fadlalla and Huwaida T. Elshoush are co-first authors.)

REFERENCES

[1] R-Fielding. (1999). *Hypertext Transfer Protocol-http/1.1. IETF RFC 2616*. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[2] D. Kopec, "History of web programming," in *Dart for Absolute Beginners*. Cham, Switzerland: Springer, 2014, pp. 275–286.

[3] J. Fonseca, M. Vieira, and H. Madeira, "The web attacker perspective—A field study," in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng.*, Nov. 2010, pp. 299–308.

[4] A. Delaitre, B. Stivalet, E. Fong, and V. Okun, "Evaluating bug finders–test and measurement of static code analyzers," in *Proc. IEEE/ACM 1st Int. Workshop Complex Faults Failures Large Softw. Syst. (COUFLESS)*, May 2015, pp. 14–20.

[5] T. Scholte, D. Balzarotti, and E. Kirda, "Have things changed now? An empirical study on input validation vulnerabilities in web applications," *Comput. Secur.*, vol. 31, no. 3, pp. 344–356, May 2012.

[6] D. Wichers. *Leadership of the OWASP Top 10 Project*. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[7] *Leadership of the OWASP Top 10 Project*. Accessed: Jun. 3, 2021. [Online]. Available: https://github.com/owasp-top/owasp-top-2007

[8] *Leadership of the OWASP Top 10 Project*. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-pdfarchive/OWASP_AppSec_Research_2010_OWASP_Top_10_by_Wichers.pdf

[9] *Leadership of the OWASP Top 10 Project*. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf

[10] *Leadership of the OWASP Top 10 Project*. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

[11] *Welcome to the OWASP Top 10–2021*, Accessed: Mar. 18, 2023. [Online]. Available: https://owasp.org/Top10/

[12] O. B. Fredj, O. Cheikhrouhou, M. Krichen, H. Hamam, and A. Derhab, "An OWASP top ten driven survey on web application protection methods," in *Proc. Int. Conf. Risks Secur. Internet Syst.* Cham, Switzerland: Springer, 2020, pp. 235–252.

[13] G. K. Pannu, "A survey on web application attacks," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 3, pp. 1–5, 2014.

[14] H. Atashzar, A. Torkaman, M. Bahrololum, and M. H. Tadayon, "A survey on web application vulnerabilities and countermeasures," in *Proc. 6th Int. Conf. Comput. Sci. Converg. Inf. Technol. (ICCIT)*, 2011, pp. 647–652.

[15] O. B. Al-Khurafi and M. A. Al-Ahmad, "Survey of web application vulnerability attacks," in *Proc. 4th Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT)*, Dec. 2015, pp. 154–158.

[16] C. Meghana, B. Chaitra, and V. Nagaveni, "Survey on—Web application attack detection using data mining techniques," *J. Comput., Internet Netw. Secur.*, vol. 4, no. 2, pp. 154–158, 2018.

[17] M. Khari, "Web-application attacks: A survey," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, 2016, pp. 2187–2191.

[18] N. ElBachirElMoussaid and A. Toumanari, "Web application attacks detection: A survey and classification," *Int. J. Comput. Appl.*, vol. 103, no. 12, pp. 1–6, Oct. 2014.

[19] SANS. *Private U.S. for-Profit Company*. Accessed: Jun. 3, 2021. [Online]. Available: https://www.sans.org/top25-software-errors/

[20] I. V. de Sousa Medeiros, "Detection of vulnerabilities and automatic protection for web applications," Ph.D. dissertation, Dept. Comput. Sci., Universidade de Lisboa, Portugal, 2016.

[21] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Saner: Composing static and dynamic analysis to validate sanitization in web applications," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2008, pp. 387–401.

[22] A. Algaith, P. Nunes, F. Jose, I. Gashi, and M. Vieira, "Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools," in *Proc. 14th Eur. Dependable Comput. Conf. (EDCC)*, Sep. 2018, pp. 57–64.

[23] J. Dahse and T. Holz, "Simulation of built-in php features for precise static code analysis," in *Proc. NDSS Symp.*, vol. 14. Princeton, NJ, USA: Citeseer, 2014, pp. 23–26.

[24] D. Hauzar and J. Kofron, "Framework for static analysis of php applications," in *Proc. 29th Eur. Conf. Object-Oriented Program. (ECOOP)*. Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015, pp. 689–711.

[25] N. L. de Poel, F. B. Brokken, and G. R. R. de Lavalette, "Automated security review of PHP web applications with static code analysis," M.S. thesis, Dept. Comput. Sci., vol. 5, 2010.

[26] P. J. C. Nunes, J. Fonseca, and M. Vieira, "PhpSAFE: A security analysis tool for OOP web application plugins," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 299–306.

[27] N. Munaiah, "Assisted discovery of software vulnerabilities," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 464–467.

[28] P. E. Black, P. E. Black, M. Kass, M. Koo, and E. Fong, "Source code security analysis tool functional specification version 1.0," U.S. Dept. Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 500-268, 2007.

