Security in Web Applications: A Comparative Analysis of Key SQL Injection Detection Techniques

Karel Ronan Veerabudren
School of Digital Technologies
Middlesex University Mauritius
Uniciti, Flic-en-Flac, Mauritius
KV230@live.mdx.ac.uk

Girish Bekaroo
School of Digital Technologies
Middlesex University Mauritius
Uniciti, Flic-en-Flac, Mauritius
g.bekaroo@mdx.ac.mu

***Abstract*—Over the years, technological advances have driven massive proliferation of web systems and businesses have harbored a seemingly insatiable need for Internet systems and services. Whilst data is considered as a key asset to businesses and that their security is of extreme importance, there has been growing cybersecurity threats faced by web systems. One of the key attacks that web applications are vulnerable to is SQL injection (SQLi) attacks and successful attacks can reveal sensitive information to attackers or even deface web systems. As part of SQLi defence strategy, effective detection of SQLi attacks is important. Even though different techniques have been devised over the years to detect SQLi attacks, limited work has been undertaken to review and compare the effectiveness of these detection techniques. As such, in order to address this gap in literature, this paper performs a review and comparative analysis of the different SQLi detection techniques, with the aim to detect SQLi attacks in an effective manner and enhance the security of web applications. As part of the investigation, seven SQLi detection techniques including machine learning based detection are reviewed and their effectiveness against different types of SQLi attacks are compared. Results identified positive tainting and adoption of machine learning among the most effective techniques and stored procedures based SQLi as the most challenging attack to detect.**

Keywords—SQL injection attacks, detection techniques, comparative analysis, SQLi, SQLia, web applications, cybersecurity.

I. INTRODUCTION

During the past two decades, there has been massive digital transformation across various sectors and today, web applications are considered as essential tools that help businesses of different sizes to stand out in the highly competitive market. Due to their importance, the global progressive web application market size is expected to reach USD 10.44 Billion by 2027 (CAGR of 31.9%) from USD 1.13 billion in 2019 [1], thus highlighting a significant growth as more enterprises are establishing their online presence. Such online systems produce and manipulate significant amounts of sensitive data, which are stored in databases, that are hosted within the organisation's servers or somewhere on the cloud. Protecting such data is extremely important to businesses since any breach in security can potentially have major adverse impacts ranging from violation of privacy of customers to reputational and financial damages for the organisation. As such, data is considered as a key asset to businesses and their security is of extreme importance.

Nevertheless, web applications are subject to different kinds of attacks, where malicious users try to gain unauthorised access to the database and the sensitive data stored within. One of the key attacks that web applications are vulnerable to is SQL injection (SQLi) attacks, which involve inserting SQL statements through a web application's input fields or hidden parameters to gain access to resources or make changes to data [2]. SQL injection attacks are popular in web applications because databases usually contain sensitive information, like credit card details and passwords and successful attacks would result in obtaining all these information. Since such successful attacks can reveal valuable information to attackers or even deface websites, SQLi attacks also feature in the top positions in the Open Web Application Security Project (OWASP) Top 10 web application security risks [3] and Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses [4].

A key strategy used to mitigate SQLi attacks involves use of defensive coding practices such as utilization of prepared statements or parameterised queries, input validation at different levels or even escaping user supplied inputs, among others. Nevertheless, defensive coding alone is not sufficient for complete SQLi mitigation. This is because developers tend to leave bugs during software development and in various instances, these bugs progress to the live phase of web systems and consequently, SQLi attacks can still happen [5]. Therefore, as part of SQLi defence strategy, effective detection of SQLi attacks is important. For detecting SQLi attacks, researchers and developers have developed different techniques over the years. However, limited published papers comprehensively review and compare the effectiveness of these techniques. In order to address this gap, this paper performs a review and comparative analysis of the different SQL injection detection techniques, with the aim to enhance security of web applications. Findings revealed in this paper are expected to be insightful to the web development as well as the research and development communities, who can utilize findings and recommendations in this paper in their endeavour to enhance security of their web applications.

