International Journal of Applied Engineering Research ISSN 0973-4562 Volume 15, Number 6 (2020) pp. 569-580

SQL Injection Attack Detection and Prevention Techniques Using Machine Learning

Ines Jemal[1], Omar Cheikhrouhou[2], Habib Hamam[3] and Adel Mahfoudhi[4]

[1]*University of Sfax, ENIS, CES, LR11ES49, 3038 Sfax, Tunisia.*
E-mail: ines.jemal@stud.enis.tn
[2]*University of Sfax, ENIS, CES, LR11ES49, 3038 Sfax, Tunisia.*
University of Mounastir, ISIMA, Mahdia, Tunisia.
E-mail: o.cheikhrouhou@tu.edu.sa
[3]*Faculty of Engineering, University of Moncton, NB, E1A 3E9, Canada.*
E-mail: habib.hamam@umoncton.ca
[4]*University of Sfax, ENIS, CES, LR11ES49, 3038 Sfax, Tunisia.*
E-mail: a.mahfoudhi@tu.edu.sa

Abstract
Web application attacks are incessantly increasing in number and in severity. The big data available on the internet motivates hackers to launch new kind of attacks. In this context, intensive research on web application security have been conducted. The most dangerous attack that target web applications is the Structured Query Language Injection (SQLI). This attack represents a serious threat to the web applications. Several research works have been conducted to mitigate this attack either by preventing it from an early stage or detecting it when it occurs. In this paper, we present an overview of the SQL injection attack and a classification of the newly proposed detection and prevention solutions. We classify the different attack sources, goals, and types. Moreover, we discuss and classify the most important and recent proposed solutions to mitigate this attack especially those based on ontology and machine learning.

Keywords: SQL injection, Web security, Ontology, Machine Learning.

I. INTRODUCTION

The increase in the development and spread of the web applications has also lead to an increase in the number and severity of the web attacks. According to Statista [1], in 2018, 953 thousand web attacks were blocked on a daily basis, up from 611 thousand daily blocked attacks in the previous year. According to the Open Web Application Security Project (OWASP) [2], the injection vulnerability continues to be the most found vulnerability in web applications.

The Structured Query Language Injection (SQLI) attack is considered as the most dangerous attacks of the injection category because it compromises the main security services: confidentiality, authentication, authorization and integrity [3]. Roughly speaking, SQLI attack consists in injecting (inserting) malicious SQL commands into input forms or queries to get access to a database or manipulate its data (e.g. send the database contents to the attacker, modify or delete the database content, etc.) [4], [5].

Indeed, today, most of the web applications use a back-end database to store data collected from the users and/or to retrieve information selected by the users. Interaction with these users are generally achieved through forms and cookies. Hackers try to exploit this characteristic by injecting malicious code into these user inputs that will be used later to construct the SQL queries. Improper validation of the user inputs can lead to the success of the SQLI attack and, therefore, can have catastrophic consequences such as the deletion of the database or the gathering of sensitive and confidential data of the web application clients.

Due to its sensitive impact, several works have addressed the SQLI attack. Some of these works try only to detect the SQLI once occurred. Other works try to prevent it before occurring. Although, several techniques are proposed to fight against SQLI attack, none of these solutions have addressed the full scope of the attack. Therefore, there is no solutions that can prevent or detect all the different types of the SQLI attack. Recently, researchers try to benefit from the machine learning techniques to propose more sophisticated solutions.

In this paper, we present a survey on the SQL injection attacks. We present the main attack sources, types and goals. Moreover, the main proposed solutions that address this attack were discussed and compared.

The main contributions of this paper are as follows:

1) An overview of the SQLI attack was presented. The different attack sources, goals and types are described and discussed.
2) A classification of the different SQLI attack detection and prevention countermeasures are presented and discussed.
3) A comparative table between the different proposed SQLI attack countermeasures was presented.
4) Newly proposed solutions, such as those based on ontology and machine learning, are described and discussed.

The remainder of this paper is as following. In Section II, we present a description of the SQLI attack by enumerating its possible sources, goals and types. Then, in Section III we classify the different existing countermeasures that was proposed to either detect or prevent the SQLI attack. Finally, we give a discussion about the presented solutions and we conclude the paper.

II. SQLI ATTACK OVERVIEW

In this section, we present a general overview of the SQLI attack. We start by discussing the SQLI sources, then classify their goals and types. Table I classifies and summarizes the main ideas and points discussed in this section.