[29] C. Cao, N. Gao, P. Liu, and J. Xiang, "Towards analyzing the input validation vulnerabilities associated with Android system services," in *Proc. 31st Annu. Comput. Secur. Appl. Conf.*, Dec. 2015, pp. 361–370.

[30] A. Z. Baset and T. Denning, "IDE plugins for detecting input-validation vulnerabilities," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2017, pp. 143–146.

[31] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 171–180.

[32] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "An empirical analysis of input validation mechanisms in web applications and languages," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, Mar. 2012, pp. 1419–1426.

[33] E. Ufuktepe and T. Tuglular, "Estimating software robustness in relation to input validation vulnerabilities using Bayesian networks," *Softw. Quality J.*, vol. 26, no. 2, pp. 455–489, Jun. 2018.

[34] M. Alkhalaf, S. R. Choudhary, M. Fazzini, T. Bultan, A. Orso, and C. Kruegel, "ViewPoints: Differential string analysis for discovering client- and server-side input validation inconsistencies," in *Proc. Int. Symp. Softw. Test. Anal.*, Jul. 2012, pp. 56–66.

[35] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proc. 19th Int. Conf. World Wide Web*, Apr. 2010, pp. 921–930.

[36] E. Shaji and N. Subramanian, "Assessing non-intrusive vulnerability scanning methodologies for detecting web application vulnerabilities on large scale," in *Proc. Int. Conf. Syst., Comput., Autom. Netw. (ICSCAN)*, Jul. 2021, pp. 1–5.

[37] A. Lavrenovs and F. J. R. Melon, "HTTP security headers analysis of top one million websites," in *Proc. 10th Int. Conf. Cyber Conflict (CyCon)*, May 2018, pp. 345–370.

[38] E. Budianto, Y. Jia, X. Dong, P. Saxena, and Z. Liang, "You can't be me: Enabling trusted paths and user sub-origins in web browsers," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2014, pp. 150–171.

[39] V. Clincy and H. Shahriar, "Web application firewall: Network security models and configuration," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 835–836.