This paper is structured as follows: Section II provides a background on SQL injection attack and its different forms. Then, in Section III, works related to comparative analysis of detection of SQLi attacks are provided, before discussing the methodology used to review to achieve the purpose of this paper and compare the SQLi attack detection techniques in Section IV. Section V describes the selected SQLi detection techniques, which are compared in Section VI. The paper is concluded in Section VII.

II. THEORETICAL BACKGROUND: SQL INJECTION ATTACKS

SQL injection attack (SQLi) is a type of security exploit in which the attacker adds SQL statements through a web application's input fields or hidden parameters to gain access

to resources or make changes to data. SQLi enable attackers to access information in an unauthorized manner, modify existing data within databases, destroy data or make it unavailable or even become administrators of database servers. There are different types of SQL injection attacks, and these are discussed in the next sections:

A. *Tautologies*

The common objective of this attack is to bypass the authentication mechanism in place and is achieved by injecting code into condition statement(s) so they always evaluate to true. As an example, consider a simple login page in which the user should type in a username and password. Normally, the username and password should both be valid for a successful login and the underlying SQL statement to process the authentication can be thought of as in Fig. 1.

```
SELECT accounts FROM users WHERE
login= username AND pass= password
```

Fig. 1. The underlying SQL statement for a login page

To perform SQLi, the attacker would enter a valid username and a crafted query in the password field as shown in Fig. 2. As a result, the underlying SQL statement for the login page would be rendered as shown in Fig. 3. The inverted comma closes the password field and the rest of the input password becomes part of the SQL query. In the same resulting statement, "1=1" is always true, making the OR statement true. Thus, the whole query becomes true, and authentication is bypassed.

```
Username: user
Password: ' or 1=1 --
```

Fig. 2. Values for username and password fields

```
SELECT accounts FROM users WHERE
login= 'user' AND pass= '' or 1=1 --'
```

Fig. 3. Resulting authentication SQL statement

B. *Illegal Queries*

The aim of this attack involves gathering information related to the database of the web application [5]. It is carried out by injecting codes in the link of the webpage or in the input fields, where the attacker attempts to cause a run-time error, while hoping to learn information from error responses. Key information retrieved by the attacker includes database version, table name or even column name and the attacker can eventually use such information to make more personalized injection [6]. An example is shown in Fig. 4 that depicts information returned by a database when illegal queries are injected. In this example, the column name and its type are returned to the error page.

```
Microsoft OLEDB Provider for ODBC Drivers error
'80040e07' [Microsoft][ODBC SQL Server Driver][SQL
Server]Syntax error converting the nvarchar value "Admin!d'
to a column of data type int./index.asp, line 5.
```

Fig. 4. Example of an error message returned by illegal queries

C. *Piggy Backed Queries*

In Piggy Backed Queries, extra queries are appended to the original one but are separated using the delimiter such as ";". In the process, all the queries are sent to and are processed by the database, including the original and the appended ones. The key objectives of this SQL attack are to gather information and to perform denial of service [7]. An example of such a query is illustrated in Fig. 5, where the "DELETE FROM" clause is appended and separated by ";". In this example, the customer_info column will be returned for the user who is logged in. Once this query is executed, the second one is processed, where The DELETE clause will be executed and the account of Albert will be deleted.

```
select customer_info from accounts where
login_id = ''admin'' AND pass = '123' ;
DELETE FROM accounts WHERE CustomerName =
'Albert';
```

Fig. 5. Example of a Piggy Backed Query

D. *Stored Procedures*

Stored procedures are SQL codes that are saved in the database as subroutines. The codes can be reused dynamically based on the user inputs. Even though the SQL codes are hidden, injection codes can be crafted to avoid authentication using stored procedures. Attackers can also use piggy back techniques to insert codes that can even shut down the whole database server.