A. SQLI Attack Sources

SQL injection vulnerabilities may be found in any application parameter that can be used in a database query. The authors in [6] cited four sources, through which the SQL Injection Attack (SQLIA) can start. These sources are user input, cookies, server variables and stored injection.

- **Injection through user input**: Web applications, generally, use forms to collect data from users (such as registration, login, etc.) or to permit users to specify the data to be retrieved (such as search, adapted view, etc.). These forms containing ”text field” could be exploited by attackers to inject malicious code, which results in gaining an indented data (retrieve secret data, etc.) or making an indented actions (manipulate database, etc.). Common fields are Login Name, Password, Address, Phone Number, Credit Card Number, and Search.
- **Injection through cookies**: Recent web applications use cookies to store users preferences. Cookies are files stored on the client machine, which contain state information generated by the web applications. An attacker could embed malicious code into the cookies contents stored in his computer, and therefore, putting web application using the cookies contents to build SQL queries vulnerable to attacks [7].
- **Injection through server variables** : Server variables are a set of parameters that contain network headers, HTTP metadata, and environmental variables. Generally, web applications use these server variables for auditing usage statistics and identifying browsing trends. If these variables are stored to a database without validation, attackers can exploit this vulnerability by placing an SQLIA directly into the server variables.
- **Stored injection**: In stored injection (called also second-order injection), attackers embed malicious inputs into a database to indirectly launch an SQLIA each time that input is used. The following code, shows an example of second-order SQL injection. In this example, the attacker as a normal user of the website, firstly registers to the application with a seeded username like ”admin’- -”. Then, the attacker will try to change his password. The SQL query to change a user password has generally the following form:

```
queryString=”UPDATE users SET password=’”
+ newPassword +”’ WHERE userName=’” +
userName + ”’ AND password=’” +oldPassword
+ ””””
```

Assume that newPassword and oldPassword are “newpwd” and “oldpwd”, which are chosen by the attacker, the query that will be sent to the database is the following:

```
UPDATE users SET password=”newpwd”
WHERE userName= ”admin” - - ” AND
password=”oldpwd”
```

Because ”- -” is the SQL comment operator, everything after it is ignored, the result of this query is that the database changes the password of the administrator (“admin”) to an attacker-specified value.

B. SQLI Attack Goals

Hackers can have different intend and goals for launching the SQLI attack. The main SQLI attack goals are:

- **Identifying injectable parameters**: As a first step, hackers try to identify which parameters could be used to inject malicious code. These parameters could be present in one of the sources described in Subsection II-A. More precisely, these parameter could be a ”username” field in a form, a ”card number” in a cookie, etc. An attacker can modify the logic of the statement by injecting SQL code, so that when executed it performs another action. For example, injecting a single quote that is used in SQL to delimit the start or end of a string value could disrupt the pairing of string delimiters and generate an application error, indicating a potential vulnerability to SQL injection.
- **Performing database fingerprinting**: To construct a query format supported by the target database engine, the attacker needs to know the database finger-print. Database finger-print is the information that identifies a precise type and edition of a database system. Each database system uses a different proprietary SQL language syntax. For example, Microsoft SQL server uses T-SQL while Oracle SQL server uses PL/SQL. As a consequence, the attacker must first find out the type and version of the database used in a web application, and then craft malicious SQL input for that database. Moreover, default vulnerability associated with that version of database can be exploited by attackers.
- **Determining database schema**: To successfully extract data from a database, the attacker needs to know the database schema information, such as table names, column number and names, and column data types. The database schema is used by hackers to create an accurate consequent attack with the purpose of extract or modify data from database. Figure 1 presents an error message returned by the database system that shows different information related to the database schema (such as number and name of columns) and system (such as ODBC). These pieces of information can be exploited by hacker to construct a successful SQLI attack.

```
Error # -2147217887 (0x80040E21)
ODBC driver does not support the requested properties.
SELECT tsk.TaskID, tsk.Title, tsk.Comments, usr.FirstName, usr.LastName,
pri.PriorityName, sta.StatusName, 0 As CommentCount, tsk.Created FROM tblTask tsk
INNER JOIN tblUser usr ON tsk.UserID = usr.UserID INNER JOIN tblTaskPriority pri
ON pri.PriorityID = tsk.PriorityID INNER JOIN tblTaskStatus sta ON sta.StatusID =
tsk.StatusID WHERE tsk.TaskID = 1' AND tsk.Active <> 0 AND tsk.Archive = 0
```