[40] T. Krueger, C. Gehl, K. Rieck, and P. Laskov, "TokDoc: A self-healing web application firewall," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2010, pp. 1846–1853.
[41] A. Razzaq, A. Hur, S. Shahbaz, M. Masood, and H. F. Ahmad, "Critical analysis on web application firewall solutions," in *Proc. IEEE 11th Int. Symp. Auto. Decentralized Syst. (ISADS)*, Mar. 2013, pp. 1–6.
[42] M-Official Site, *X-XSS-Protection*, Accessed: Jun. 3, 2021. [Online]. Available: https://developer.mozilla.org/en-U.S./docs/Web/HTTP/Headers/X-XSS-Protection
[43] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in *Proc. IEEE Symp. Secur. Privacy*, Dec. 2010, pp. 332–345.
[44] K. A. Williams, X. Yuan, H. Yu, and K. Bryant, "Teaching secure coding for beginning programmers," *J. Comput. Sci. Colleges*, vol. 29, no. 5, pp. 91–99, 2014.
[45] M. Zaidman, "Teaching defensive programming in Java," *J. Comput. Sci. Colleges*, vol. 19, no. 3, pp. 33–43, 2004.
[46] X. Qie, R. Pang, and L. Peterson, "Defensive programming: Using an annotation toolkit to build DoS-resistant software," in *Proc. 5th Symp. Operating Syst. Design Implement. (OSDI)*, 2002, pp. 1–16.
[47] B. Chen, D.-W. Xu, S.-D. Gao, and L. Yu, "Cultivating the ability of security coding for undergraduates in programming teaching," in *Proc. 4th Int. Conf. Comput. Sci. Educ.*, Jul. 2009, pp. 1425–1430.
[48] M. Stueben, *Good Habits for Great Coding*. Cham, Switzerland: Springer, 2018.
[49] L. Conklin. *OWASP Code Review Guide*. Accessed: Jun. 10, 2021. [Online]. Available: https://owasp.org/www-project-code-review-guide/
[50] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Sep. 2006, pp. 343–350.
[51] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security Privacy*, vol. 2, no. 6, pp. 76–79, Nov. 2004.
[52] M. Berman, S. Adams, T. Sherburne, C. Fleming, and P. Beling, "Active learning to improve static analysis," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 1322–1327.
[53] Z. Zhioua, S. Short, and Y. Roudier, "Static code analysis for software security verification: Problems and approaches," in *Proc. IEEE 38th Int. Comput. Softw. Appl. Conf. Workshops*, Jul. 2014, pp. 102–109.
[54] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Static analysis-based approaches for secure software development," in *Proc. Int. ISCIS Secur. Workshop*. Cham, Switzerland: Springer, 2018, pp. 142–157.
[55] G. Díaz and J. R. Bermejo, "Static analysis of source code security: Assessment of tools against SAMATE tests," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1462–1476, Aug. 2013.
[56] M. Bishop, "Checking for race conditions in file accesses," *Comput. Syst.*, vol. 2, no. 2, pp. 131–152, 1996.
[57] R. Dewhurst. *Static Code Analysis*. Accessed: Oct. 10, 2020. [Online]. Available: https://owasp.org/www-community/controls/Static_Code_Analysis
[58] W. R. Bush, J. D. Pincus, and D. J. Sielaff, "A static analyzer for finding dynamic programming errors," *Softw., Pract. Exper.*, vol. 30, no. 7, pp. 775–802, Jun. 2000.
[59] M. Sridharan, S. Artzi, M. Pistoia, S. Guarnieri, O. Tripp, and R. Berg, "F4F: Taint analysis of framework-based web applications," in *Proc. ACM Int. Conf. Object Oriented Program. Syst. Lang. Appl.*, 2011, pp. 1053–1068.
[60] J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw, "ITS4: A static vulnerability scanner for C and C++ code," in *Proc. 16th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2001, pp. 257–269.
[61] A. Petukhov and D. Kozlov, "Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing," Comput. Syst. Lab., Dept. Comput. Sci., Moscow State Univ., Moscow, Russia, 2008, pp. 1–120.
[62] G. Antoniol, M. Di Penta, and M. Zazzara, "Understanding web applications through dynamic analysis," in *Proc. 12th IEEE Int. Workshop Program Comprehension*, Jun. 2004, pp. 120–129.
[63] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *J. Comput. Virol.*, vol. 2, no. 1, pp. 67–77, 2006.
[64] L. Li and C. Wang, "Dynamic analysis and debugging of binary code for security applications," in *Proc. Int. Conf. Runtime Verification*. Cham, Switzerland: Springer, 2013, pp. 403–423.
[65] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "Jäk: Using dynamic analysis to crawl and test modern web applications," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2015, pp. 295–316.
[66] G. A. Di Lucca and M. Di Penta, "Integrating static and dynamic analysis to improve the comprehension of existing web applications," in *Proc. 7th IEEE Int. Symp. Web Site Evol.*, Jul. 2005, pp. 87–94.
[67] G. A. Di Lucca, M. Di Penta, A. R. Fasolino, and P. Tramontana, "Supporting web application evolution by dynamic analysis," in *Proc. 8th Int. Workshop Princ. Softw. Evol. (IWPSE)*, 2005, pp. 175–184.
[68] A. Alhuzali, R. Gjomemo, B. Eshete, and V. Venkatakrishnan, "NAVEX: Precise and scalable exploit generation for dynamic web applications," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 377–392.
[69] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, vol. 2.
[70] S. Rawat, D. Ceara, L. Mounier, and M.-L. Potet, "Combining static and dynamic analysis for vulnerability detection," 2013, *arXiv:1305.3883*.
[71] X. He, L. Xu, and C. Cha, "Malicious Javascript code detection based on hybrid analysis," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 365–374.
[72] I. Medeiros, N. Neves, and M. Correia, "Equipping WAP with WEAPONS to detect vulnerabilities: Practical experience report," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016, pp. 630–637.
[73] I. Medeiros, N. Neves, and M. Correia, "Statically detecting vulnerabilities by processing programming languages as natural languages," *IEEE Trans. Rel.*, vol. 71, no. 2, pp. 1033–1056, Jun. 2022.
[74] H. Tribus, I. Morrigl, and S. Axelsson, "Using data mining for static code analysis of C," in *Proc. Int. Conf. Adv. Data Mining Appl.* Cham, Switzerland: Springer, 2012, pp. 603–614.
[75] V. Barstad, M. Goodwin, and T. Gjøsæter, "Predicting source code quality with static analysis and machine learning," in *Norsk IKT-Konferanse for Forskning og Utdanning*, 2014.
[76] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Trans. Dependable Secure Computing*, vol. 12, no. 6, pp. 688–707, Nov. 2015.
[77] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, 2018, pp. 1–10.
[78] G. Grieco and A. Dinaburg, "Toward smarter vulnerability discovery using machine learning," in *Proc. 11th ACM Workshop Artif. Intell. Secur.*, Jan. 2018, pp. 48–56.
[79] J. A. Kupsch and B. P. Miller, "Manual vs. automated vulnerability assessment: A case study," in *Proc. 1st Int. Workshop Manag. Insider Secur. Threats (MIST)*, 2009, pp. 83–97.
[80] P. E. Black, "Counting bugs is harder than you think," in *Proc. IEEE 11th Int. Work. Conf. Source Code Anal. Manipulation*, Sep. 2011, pp. 1–9.
[81] N. Patel and N. Shekokar, "Implementation of pattern matching algorithm to defend SQLIA," *Proc. Comput. Sci.*, vol. 45, pp. 453–459, Jan. 2015.
[82] L. Li, J. Qi, N. Liu, L. Han, and B. Cui, "Static-based test case dynamic generation for SQLIVs detection," in *Proc. 10th Int. Conf. Broadband Wireless Comput., Commun. Appl. (BWCCA)*, Nov. 2015, pp. 173–177.
[83] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Numerical encoding to tame SQL injection attacks," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2016, pp. 1253–1256.
[84] M. Chenyu and G. Fan, "Defending SQL injection attacks based-on intention-oriented detection," in *Proc. 11th Int. Conf. Comput. Sci. Educ. (ICCSE)*, Aug. 2016, pp. 939–944.
[85] R. P. Karuparthi and B. Zhou, "Enhanced approach to detection of SQL injection attack," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 466–469.
[86] M. Ceccato, C. D. Nguyen, D. Appelt, and L. C. Briand, "SOFIA: An automated security Oracle for black-box testing of SQL-injection vulnerabilities," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, Aug. 2016, pp. 167–177.
[87] M. Lodeiro-Santiago, C. Caballero-Gil, and P. Caballero-Gil, "Collaborative SQL-injections detection system with machine learning," in *Proc. 1st Int. Conf. Internet Things Mach. Learn.*, Oct. 2017, pp. 1–5.