E. *Union Query*

The intent of this attack is data extraction, where the UNION statement is used to add another query to the original one. It can also be used with the illegal queries attack. Once information about the database is obtained, the malicious query is crafted to extract the information wanted. The result is appended with the original query, forcing the database to return the wanted information [8]. An example of a Union query attack is illustrated in Fig. 6, where the Union query requests credit card information for a particular account number. When the database processes the whole query, the web application will return the credit card information instead of authenticating the user.

```
Username: ' UNION SELECT cardno FROM CreditCards WHERE accno=101 --
Password:
```

Fig. 6. Union query attack

F. *Alternate Encodings*

The aim of this attack is to remain undetected from intrusion detection systems (IDS) that monitor a user's input in order to determine if the text contains SQL injection code. In case such IDS detect injections, the text is dropped before reaching the database. Therefore, to bypass those systems, encoding of SQL keywords are changed into alternative ones, like hexadecimal, Unicode or ASCII [8]. For example, instead of writing "SHUTDOWN" in the injection string, "char(0x73687574646f776e)" is written [9]. This value can escape from the IDS, and when the database processes it, it will yield the same action as "SHUTDOWN".

G. Inference

This SQL attack aims to determine the database schema in web applications where there is no information produced by illegal queries [8]. The inference attack can be categorised into two types:

- Blind injection:

SQL injections codes statements that ask true or false questions to the database and if the query results are true, the application will continue to operate normally. However, if the query resolves to false, the application may operate differently, although there are no clear error messages. In the latter case, the web application is vulnerable to SQL injection, and further queries can be applied to determine the nature of the database [10].

- Timing Attacks:

Information is obtained based on the delay involved in responding to the query by the database server. The injected query contains codes that make the database wait for a specific amount of time based on a particular condition. If the condition is true, the application is paused, and the response is delayed [10]. Based on the time taken, the attacker can determine the structure and obtain information about the database.

III. Related Works

Various works have been carried out around SQLi attacks to discuss techniques employed to detect SQL injection. Nevertheless, limited studies performed a comparative analysis on the SQL injection detection methods. Within published literature, a previous study [11] identified the different SQL injection attacks and discussed the detection mechanisms for those attacks. Within the same study, techniques were classified into two categories, notably, detection and prevention. The paper ended with comparing the requirements needed to deploy such techniques and it was found that improvements in the abilities of current techniques are needed in order to more effectively stop SQL injection attacks. As key limitation, the attacks that each technique can efficiently detect and prevent were not discussed within the paper. Another study [12] surveyed SQL injection attacks and described the goal and purpose of each one of them. The study also compared machine-learning detection techniques proposed by previous researchers in terms of performance and precision. Following the comparative analysis, it was found that existing techniques are not completely successful and that some of them are impractical since they do not address all types of SQL attacks. In another research [13], the dangers of SQL injection attacks were discussed, in addition to the most common types of SQL attacks. The comparison performed focused on analysing the SQL injection detection and prevention techniques with respect to attack types. Moreover, another previous study [14] presented a taxonomy of SQL injection attacks where the different attacks were described and categorised. Nevertheless, limited comparison between the attacks and detection techniques were performed, as these were not the core focus of the study. As such, from existing literature, limited work has been done to comparatively analyse and determine the most effective SQL injection detection technique. Therefore, it becomes relevant to undertake this study.

IV. Methodology

In order to achieve the purpose of this paper and to perform a comparative analysis of the different SQL injection detection techniques, existing research databases were explored. These include IEEE Xplore Digital Library, ACM Digital Library, Google Scholar and ScienceDirect and these were scanned to obtain relevant results related to SQL injection detection techniques. Recent papers were favoured to have a state of the art attacks and detection methods available. The key terms used in the search process include “SQL injection”, “detection”, “SQLi attack”, “SQLia” and “SQLi”, among others, while also combining terms. Following an initial pool of results, filtering was conducted to assess relevance of the detection techniques and seven such detection techniques were selected for review and analysis.

These SQLi detection techniques were then thoroughly examined by referring to the published resources from the pool of selected papers, whereby also considering any further publications related to each SQLi detection technique. As part of the analysis, key points pertaining to their effectiveness in detecting different forms of SQLi attacks were noted. These results were eventually compiled into a table to compare SQLi detection techniques against SQLi attacks, to further analyse contents and their effectiveness. In the next sections, findings following the research are presented, to start with a review of the techniques available to detect SQLi attacks.