Fig. 1: verbose error message

- **Extracting data**: These types of attacks employ techniques in order to extract data values from the database. This attack presents critical risk to web application as extracted information could be sensitive and highly top secret to the web application (example getting customer bank information). Attacks with this intent are the most common type of SQLIA.
- **Database alteration**: The goal of these attacks is to alter or change information in a database. For illustration, a hacker can pay much less for an online product by modifying its price, which is generally stored in a database. Another possible attack, consists in adding a malicious link in an online discussion database to commence succeeding Cross-Site-Scripting attacks.
- **Performing denial of service**: This attack intend is to deny service to other users and can have different form, such as shutdown the database of a web application, locking or dropping database tables, etc.
- **Bypassing authentication**: The aim of this attack is to bypass the authentication mechanisms of the web application. If the intruder succeed to launch such attack it could take the rights and privileges of another user, generally with high rights and privileges [6].
- **Executing remote commands**: Remote commands are executable code resident on the compromised database server. These commands can be stored procedures or functions available to database users. In this type of attacks, hackers attempt to execute arbitrary commands on the database, which can lead to denial of service by executing the shutdown command or database disruption.
- **Performing privilege escalation**: These attacks try to escalate the privileges of the attacker taking advantage of some implementation errors or logical flaws in the database. As opposed to bypassing authentication attacks, these attacks focus on exploiting the database user privileges. This attack can have a critical consequence especially when the attacker gains the root privilege.

C. SQLI Attack Types

SQLI attack can have several types and forms [6]. In this section, we discuss and classify the main SQLI attack types, which are:

- **Tautologies:** The general goal of a *tautology-based attack* is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages are to bypass authentication pages and extract data. In the following example, an attacker submits ” ’ or 1=1 - -” for the login input field (the values submitted for the other fields are irrelevant). The resulting query is:

```
SELECT accounts FROM users WHERE login=”
or 1=1 - - AND pass=” AND pin=”
```

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology, which results in a successful authentication of the attacker and the attacker can show all the accounts saved in the database.

- **Blind SQL injection:** *Blind SQL injection* is a type of SQLI attack that asks the database true or false questions and determines the answer based on the application’s response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
- **Union query:** In union query attack, the attacker uses the UNION operator to join a malicious query to the original query. The result of the malicious query will be joined to the result of the original query, allowing the attacker to obtain the values of columns of other tables. The following query is an example of a union query SQL injection attack:

```
SELECT accounts FROM users WHERE login=”
UNION SELECT cardNo from CreditCards where
acctNo=10032 - - AND pass=” AND pin=
```

Although, the original first query returns the null set, whereas the second query returns data from the ”CreditCards” table.

- **Piggy-backed query:** In piggy-backed query attack, the attacker intends to inject additional queries to extract data, modify or add data. Attackers inject additional queries to the original query, and as a result the DBMS receives multiple SQL queries. The following query is an example of a piggy-backed query SQL injection attack:

```
SELECT * FROM userDetails WHERE userid =
’12’ and password = ’cle’; drop table userDetails ;
```

- **Stored procedures:** In stored procedures, the attacker aims to run stored procedures already saved in the database. Indeed, most existing databases are extended with a standard set of implemented functions called stored procedures that allow even the interaction with the operating system. These stored procedures are generally invoked by developers in their codes to avoid re-writing standard functions. Therefore, an attacker can exploit this property and once determines the web application backend database, SQLIAs can be crafted to execute stored procedures existing in the determined database, including even procedures that interact with the operating system [6].
- **Alternate encoding:** In alternate encoding, the attacker tries to conceal the injected text in order to avoid detection by defensive coding practices and automated prevention techniques. More precisely, alternate encodings are enabling techniques that allows attackers to escape detection countermeasures. These evasion techniques are widely used by intruder because they know that most IDS scan the query for certain known ”bad characters”, such as single quotes and comment operators. By changing the character encoding a malicious code can evade detection mechanism. The following query shows an example of alternate encoding SQLIA:

```
SELECT accounts FROM users WHERE login="legalUser"; exec(char(0x73687574646f776e)) - - AND pass="" AND pin=
```

The char() function, cited in this example, takes an integer or hexadecimal encoding of a character as input and returns the character spelling. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "SHUTDOWN". This result in database shutdown and might lead to denial of service attack.

- **Illegal/logically incorrect queries:** In illegal incorrect queries, attackers intend to input a manipulated query into the database to generate an error message which contains some information about the cause of the error, in general error message contains give an idea what's look like of what the database schema looks like. Figure 1 shows an example of such error message.