[88] Y. Zhu, G. Zhang, Z. Lai, B. Niu, and Y. Shen, "A two-tiered defence of techniques to prevent SQL injection attacks," in *Proc. Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.* Cham, Switzerland: Springer, 2017, pp. 286–295.

[89] K. Umar, A. B. Sultan, H. Zulzalil, N. Admodisastro, and M. T. Abdullah, "Formulation of SQL injection vulnerability detection as grammar reachability problem," in *Proc. Int. Conf. Inf. Commun. Technol. Muslim World (ICTM)*, Jul. 2018, pp. 179–184.

[90] L. Saoudi, K. Adi, and Y. Boudraa, "A rejection-based approach for detecting SQL injection vulnerabilities in web applications," in *Proc. Int. Symp. Found. Pract. Secur.* Cham, Switzerland: Springer, 2019, pp. 379–386.

[91] H. Gao, J. Zhu, L. Liu, J. Xu, Y. Wu, and A. Liu, "Detecting SQL injection attacks using grammar pattern recognition and access behavior mining," in *Proc. IEEE Int. Conf. Energy Internet (ICEI)*, May 2019, pp. 493–498.

[92] M. S. Aliero, I. Ghani, K. N. Qureshi, and M. F. Rohani, "An algorithm for detecting SQL injection vulnerability using black-box testing," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 1, pp. 249–266, Jan. 2020.

[93] A. Kumar, S. Rai, and R. Boghey, "A novel approach for sql injection avoidance using two-level restricted application prevention (trap) technique," in *Proc. Int. Conf. Innov. Comput. Commun.* Cham, Switzerland: Springer, 2021, pp. 227–238.

[94] B. M. Thombare and D. R. Soni, "Prevention of SQL injection attack by using black box testing," in *Proc. 23rd Int. Conf. Distrib. Comput. Netw.*, Jan. 2022, pp. 266–272.

[95] K. D'silva, J. Vanajakshi, K. N. Manjunath, and S. Prabhu, "An effective method for preventing SQL injection attack and session hijacking," in *Proc. 2nd IEEE Int. Conf. Recent Trends Electron., Inf. Commun. Technol. (RTEICT)*, May 2017, pp. 697–701.

[96] K. Zhang, "A machine learning based approach to identify SQL injection vulnerabilities," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1286–1288.

[97] O. C. Abikoye, A. Abubakar, A. H. Dokoro, O. N. Akande, and A. A. Kayode, "A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-morris-pratt string match algorithm," *EURASIP J. Inf. Secur.*, vol. 2020, no. 1, pp. 1–14, Dec. 2020.

[98] R. Jahanshahi, A. Doupé, and M. Egele, "You shall not pass: Mitigating SQL injection attacks on legacy web applications," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 445–457.

[99] T. Latchoumi, M. S. Reddy, and K. Balamurugan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," *Eur. J. Mol. Clin. Med.*, vol. 7, no. 2, p. 2020, 2020.

[100] K. Kamtuo and C. Soomlek, "Machine learning for SQL injection prevention on server-side scripting," in *Proc. Int. Comput. Sci. Eng. Conf. (ICSEC)*, Dec. 2016, pp. 1–6.

[101] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1087–1090.

[102] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "An applied pattern-driven corpus to predictive analytics in mitigating SQL injection attack," in *Proc. 7th Int. Conf. Emerg. Secur. Technol. (EST)*, Sep. 2017, pp. 12–17.

[103] Z. Chen and M. Guo, "Research on SQL injection detection technology based on SVM," in *Proc. MATEC Web Conf.*, vol. 173. Les Ulis, France: EDP Sciences, 2018, p. 01004.

[104] K. Ross, M. Moh, T.-S. Moh, and J. Yao, "Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection," in *Proc. ACMSE Conf.*, Mar. 2018, pp. 1–8.