V. Detection of SQL Injection

As part of SQLi defence strategy, effective detection of SQLi attacks is important. Using the methodology described in the previous section, seven SQLi detection techniques were retrieved from published literature. These techniques provide further assurance against SQLi defence whereby attempt to detect such attacks in a timely manner in order to prevent SQLi attacks and minimise damages. These SQLi detection techniques are discussed below:

A. AMNESIA

A previous study proposed a technique and tool where the core logic involves differentiating between legitimate and malicious queries through a policy [10]. In the process of verification, both static and dynamic analysis are used. Within the static phase, an existing string analysis is performed with the aim to extract from the web-application’s code a model of all the query strings that could be generated by the application. On the other hand, the dynamic phase actively monitors whether dynamically created queries adhere against the statically built model. In case a query is not compliant with the model, it is marked as an SQL injection. The query is eventually blocked by the tool and administrators are alerted with relevant information. Although the technique and tool showed to be able to detect various instances of SQLi, AMNESIA cannot prevent stored procedure attacks because the subroutines are in-built in the database and are not generated in the models by the static phase [15].

B. CANDID

The Candidate Evaluation for Discovering Intent Dynamically (CANDID) operates by dynamically constructing the structure of programmer intended-queries and then comparing whether queries under analysis match the

structures of the programmer-intended query or not [15]. In case there is no match, the query is considered as an SQLi. Nevertheless, like AMNESIA, the CANDID technique was found to be ineffective against some injection attacks, such as stored procedures.

C. SQL Domain Object Model (SQL DOM)

There are several ways to send SQL statements to a DBMS and one of the ways is through a call-level interface (CLI). This method involves building dynamic SQL statements via string concatenation and string replacement but is prone to errors and is vulnerable to SQL injection attacks. To defend against such attacks, a previous study presented SQL DOM, which consists of two parts, notably an abstract object model and an executable named sqldomgen. [16]. The key goal of the Abstract Object Model is to construct an object model that could be utilized to build every possible valid SQL statement that would be executed during runtime. This was implemented through a constructor that takes all values as parameters such that the compiler would then need to ensure that all required values were supplied to a statement. On the other hand, the sqldomgen performs three key steps. The first step involves acquisition of the database schema, before iterating through the tables and columns present in the schema to then output numerous files that contain a strongly-typed instance of the abstract object model. Finally, the source files are compiled into a dynamic link library (DLL) that has the ability to detect errors in code that accesses database during compile-time, thus potentially also detecting injections.

D. SQLrand

The key idea behind the SQLrand technique involves changing SQL keywords into random keywords such that it is possible to detect and abort queries that include injected code [17]. In other words, SQL injection attacks would not be executed because the standard SQL keywords would not be sent to the database and the execution will be terminated. In terms of operation of SQLrand, the standard SQL keywords are appended by a randomly generated integer that is not easily guessable by an attacker. In case any attacker tries an SQLi, the statement would result in an invalid expression as the SQL keywords appended with the correct random integer are expected to be missing from the injected statement. The implementation of this technique consists of a proxy between the client's browser and the database. It is the role of proxy to convert the randomised keywords into the standard keywords understood by the database. When the proxy detects that the receiving query does not contain the random integer, it rejects it. However, as a recent study [18] showed, the security level of SQLrand is as strong as the randomly generated key. If the key is compromised, then the web application is susceptible to injection attacks.

E. Positive Tainting

This technique is based on the dynamic tainting approach, which is a popular one for addressing security issues pertaining to input validation [18]. The traditional dynamic tainting approaches work by marking certain untrusted data such as user input as tainted to then monitor the flow of tainted data during runtime. In case of issues, the tainted data is then prevented from being used in potentially harmful ways, thus preventing compromising the security of the system. The positive tainting approach makes various conceptual and practical enhancements over traditional dynamic-tainting approaches by taking advantage of the key features of SQL injection attacks. As compared to traditional tainting, the positive tainting approach operates by identifying, marking, and monitoring trusted data instead of untrusted data. In other words, this technique works by detecting reliable and trustworthy data, to then apply dynamic tainting to monitor the data during runtime while ensuring only trusted data are allowed to form the SQL statement. Finally, syntax-aware evaluation is carried out before the database executes the query.