Table I summarizes the different SQLI attack sources, goals and types.

TABLE I: SQLI Attack Sources, Types and Goals classification

Parameter For Classification	Categories
Attack Sources	User input Cookies Server variables Second order injection
Attack Goals	Database finger printing Analysing schema Extracting data Amending data Executing dos Equivocating detection Bypassing authentication Remote control Privilege intensification
Attack Types	Tautology Illegal/logically incorrect queries Union query Piggyback query Stored procedure Inference Alternate encoding

III. CLASSIFICATION OF SQLI ATTACK DETECTION AND PREVENTION TECHNIQUES

A large number of techniques have been proposed to address the SQLI attack, as it presents the most widely spread attack. Proposed techniques can be classified according to the web application life cycle. Some techniques address the secure development of the web application, others address security of the web application at runtime. In this paper, we have mainly addressed these later techniques and we give a classification according to the concepts used in these techniques.

A. Query-model based SQLI countermeasures

CANDID "**CAN**didate evaluation for **D**iscovering **I**ntent **D**ynamically" [8], is a tool that permits to check the user generated query structure in a flexible manner at the runtime with those that are self-evidently non-attacking. The idea of the CANDID approach is to dynamically generate the structure of the programmer-intended query using candidate benign inputs. Then, this benign query structure is compared with the structure of the user constructed query with user inputs. The main weakness of CANDID is that it cannot deal with external functions call and that it fails to transform some particular code (such as code containing loop). Inspired by the CANDID approach, the authors in [9] proposed SQLStor an SQLI detection scheme that is based on semantic comparison. The semantic comparison is done by comparing the syntax tree structure of an original query to the syntax tree structure of a constructed benign query. The scheme, first, generates a *Benign Query* from the *Original Query* generated by the application. This is done by replacing user inputs to the query with benign inputs. Then, the scheme compare the syntax tree structure of both queries. If the syntax trees of both queries are equivalent, then the queries are inducing equivalent semantic actions and therefore the original query is considered as a safe query, else it is considered as malicious query. One limitation of the SQLStor scheme is that it addresses only the stored procedure type of the SQLI attack. The authors in [10] presented an approach based on XML (eXtensible Markup Language) to detect only the tautology type of the SQLI attack. The approach contains an XML file maker that intercepts the user query and converts it into XML format. Then, the XML file pass the user credential to an Xschema validator. This later compares the produced user query with the already defined legitimate query in the Xschema file. The user is granted access to the system only if both queries matches, otherwise the access to system is blocked. Comparing to other existing approaches used to detect SQLIA, this method is more suitable in sense of execution time but still limited for other types of SQLIA. Another drawback of this approach is the use of predefined pattern and therefore a hacker can inject malicious patterns that are not addressed by the developers. The authors in [11], [12] focused on the prevention of the injection attacks that target the database management system (DBMS) behind the web applications by embedding the protections in the DBMS itself. This permits to avoid the semantic mismatch between how SQL queries are believed to be executed by the DBMS and how they are actually executed. The proposed approach called SElf-Protecting daTabases preventIng attaCks (SEPTIC) runs in three modes, one for training (training mode) and two for operation mode (prevention mode and detection mode). The output of the training mode is a set of valid query models stored in SEPTIC. SEPTIC attacks detection mechanism consists in comparing each query structure with the stored query models. If there is no match, an SQLI attack was detected. In the prevention mode, the queries considered as insecure are dropped and the DBMS stops the query processing. In the detection mode, these queries are executed, not dropped. SEPTIC was implemented in MySQL and evaluated experimentally within web applications written in PHP and Java/Spring.

B. Obfuscation based SQLI countermeasures

The authors in [13] presented a technique of Obfuscation/deObfuscation in order to detect SQLIA