[105] M. Volkova, P. Chmelar, and L. Sobotka, "Machine learning blunts the needle of advanced SQL injections," *MENDEL*, vol. 25, no. 1, pp. 23–30, Jun. 2019.

[106] Y. Li and B. Zhang, "Detection of SQL injection attacks based on improved TFIDF algorithm," *J. Phys., Conf.*, vol. 1395, no. 1, Nov. 2019, Art. no. 012013.

[107] M. Hasan, Z. Balbahaith, and M. Tarique, "Detection of SQL injection attacks: A machine learning approach," in *Proc. Int. Conf. Electr. Comput. Technol. Appl. (ICECTA)*, Nov. 2019, pp. 1–6.

[108] K. Kuroki, Y. Kanemoto, K. Aoki, Y. Noguchi, and M. Nishigaki, "Attack intention estimation based on syntax analysis and dynamic analysis for SQL injection," in *Proc. IEEE 44th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Jul. 2020, pp. 1510–1515.

[109] R. K. Pathak, "Handling sql injection attack using progressive neural network," in *Proc. Int. Conf. Inf., Commun. Comput. Technol.* Cham, Switzerland: Springer, 2020, pp. 231–241.

[110] D. Tripathy, R. Gohil, and T. Halabi, "Detecting SQL injection attacks in cloud SaaS using machine learning," in *Proc. IEEE IEEE 6th Intl Conf. Big Data Secur. Cloud (BigDataSecurity) Intl Conf. High Perform. Smart Comput., (HPSC) IEEE Intl Conf. Intell. Data Secur. (IDS)*, May 2020, pp. 145–150.

[111] D. Parashar, L. M. Sanagavarapu, and Y. R. Reddy, "SQL injection vulnerability identification from text," in *Proc. 14th Innov. Softw. Eng. Conf. Formerly Known India Softw. Eng. Conf.*, Feb. 2021, pp. 1–5.

[112] M. Gowtham and H. Pramod, "Semantic query-featured ensemble learning model for SQL-injection attack detection in IoT-ecosystems," *IEEE Trans. Rel.*, vol. 71, no. 2, pp. 1057–1074, Jun. 2022.

[113] R. R. Choudhary, S. Verma, and G. Meena, "Detection of SQL injection attack using machine learning," in *Proc. IEEE Int. Conf. Technol., Res., Innov. Betterment Soc. (TRIBES)*, Dec. 2021, pp. 1–6.

[114] Y. Luo, "SQLi-fuzzer: A SQL injection vulnerability discovery framework based on machine learning," in *Proc. IEEE 21st Int. Conf. Commun. Technol. (ICCT)*, Oct. 2021, pp. 846–851.

[115] S. Rahul, C. Vajrala, and B. Thangaraju, "A novel method of honeypot inclusive WAF to protect from SQL injection and XSS," in *Proc. Int. Conf. Disruptive Technol. Multi-Disciplinary Res. Appl. (CENTCON)*, vol. 1, Nov. 2021, pp. 135–140.

[116] A. Sivasangari, J. Jyotsna, and K. Pravalika, "SQL injection attack detection using machine learning algorithm," in *Proc. 5th Int. Conf. Trends Electron. Informat. (ICOEI)*, 2021, pp. 1166–1169.

[117] M. O. Adebiyi, M. O. Arowolo, G. I. Archibong, M. D. Mshelia, and A. A. Adebiyi, "An sql injection detection model using chi-square with classification techniques," in *Proc. Int. Conf. Electr., Comput. Energy Technol. (ICECET)*, Dec. 2021, pp. 1–8.

[118] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "SQL injection behavior mining based deep learning," in *Proc. Int. Conf. Adv. Data Mining Appl.* Cham, Switzerland: Springer, 2018, pp. 445–454.

[119] Q. Li, F. Wang, J. Wang, and W. Li, "LSTM-based SQL injection detection method for intelligent transportation system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4182–4191, May 2019.

[120] H. Zhang, B. Zhao, H. Yuan, J. Zhao, X. Yan, and F. Li, "SQL injection detection based on deep belief network," in *Proc. 3rd Int. Conf. Comput. Sci. Appl. Eng.*, Oct. 2019, pp. 1–6.

[121] X. Xie, C. Ren, Y. Fu, J. Xu, and J. Guo, "SQL injection detection for web applications based on elastic-pooling CNN," *IEEE Access*, vol. 7, pp. 151475–151481, 2019.

[122] M. Liu, K. Li, and T. Chen, "DeepSQLi: Deep semantic learning for testing SQL injection," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 286–297.

[123] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "Detection of SQL injection based on artificial neural network," *Knowl.-Based Syst.*, vol. 190, Feb. 2020, Art. no. 105528.

[124] M. Li, B. Liu, G. Xing, X. Wang, and Z. Wang, "Research on integrated detection of SQL injection behavior based on text features and traffic features," in *Proc. Int. Conf. Comput. Eng. Netw.* Cham, Switzerland: Springer, 2020, pp. 755–771.