F. Fault Injection and Behaviour Monitoring

The fault injection approach to SQL injection detection involves a dynamic analysis process utilized for software verification and software security assessment. In the process, specially crafted malicious input patterns are utilized as input data on purpose thereby enabling developers to observe the behaviour of software under attack. This approach has been used in a previous study to detect vulnerabilities in web applications by observing the outputs following injection of specially crafted SQL statements [19]. In the same study, a tool called Web Application Vulnerability and Error Scanner (WAVES) was proposed, that implements this approach. In this tool, the interface is in the form of a crawler that enables black box dynamic analysis of Web applications. Through the use of a complete crawling technique, all data entry points of a web application are identified, after a reverse engineering process. Eventually, using a self-learning injection knowledge base, fault injection techniques are applied in order to detect SQL injection vulnerabilities.

G. Machine Learning-Based Detection

Recently, Machine Learning (ML) has been employed by different studies in order to detect SQLi attacks [20]. ML is a sub-branch of AI where machines process data and learn on their own, without constant human supervision. Among the studies involving the application of ML to detect SQLi attacks, previous research [21] presented a Hybrid architecture (HIPS) that aims to detect web application attacks including SQLi attack. In this study, a method was utilized to dissect HTTP request in order to detect anomalies and involved the integration of an ML classifier and a firewall inspection engine based on attack signature. In addition, within SQL-IDS, neural networks have been used to detect SQLi whereby involving application of techniques such as Back Propagation Neural Network in order to detect 7 types of SQLi, notably, tautology, illegal/logically incorrect queries, piggy-backed query, Union query, stored procedures, inference and alternate encoding [22, 23]. Evaluation conducted in the same studies achieved an overall accuracy of 96.8% and 95% respectively for detecting such attacks. Furthermore, another study proposed an ML algorithm to create new rules for a network firewall, with the intent to differentiate between malicious and normal traffic, during which SQLi attacks can be detected [24].

VI. Comparative Analysis

In order to compare and analyse the effectiveness of the different detection techniques, published literature was thoroughly reviewed and analysed. This includes different related works on SQLi attack detection [11, 12, 25, 26] and articles referenced in the previous sections in this study. As discussed in the methodology section, this helped to compile Table I, which summarises the effectiveness of SQLi detection techniques against SQLi attacks. In the same table, effectiveness of a technique was classified into three categories, notably, fully effective, partially effective and not effective (none).

Findings revealed that Machine Learning Based Detection and Positive tainting were the only techniques that have showed the capability to fully detect the SQLi attacks investigated in this study. Both techniques employ a similar dynamic approach, using trusted data as a reference to avoid SQL injection. Interestingly, fault injection and behaviour monitoring can only partially detect the different SQLi attacks [11, 12]. The reason is that the technique uses a web crawler to identify all entry points of a web application and therefore, the effectiveness of this technique depends mainly on the web crawler. If the crawler could detect all pages containing input forms, the forms would be tested for SQLi vulnerabilities. However, the web crawler cannot discover all pages and all forms even if configured to have the same behaviour as web browsers since they cannot bypass all warnings or popups the web application has [19].

In terms of attacks, it could be noted that the SQLi attack through stored procedures is the most challenging one to detect as these are built within the database. As such, it is difficult for a detection technique to distinguish if a malicious SQL query could be interpreted as an injection by stored procedures. Given that there are various DBMS, which also have multiple versions, each of them may have different vulnerabilities in relation to stored procedures, which may be difficult to accurately detect. This attack in particular could not be detected by techniques including AMNESIA, CANDID, SQL DOM and SQLrand, but can be detected through positive tainting and machine learning.