before routing the query to DMBS. This Obfuscation technique based on two phases static stage, which modifies queries of application in a obfuscation form and dynamic stage, which aims to combine the obfuscation queries with the input data of user. The original query is retrieved in step of deObfuscation. In Random4 [14], the authors suggested using an encryption algorithm based on randomization to prevents illegal access to the database. The random4 algorithm is based on randomization and it converts the user input into a cipher text incorporating the concept of cryptographic salt. The drawback of this solution is that it limits the user input possible values. In [15], a scheme called SQLshield was proposed to prevent SQL injection attacks in web applications. The idea of the SQLshield is to randomize the user input data before the SQL query is executed at the database server. The randomization technique used in SQLshield modifies the user input data without diverting the resultant SQL query from its programmer-intended execution. The randomization is achieved by appending a random key to some specific tokens of the user input. Comparison with other schemes such as CANDID [8] shows that SQLshield outperforms these approaches. The drawback of SQLshield is that it requires the intervention of user to determine which user input contribute in the SQL query and therefore need to be randomized. Moreover, the execution time of the algorithm depends on the length of the user input and therefore might be ineffective for longer length of user input. To overcome the need of user intervention, the authors in [16] proposed AutoRand for Automatic Keyword Randomization to prevent SQLI attacks. AutoRand transforms all SQL keywords in a Java program into a randomized value. Before a query execution, AutoRand checks that each SQL command is randomized to the appropriate value. If it is the case the query is de-randomized and executed. One weakness of AutoRand is its limitation to Java program.

C. Monitoring and Auditing Based SQLI countermeasures

Preventing SQL Injection Attacks based on Query Optimization Process (PSIAQOP) [17] tries to prevent SQLI by optimizing query of user input. The process begins by analyzing the source code used in the web application and vulnerable in execution, then optimizing these vulnerable queries. The optimization engine generates a set of valid execution in accordance with the heuristic rules. Finally, the hotspot (vulnerable part of code) is replaced by the web application code with its optimized query.

In [18], the authors discussed a novel model based on the dynamic analyzer. Dynamic analyzer plays the role of the user who demands a page and analyzing it: the Tester examine the page if it is vulnerable or not then generate a response to the user. If the response shows that the page is not vulnerable, the request is processed, otherwise, the request is rejected and the knowledge base must be updated by adding the vulnerabilities found in the page and rules.

The authors in [19] proposed a mechanism called Cloud Computing SQLIA Detection (CCSD) applied to cloud computers in order to distinguish SQLIA from legitimate queries. This approach consists of five units: 1- Data Collector: plays the role of capturing information, stacks of executions, and the analysis tree in a query, 2-Modeler: to analyze for role requests, browse trees and compress the execution stack captured by data collector by functions hashes, 3- Repository: Repository saves compressed piles and trees, 4- Comparator: checks if the request is not vulnerable it sends the request to the database otherwise it rejects this request and sends an answer to the log, 5- Log: saves the request sending by the Comparator. CCSD is characterized by high accuracy rate, low false positive rate, and low time consumption.

In [20], the authors defined a tool for DEtection and PREvention website from Input Validation Attacks. It consists of two modules detection and Prevention. Detection is accomplished by using multi-layer defense mechanism: Removal of illegal character, Validation through regular expression check and IP address tracking. Prevention is defined by using Mutation testing and a graphical user interface, which the user can select an attack (SQLI or Cross-site Scripting or Buffer Overflow) against which he wants to protect website from it, then the website is tested for the vulnerability selected.

The authors in [21] presented a Web Defacement and Intrusion Monitoring Tool (WDIMT). This tool aims to detect defacements, degradations and intrusions on the web and allows changes and deletion of files containing intruders. WDIMT also proposes a solution to recover the original content of the web page, these commands are executed only after the use of the Linux terminal command.

In [22], the authors presented JoanAudit: a new tool of security slicing for auditing java web services and web applications from common injection vulnerabilities. The process of JoanAudit allows to generate a vulnerability report and to extract control information and data dependency from the analysed program. This tool is characterized by its scalability to web systems and its simple configuration for injection vulnerabilities.

D. Entropy Based SQLI countermeasures

The authors in [23] proposed an automated testing approach, namely $\mu 4SQLi$. $\mu 4SQLi$ can produce effective inputs that lead to executable and malicious SQL queries. The idea is to produce inputs that bypass web application firewalls. The goal of this tool is to detect potential SQL vulnerabilities in a given web application. Inspired by the previous work, Kumar et al., proposed in [24] a Detection Block Model for SQL Injection Attacks. The proposed solution is also based on information theory. For this purpose, entropy for each possible query is computed and then saved in a database. Then, for each submitted query the entropy is computed and compared to the ones saved in database. The scheme was improved in [25] by adding a Message Authentication Code (MAC) derived from entropy. This prevents attackers from knowing the value of acceptable entropy. The authors in [26] also proposed a system based on information theory. It measures query entropy using token probability distribution. Then, during execution the system computes the complexity to identify any changes from the measured entropy. Dynamic

query with malicious inputs alters its intended structure and therefore its entropy level changes.