[125] D. Chen, Q. Yan, C. Wu, and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *J. Phys., Conf.*, vol. 1757, no. 1, Jan. 2021, Art. no. 012055.

[126] K. Jothi, N. Pandey, P. Beriwal, and A. Amarajan, "An efficient SQL injection detection system using deep learning," in *Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE)*, Mar. 2021, pp. 442–445.

[127] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, "A CNN-BiLSTM based approach for detection of SQL injection attacks," in *Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE)*, Mar. 2021, pp. 378–383.

[128] P. Wen, C. He, W. Xiong, and J. Liu, "SQL injection detection technology based on BiLSTM-attention," in *Proc. 4th Int. Conf. Robot., Control Autom. Eng. (RCAE)*, Nov. 2021, pp. 165–170.

[129] A. A. R. Farea, C. Wang, E. Farea, and A. Ba Alawi, "Cross-site scripting (XSS) and SQL injection attacks multi-classification using bidirectional LSTM recurrent neural network," in *Proc. IEEE Int. Conf. Prog. Informat. Comput. (PIC)*, Dec. 2021, pp. 358–363.

[130] W. Zhang, Y. Li, X. Li, M. Shao, Y. Mi, H. Zhang, and G. Zhi, "Deep neural network-based SQL injection detection method," *Secur. Commun. Netw.*, vol. 2022, pp. 1–9, Mar. 2022.