The summary of the effectiveness of the techniques investigated is depicted in Table II. From the same table, it could be deduced that the most effective techniques for detecting SQL injection attacks involve the application of machine learning techniques as well as positive tainting, followed by AMNESIA and SQL DOM. On the other hand, CANDID and fault injection seem to be the least effective ones due to their limitations in fully detecting various types of SQLi attacks. As such, similar to a previous work [12], it can be seen that some detection techniques do not address all types of SQL attacks and, are therefore impractical to use. Nevertheless, as OWASP suggests, defensive coding along with those complementary detection and prevention techniques are expected to significantly increase security against SQLi attacks [3].

TABLE II. SUMMARY OF TECHNIQUES' EFFECTIVENESS

SQLi Detection Techniques	Fully	Partially	None
AMNESIA	6	0	1
CANDID	1	5	1
SQLDOM	6	0	1
SQLrand	4	0	3
Fault Injection	0	7	0
Machine Learning Based Detection	7	0	0
Positive tainting	7	0	0

Even though this study derived some key insights related to detection of SQLi attacks, it is also limited in different ways. Firstly, findings were derived following analysis of published literature and a better approach would be to practically apply the techniques individually to investigate their effectiveness and accuracy by also using some metrics. In addition, the application of machine learning could be better investigated since accuracy and effectiveness are expected to vary depending on different factors such as algorithm being used, involved dataset, and trained model, among others.

VII. Conclusion

This paper conducted a review and comparative analysis of the different SQL injection detection techniques, with the aim to detect SQLi attacks in an effective manner and enhance security of web applications. As part of this paper, seven SQLi detection techniques were reviewed and these include AMNESIA, CANDID, SQL DOM, SQLrand, Positive tainting, fault injection and behaviour monitoring, as well as machine learning based detection. These detection techniques were analysed to understand their effectiveness against

TABLE I. EFFECTIVENESS OF SQLI DETECTION TECHNIQUES AGAINST SQLI ATTACKS

SQLi Attack	SQLi Detection Technique						
	AMNESIA	***CANDID***	***SQL DOM***	***SQLrand***	***Positive Tainting***	***Fault Injection & Behavior Monitoring***	***Machine Learning Based Detection***
Tautologies	Fully	Fully	Fully	Fully	Fully	Partially	Fully
Illegal Queries	Fully	Partially	Fully	None	Fully	Partially	Fully
Piggy Backed Queries	Fully	Partially	Fully	Fully	Fully	Partially	Fully
Stored Procedures	None	None	None	None	Fully	Partially	Fully
Union Query	Fully	Partially	Fully	Fully	Fully	Partially	Fully
Alternate Encodings	Fully	Partially	Fully	None	Fully	Partially	Fully
Inference	Fully	Partially	Fully	Fully	Fully	Partially	Fully

different forms of SQLi attacks, notably, tautologies, illegal queries, piggy-backed queries, stored procedures, union query, alternate encodings and inference. Results showed that positive tainting and machine learning based techniques are the most effective ones with the ability to detect the different forms of SQLi attacks investigated in this paper. Among the different kinds of SQLi attacks, stored procedures were found to be the most challenging to detect as these are built within the database. It was also found that most detection techniques do not address all types of SQL attacks and a combination of techniques can potentially yield more effective results. As future works, the limitations identified in this paper could be addressed. For instance, an approach involving practical application of each SQLi detection technique could be adopted to investigate their effectiveness against different forms of SQLi attacks.

REFERENCES

[1] Emergen Research, "Progressive Web Application Market," 2020. [Online]. Available: https://www.emergenresearch.com/industry-report/progressive-web-application-market. [Accessed 22 Apr 2022].

[2] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh and R. Al-Qassas, "A systematic review of detection and prevention techniques of SQL injection attacks," *Information Security Journal: A Global Perspective*, pp. 1-14, 2021.

[3] OWASP, "OWASP Top Ten," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 10 Jan 2022].

[4] CWE, "2021 CWE Top 25 Most Dangerous Software Weaknesses," 2021. [Online]. Available: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html. [Accessed 10 Jan 2022].

[5] M. Borade and N. Deshpande, "Extensive Review of SQLIA's Detection and Prevention Techniques," *International Journal of Emerging Technology and Advanced Engineering,* vol. 3, no. 10, pp. 614-626, 2013.