Combining the information theory with the machine learning, the authors in [27] proposed to defeat SQLI attack in authentication. The proposed scheme first generate a path-index of the SQL query. The path-index consists in replacing each keyword in the query with its corresponding index in an index table defined by the scheme. Then, the scheme compute the edit-distance between the run-time query and the developer intended query. If this distance in the threshold range the query is considered benign, otherwise it is considered as malicious. To determine the optimal value of this threshold, the authors used machine learning techniques. Tested on some scenarios the scheme gives a detection rate of more than 92%. However, the scheme complexity might introduce an overhead that delay the query execution. Moreover, the main drawback of this method is that it requires effective training phase and has a limitation with the use of maximum variables in a dynamic SQL query.

E. Ontology Based SQLI countermeasures

Ontology based solution try to benefits from the attack vocabulary and their semantic relations. Ontology is a good way to represent the vocabulary of an attack, which is not only restricted to its signature, but it also includes the characteristics, features, and any actors related to system. By using ontology, systems are able to capture the context of users input, which allows to design the defense mechanisms against web attackers. Ontology also offers an excellent comprehensive of metrics as key words and can provide a generic solution for various environments. Ontology can also classify the attacks, describes their signatures and their outcomes on systems. Creation of ontology for web applications is similar to building ontology for human anatomy which requires precision and systematic labelling for each component. Ontology can be considered as a set of concepts. Taking the example of ontology cited in [28], the knowledge objects or core concepts are defined as follows: class Assets: can be databases, class Methods: present the used strategy to reach a goal as antivirus, class Tools: can be a software that support methods, class Security Properties: present the security goals as confidentiality or integrity, class Vulnerabilities: present weakness that makes it possible for a threat to arise, class Threats: a potential violation of security, and class Notation: present the characteristics of attacks. In what follows, we present the main SQLI attack countermeasures based on ontology.

Denker et al. [29], [30] addressed the security of semantic web using Ontology Web Language (OWL) and Security Web Services using DARPA Agent Markup Language (DAML). The authors make a big conceptualization of the domain but use fewer attributes to define concepts. These ontologies need to assign more properties to each concept in order to set the attributes correctly. Kim et al. [31] developed a more general ontology that has more concepts than the one proposed by Denker [29]. It represents security instructions such as attacks mechanisms, protocols, algorithms and credentials. Moreover, the proposed ontology can be applied to any electronic resource. The authors used the Web Ontology Language (OWL) to create their security ontology that consists of several parts, like main security, algorithms, assurance or semantic web services. Dobson et al. [32] focused on the field of reliability requirements and revised their ontologies for requirements engineering that presented large number of concepts to model a specific domains as cyber attacks. Undercoffer et al. [33] used ontology to describe computer network attacks for distributed IDS. The proposed ontology stored the victim parameters for each attack, and then assigns the attack to a class. Moreover, the authors analyzed around 4000 vulnerabilities with their exploitation strategies and present several scenarios of uses cases with common attacks. The authors claim that ontology provides to IDS a widespread understanding of information issues. Raskin et al. [34] summarized a large variety of item sets with precise specification of security knowledge to improve prevention and reaction capabilities.

The main drawbacks of the previous cited works is that they only model the SQLI attack and did not precise how the prevention or the detection of the SQLI attack can be done based on these models. Moreover, they only model a subset of the SQLI attack types. To overcome these limits, Abderazed et al. [35] created an ontological model of attack detection that effectively captures the context of user input. The authors proposed two ontologies: ontology of attack and ontology of communication protocol. These ontologies are developed with OWL-DL through interaction with OWL GUI and using the Methontology framework [36], and they have been carried out by OntoCLean [37] to check their consistency. The proposed ontology of attack cover all attacks mentioned in the Common Vulnerabilities and Exposures [38] which includes the SQLIA. It captures the context source and the target of the attacks, the various techniques used by the hackers, the impact on the system components, the vulnerabilities exploited by the attacks, and the control in terms of policies to mitigate these attacks. The Security Web Application Ontology (SecWAO) [28] [39] proposed by Marianna Bush was integrated with Uml-based Web Engineering (UWE). It is an ontology used in the scope of Software Development Life Cycle, based on UML. It supports web developers to specify security requirements to make design decisions. This ontology used libraries for prepared statements to avoid database query injection, it provides relevant instances and relates them by different kinds of relationships such as: belongs to, uses, and depends on. In [40], the authors proposed a new paradigm called Network Security Situation Awareness (NSSA) in the domain of Internet-Of-Things (IOT). This work focuses on Detecting and Prediction attacks in application and network layer. The authors reflect four key domains to describe the IOT security situation: context, attack, vulnerability and network flow. They build the proposed ontology with OWL DL enriched by Semantic Web Rule Languages (SWRL) to enhance the reasoning ability of the NSSA model. These rules offer more meaningful power than OWL alone. The building ontology includes six top classes: Context, Sensor, Alert, Attack, Vulnerability and Netflow. They are enriched by the relation Object properties: hasVulnerability, exploited by, exploit, supplyInformation, reflect, generate, and reason. For the evaluation of their work, the authors considered