[131] A. Falor, M. Hirani, H. Vedant, P. Mehta, and D. Krishnan, "A deep learning approach for detection of SQL injection attacks using convolutional neural networks," in *Proc. Data Analytics Manage.* Cham, Switzerland: Springer, 2022, pp. 293–304.
[132] Q. Li, W. Li, J. Wang, and M. Cheng, "A SQL injection detection method based on adaptive deep forest," *IEEE Access*, vol. 7, pp. 145385–145394, 2019.
[133] Z. Xiao, Z. Zhou, W. Yang, and C. Deng, "An approach for SQL injection detection based on behavior and response analysis," in *Proc. IEEE 9th Int. Conf. Commun. Softw. Netw. (ICCSN)*, May 2017, pp. 1437–1442.
[134] X. Qi and T. Dai, "Research on the detection method of SQL injection attack based on sequence alignment," *J. Phys., Conf.*, vol. 1550, no. 3, May 2020, Art. no. 032054.
[135] G. Singh, D. Kant, U. Gangwar, and A. P. Singh, "SQL injection detection and correction using machine learning techniques," in *Proc. Emerg. ICT Bridging Future 49th Annu. Conv. Comput. Soc. India (CSI)*, vol. 1. Cham, Switzerland: Springer, 2015, pp. 435–442.
[136] J. Thome, L. K. Shar, and L. Briand, "Security slicing for auditing XML, XPath, and SQL injection vulnerabilities," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 553–564.
[137] J. Thomé, L. K. Shar, D. Bianculli, and L. C. Briand, "JoanAudit: A tool for auditing common injection vulnerabilities," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 1004–1008.
[138] V. Clincy and H. Shahriar, "Web service injection attack detection," in *Proc. 12th Int. Conf. for Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 173–178.
[139] G. Deshpande and S. Kulkarni, "Modeling and mitigation of XPath injection attacks for web services using modular neural networks," in *Recent Findings in Intelligent Computing Techniques*. Cham, Switzerland: Springer, 2019, pp. 301–310.
[140] P. Bulusu, "Detection of lightweight directory access protocol query injection attacks in web applications," 2015.
[141] H. Shahriar, H. M. Haddad, and P. Bulusu, "OCL fault injection-based detection of LDAP query injection vulnerabilities," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jun. 2016, pp. 455–460.
[142] M. K. Jawalkar, P. S. Gokhale, and A. M. Dixit, "JIID: Java input injection detector for pre-deployment vulnerability detection," in *Proc. IEEE Int. Conf. Res. Comput. Intell. Commun. Netw. (ICRCICN)*, Nov. 2015, pp. 444–449.
[143] C. Alonso, R. Bordón, M. Beltrán, and A. Guzmán, "LDAP injection & blind LDAP injection," *Figure*, vol. 1, p. 4, 2008.
[144] H. Ma, T.-Y. Wu, M. Chen, R.-H. Yang, and J.-S. Pan, "A parse tree-based NoSQL injection attacks detection mechanism," *J. Inf. Hiding Multimedia Signal Process.*, vol. 8, no. 4, pp. 916–928, 2017.
[145] S. Joseph and K. Jevitha, "An automata based approach for the prevention of NoSQL injections," in *Proc. Int. Symp. Secur. Comput. Commun.* Cham, Switzerland: Springer, 2015, pp. 538–546.
[146] A. M. Eassa, M. Elhoseny, H. M. El-Bakry, and A. S. Salama, "NoSQL injection attack detection in web applications using RESTful service," *Program. Comput. Softw.*, vol. 44, no. 6, pp. 435–444, Nov. 2018.
[147] M. R. Ul Islam, M. S. Islam, Z. Ahmed, A. Iqbal, and R. Shahriyar, "Automatic detection of NoSQL injection using supervised learning," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2019, pp. 760–769.
[148] V. S. Stency and N. Mohanasundaram, "A study on XSS attacks: Intelligent detection methods," *J. Phys., Conf.*, vol. 1767, no. 1, Feb. 2021, Art. no. 012047.
[149] "Intelligent detection methods," *J. Phys., Conf.*, vol. 1767, no. 1, 2021, Art. no. 012047.
[150] H. Takahashi, K. Yasunaga, M. Mambo, K. Kim, and H. Y. Youm, "Preventing abuse of cookies stolen by XSS," in *Proc. 8th Asia Joint Conf. Inf. Secur.*, Jul. 2013, pp. 85–89.
[151] S. Rathore, P. K. Sharma, and J. H. Park, "XSSClassifier: An efficient XSS attack detection approach based on machine learning classifier on SNSs," *J. Inf. Process. Syst.*, vol. 13, no. 4, pp. 1014–1028, 2017.
[152] R. Banerjee, A. Baksi, N. Singh, and S. K. Bishnu, "Detection of XSS in web applications using machine learning classifiers," in *Proc. 4th Int. Conf. Electron., Mater. Eng. Nano-Technol. (IEMENTech)*, Oct. 2020, pp. 1–5.
[153] G. Kaur, Y. Malik, H. Samuel, and F. Jaafar, "Detecting blind cross-site scripting attacks using machine learning," in *Proc. Int. Conf. Signal Process. Mach. Learn.* Cham, Switzerland: Springer, Nov. 2018, pp. 200–210.
[154] B. Gogoi, T. Ahmed, and H. K. Saikia, "Detection of XSS attacks in web applications: A machine learning approach," *Int. J. Innov. Res. Comput. Sci. Technol.*, vol. 9, no. 1, pp. 2347–5552, 2021.
[155] C. Gupta, R. K. Singh, and A. K. Mohapatra, "GeneMiner: A classification approach for detection of XSS attacks on web services," *Comput. Intell. Neurosci.*, vol. 2022, pp. 1–12, Jun. 2022.
[156] J. Lu, Z. Wei, Z. Qin, Y. Chang, and S. Zhang, "Resolving cross-site scripting attacks through fusion verification and machine learning," *Mathematics*, vol. 10, no. 20, p. 3787, Oct. 2022.
[157] Z. Liu, Y. Fang, C. Huang, and Y. Xu, "GAXSS: Effective payload generation method to detect XSS vulnerabilities based on genetic algorithm," *Secur. Commun. Netw.*, vol. 2022, pp. 1–15, Mar. 2022.
[158] Q. Wang, H. Yang, G. Wu, K.-K.-R. Choo, Z. Zhang, G. Miao, and Y. Ren, "Black-box adversarial attacks on XSS attack detection model," *Comput. Secur.*, vol. 113, Feb. 2022, Art. no. 102554.
[159] Y. Zhou and P. Wang, "An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence," *Comput. Secur.*, vol. 82, pp. 261–269, May 2019.
[160] S. Abaimov and G. Bianchi, "CODDLE: Code-injection detection with deep learning," *IEEE Access*, vol. 7, pp. 128617–128627, 2019.
[161] L. Lei, M. Chen, C. He, and D. Li, "XSS detection technology based on LSTM-attention," in *Proc. 5th Int. Conf. Control, Robot. Cybern. (CRC)*, Oct. 2020, pp. 175–180.
[162] P. Chaudhary, B. B. Gupta, and A. K. Singh, "Securing heterogeneous embedded devices against XSS attack in intelligent IoT system," *Comput. Secur.*, vol. 118, Jul. 2022, Art. no. 102710.
[163] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *Proc. Int. Conf. Comput. Artif. Intell.*, Mar. 2018, pp. 47–51.
[164] R. Kadhim and M. Gaata, "A hybrid of CNN and lstm methods for securing web application against cross-site scripting attack," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 21, pp. 1022–1029, 2020.
[165] F. M. M. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar, and W. Xiaoxi, "MLPXSS: An integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique," *IEEE Access*, vol. 7, pp. 100567–100580, 2019.
[166] I. Odun-Ayo, W. Toro-Abasi, M. Adebiyi, and O. Alagbe, "An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning," *Bull. Electr. Eng. Informat.*, vol. 10, no. 5, pp. 2442–2453, 2021.
[167] W. Yang, W. Zuo, and B. Cui, "Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network," *IEEE Access*, vol. 7, pp. 29891–29900, 2019.
[168] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, and J. Chen, "Adversarial examples detection for XSS attacks based on generative adversarial networks," *IEEE Access*, vol. 8, pp. 10989–10996, 2020.
[169] S. Gupta and B. B. Gupta, "Automated discovery of Javascript code injection attacks in PHP web applications," *Proc. Comput. Sci.*, vol. 78, pp. 82–87, Jan. 2016.
[170] S. Gupta and B. B. Gupta, "Enhanced XSS defensive framework for web applications deployed in the virtual machines of cloud computing environment," *Proc. Technol.*, vol. 24, pp. 1595–1602, Jan. 2016.
[171] S. Gupta, B. B. Gupta, and P. Chaudhary, "Hunting for DOM-based XSS vulnerabilities in mobile cloud-based online social network," *Future Gener. Comput. Syst.*, vol. 79, pp. 319–336, Feb. 2018.
[172] A. Steinhauser and P. Tuma, "Database traffic interception for graybox detection of stored and context-sensitive XSS," *Digit. Threats, Res. Pract.*, vol. 1, no. 3, pp. 1–23, Sep. 2020.
[173] I. Medeiros, N. Neves, and M. Correia, "DEKANT: A static analysis tool that learns to detect web application vulnerabilities," in *Proc. 25th Int. Symp. Softw. Test. Anal.*, Jul. 2016, pp. 1–11.
[174] I. Medeiros, N. Neves, and M. Correia, "Detecting and removing web application vulnerabilities with static analysis and data mining," *IEEE Trans. Rel.*, vol. 65, no. 1, pp. 54–69, Mar. 2016.
[175] S. P. Chandramouli, P.-M. Bajan, C. Kruegel, G. Vigna, Z. Zhao, A. Doupé, and G.-J. Ahn, "Measuring E-mail header injections on the world wide web," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, Apr. 2018, pp. 1647–1656.
[176] R. Chandel. *Comprehensive Guide on Local File Inclusion (LFI)*. Accessed: Dec. 8, 2022. [Online]. Available: https://www.hackingarticles.in/comprehensive-guide-to-local-file-inclusion/