[6] S. Choudhary, A. Jain and A. Kumar, "A Detail Survey on Various Aspects of SQLIA," *International Journal of Computer Applications,* vol. 161, no. 12, 2017.

[7] J. Singh, "Analysis of SQL Injection Detection Techniques," *Theoretical and Applied Informatics,* vol. 28, no. 1 & 2, pp. 37-55, 2017.

[8] M. Medhane, "Efficient solution for SQL injection attack detection and prevention," *International Journal of Soft Computing and Engineering (IJSCE),* vol. 3, pp. 396-398, 2013.

[9] G. Yiğit and M. Arnavutoğlu, "SQL Injection Attacks Detection & Prevention Techniques," *International Journal of Computer Theory and Engineering,* vol. 9, no. 5, pp. 351-356, 2017.

[10] W. Halfond and A. Orso, "AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005.

[11] A. Tajpour, M. Massrum and M. Heydari, "Comparison of SQL injection detection and prevention techniques," in *2010 2nd International Conference on Education Technology and Computer* , 2010.

[12] B. Nagpal, N. Chauhan and N. Singh, "A survey on the detection of SQL injection attacks and their countermeasures," *Journal of Information Processing Systems,* vol. 13, no. 4, pp. 689-702, 2017.

[13] W. Halfond, J. Viegas and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proceedings of the IEEE international symposium on secure software engineering (Vol. 1, pp. 13-15),* 2006.

[14] A. Sadeghian, M. Zamani and S. Abdullah, "A taxonomy of SQL injection attacks," in *2013 International Conference on Informatics and Creative Multimedia*, 2013.

[15] P. Bisht, P. Madhusudan and V. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks," *ACM Transactions on Information and System Security (TISSEC),* vol. 13, no. 2, pp. 1-39, 2010.

[16] R. McClure and I. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," in *Proceedings of the 27th International Conference on Software Engineering*, 2005.

[17] S. Boyd and A. Keromytis, "SQLrand: Preventing SQL injection attacks," in *International conference on applied cryptography and network security*, Berlin, Heidelberg, 2004.

[18] W. Halfond, A. Orso and Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL injection attacks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, 2006.

[19] Y. Huang, S. Huang, T. Lin and C. Tsai, "Web application security assessment by fault injection and behavior monitoring," in *Proceedings of the 12th international conference on World Wide Web* , 2003.

[20] I. Jemal, O. Cheikhrouhou, H. Hamam and A. Mahfoudhi, "Sql injection attack detection and prevention techniques using machine learning," *International Journal of Applied Engineering Research,* vol. 15, no. 6, pp. 569-580, 2020.

[21] A. Makiou, Y. Begriche and A. Serhrouchni, "Improving Web Application Firewalls to detect advanced SQL injection attacks," in *2014 10th International Conference on Information Assurance and Security*, 2014.

[22] N. Sheykhkanloo, "SQL-IDS: evaluation of SQLi attack detection and classification based on machine learning techniques," in *Proceedings of the 8th International Conference on Security of Information and Networks*, 2015.

[23] N. Sheykhkanloo, "A learning-based neural network model for the detection and classification of SQL injection attacks," *International Journal of Cyber Warfare and Terrorism (IJCWT),* vol. 7, no. 2, pp. 16-41, 2017.

[24] R. Verbruggen and T. Heskes, "Creating firewall rules with machine learning techniques," Nijmegen Netherlands: Kerckhoffs institute Nijmegen, 2014.

[25] D. Loughran, M. Salih and V. Subburaj, "August. All About SQL Injection Attacks," *Journal of The Colloquium for Information Systems Security Education,* vol. 6, no. 1, pp. 24-24, 2018.

[26] D. Kindy and A. Pathan, "A Detailed Survey on Various Aspects of SQL Injection in Web Applications: Vulnerabilities, Innovative Attacks, and Remedies," *International Journal of Communication Networks and Information Security,* vol. 5, no. 2, p. 80, 2013.