CVE-2013-0375 [38], a vulnerability that describes a subset of types of the SQLI attack. The main weakness of this solution is that it does not detect *zero-day* attacks, as the detection mechanism is based on the already described attacks on the ontology model.

F. Machine Learning Based SQLIA countermeasures

Machine Learning (ML) is a sub-field of artificial intelligence that gives machines the skills to learn from data and predicts the optimum model. ML techniques can be grouped into three main categories Supervised Learning, Unsupervised Learning and Reinforcement Learning. The Supervised Learning method is based on a set of input data X which deals to the result Y, while in the unsupervised learning, the system only has input data X without being told the expected output result Y. Recently, several works based on machine learning was proposed to detect the SQLI attack. These proposed solutions used different metrics to evaluate their ML models. However, *Accuracy* and *precision* are the two significant used metrics. These metrics define the ability and effectiveness of the proposed model to detect SQLI attacks. Accuracy and precision can be defined by these two equations.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Precision = \frac{TP}{TP + FP}$$

TN is the True Negative rate. It presents the number of correctly predicted normal requests.
TP is the True Positive rate. It presents the number of correctly predicted malicious requests.
FN is the False Negative rate. It presents the number of incorrectly predicted normal requests.
FP is the False Postive rate. It presents the number of incorrectly predicted malicious requests.

Table II presents the confusion matrix that describes the necessary metrics to evaluate the machine learning classifiers.

TABLE II: Confusion Matrix

	Predicted as Normal Request	Predicted as Malicious Request
Normal Request	TN	FN
Malicious Request	FP	TP

In [41], the authors presented Hybrid architecture (HIPS) to detect web application attacks including SQLI attack. They proposed a novel method to dissect the http request and to detect anomaly. This method defines collaboration between a machine learning classifier and a firewall inspection engine based on attacks signature. The authors used the classifier Bayesian Naif [42] and Bayesian Multinomial [42] to classify the malicious http request from legitimate http request. They used Total Cost Rate method (TCR) to extract features from Http Request and Mutual Information method to select the basic features which represent the input of the machine learning classifiers. The authors used Received Operating Characteristic (ROC) a standard performance measurement tool which allows reproducing a curve of true positive rate according to the false positive rate. To evaluate the HIPS, the authors introduced two parameters: λ to give more weight to the error that considers a legitimate request as a malicious request than other errors, and α threshold which represents an arbitration parameter between the result of Bayesian classifier and the result of the inspection engine. Despite HIPS achieved a good accuracy of 97.6%, it did not grant a solution to handle false negatives and false positives.

In SQL-IDS [43], [44], the authors proposed to use two different neural networks. In their first paper [43], they proposed to use Back Propagation Neural Network that aims to detect 7 types of SQLI Tautology, Illegal/logically incorrect queries, Piggy-backed query, Union Query, Stored Procedures, Inference, Alternate Encoding. Using a small dataset that contains only 13,000 URLs, the authors classified and tested one by one the different types of SQLIA. Their model achieved an overall accuracy of 96.8%. In their second paper [44], they proposed to use a Neural Network Based Model which has three elements a URL generator, a URL classifier, and a Neural Network model. The authors used a new and larger dataset than the first one to train and test their model. The achieved accuracy was 95%.