[177] *Comprehensive Guide on Local File Inclusion (LFI)*. Accessed: Dec. 8, 2022. [Online]. Available: https://www.hackingarticles.in/comprehensive-guide-to-local-file-inclusion/

[178] M. S. Tajbakhsh and J. Bagherzadeh, "A sound framework for dynamic prevention of local file inclusion," in *Proc. 7th Conf. Inf. Knowl. Technol. (IKT)*, May 2015, pp. 1–6.

[179] M. Maruf Hassan, T. Bhuyian, M. Khaled Sohel, M. Hasan Sharif, and S. Biswas, "SAISAN: An automated local file inclusion vulnerability detection model," *Int. J. Eng. Technol.*, vol. 7, no. 2.3, p. 4, Mar. 2018.

[180] Portswigger. *Directory Traversal*. Accessed: Dec. 8, 2022. [Online]. Available: https://portswigger.net/web-security/file-path-traversal

[181] M. Flanders, "A simple and intuitive algorithm for preventing directory traversal attacks," 2019, *arXiv:1908.04502*.

FARIS FAISAL FADLALLA received the B.Sc. degree from the Faculty of Information Technology, Almanhal Academy of Science, Sudan, and the M.Sc. degree from the Faculty of Information Technology, University of Science and Technology, Sudan. He is currently pursuing the Ph.D. degree with the Faculty of Mathematical Sciences and Informatics, University of Khartoum, Sudan.

In 2019, he participated in the Bug Bounty Program. He discovered and reported vulnerabilities in Twitter as well as the U.S. Department of Defense and got awarded as well as put in the hall of fame in Twitter. His research interests include application security, information, and network security.

HUWAIDA T. ELSHOUSH received the bachelor's degree in computer science (Division 1), the master's degree in computer science, and the Ph.D. degree in information security from the Faculty of Mathematical Sciences and Informatics, University of Khartoum, Sudan, in 1994, 2001, and 2012, respectively. Her M.Sc. dissertation was titled, Frame Relay Security.

She is currently an Associate Professor with the Computer Science Department, Faculty of Mathematical Sciences and Informatics, University of Khartoum, where she is also acting as the Head of Research Office. She is also the Deputy Dean of Basic Sciences and Engineering with the Graduate College, University of Khartoum. She has more than 29 publications and some of her publications appeared in *Applied Soft Computing* (Elsevier), *PLOS One*, IEEE Access, *Multimedia Tools and Applications*, *PeerJ Computer Science*, *Journal of Information Hiding and Multimedia Signal Processing*, and Springer book chapters. Her research interests include information security, cryptography, steganography, and intrusion detection systems.

Dr. Elshoush received several awards and honors, including the Second-Place Prize from the ACM Student Research Competition SRC–SAC, Coimbra, Portugal, in 2013. Her article titled "An Improved Framework for Intrusion Alert Correlation" was awarded the Best Student Paper Award from the 2012 International Conference of Information Security and Internet Engineering (ICISIE) in WCE 2012. Other prizes were the best student during the five years of her undergraduate study. She is a reviewer of many international reputable journals related to her fields, including *Applied Soft Computing* (Elsevier).