In [45], the authors proposed a machine learning algorithm in order to create new rules for a network firewall. These rules permit to differentiate between malicious and normal network traffic. The proposed methodology consists of the following steps: create data, extract features, combine data, training and testing the model. The authors used two computers on a same LAN that generates web traffic. They created their properly dataset of http request because they consider that the existing open dataset like KDD Cup [46] and DARPA [47] are old. Moreover, they used the command TCPDump in UNIX to capture all network traffic, in result a PCAP file was generate which contains all packets. This PCAP file was loaded into the Wireshark tool to visualize and inspect the data. After the features extraction from malicious and normal data, the authors combine both data to test the different machine learning techniques. To train and test the proposed model, they choose to use Decision Tree, SVM, Random Tree, Jribber, Neural Network, and Random Forest. To train, validate and test their neural network, the authors used 6 different datasets that contains 1876 malicious packets and 11444 normal packets. The scheme achieved 96.3%, 61.4% and 100% of correct responses for Piggy Backed Query, Union Query, and the rest types of SQLI attack respectively.

In [48], the authors proposed to use a Stacked AutoEncoder (SAE) based on protocol traffic classification. This work is based on Neural Network and Deep Learning and justify that SAE is more efficient in terms of features extraction and features selection than artificial neural networks methods. The work in [49] used Decision Tree in IDS systems to classify and detect SQLIA. The authors used the NSL-KDD [50] dataset to train and test their method that achieved 83.7% of accuracy. In [51], the authors used a Neural network Multi-layer Feed Forward to detect SQLI attacks. Even with a small dataset of 300 SQLI and 200 XSS attacks used to train and test the proposed neural network, this method achieved a low accuracy rate = 66.67%. In [52], the authors concentrated on detecting three types of SQLIA namely, alternate encoding, union query and tautology. They proposed

to use the Naive Bayes machine to extract and select key features from the dataset and Role Based Access Control mechanism to make the decision. This method achieved 93.3% of accuracy.

Based on graph theory and machine learning, the authors in [53] presented SQLiGOT: a novel SQLI attack detection methods that uses graph of tokens. SQLiGOT is mainly composed by a graph generator and the SVM classifier. After converting the SQL query into tokens, the generator construct a graph having as nodes the tokens and as weights the relation between these tokens. Then, the SVM is used to classify query and to detect SQLIA. To evaluate SQLiGOT, the authors used a dataset collected from different sources that contains 4610 injected sequences and 4884 genuine sequences. With a ratio of 80% for training (3907 genuine + 3688 injected) and 20% for testing (977 genuine + 922 injected), the scheme achieved 99.47% accuracy and 0.31% false positive rate. Even, the total theoretical processing time introduced by SQLiGoT is within 50 ms over each page load, it presents an overhead for a web server executing more than 100 requests per second.

In [54], the authors used machine learning algorithms to tune the rules of the Web Application Firewall (WAF) Modsecurity [55]. The authors used three classifiers SVM, KNN (with K = 3), and Random Forest classifiers. They showed that they are able to improve the detection capabilities of the Modsecurity WAF and more precisely they have reduced the false negative rate. However, in order to work properly the proposed scheme need a large dataset of malicious attacks related to the given application. They used three datasets CSIC-2010 [56], DRUPAL [57], and PKDD2007 [58]. They used three scenarios to train and test their model. The average precision of SVM, random forest and KNN remains on 97%. Merging the tokenization technique with neural network, the authors in [59] proposed a Token based Detection and Neural Network based Reconstruction (TbD-NNbR) framework to detect and block SQLI attacks. The proposed framework consists of two modules; the detection module and the reconstruction module and is located on a proxy server that lies in between the web server and the database server. The detection module (TbD) is token based meaning that it tries to match the statically generated legal query tokens against the parsed dynamic query tokens at runtime. An SQLI attack is detected if there is no match between the analyzed query and the stored legal queries in the template repository. The main drawback of this techniques is that it needs a large repository that contains all possible valid queries, otherwise the false negative rate will be high. For this purpose, the authors added the neural network reconstruction module NNbR. The idea of the reconstruction module is to reduce the denial of service against authenticated users. Indeed, when the detection module TbD detects an authenticated user query as attack it forwards this query to the NNbR module for reconstruction. The reconstruction consists in generating a valid query from the malicious one by either eliminating the injected portions of the actual query or substituting it with a null value. This process permits to reduce the denial of service and to provide better system availability for authenticated users. The NNbR module uses the Back Propagated-Neural Network (BP-NN) model to train the queries and learn the optimal the legal queries model. Although, the authors claim a high accuracy rate the reported results are done on small datasets (only 1655 queries). Moreover, as already mentioned the scheme might have a high false positive rate and this problem is resolved only for authenticated users.

Table III describes and compares between the previously discussed works using machine learning algorithms. It exposes the different ML classifiers that are used within different context, trained and tested with different datasets. The different operating conditions of the classifiers explain the various performance measures found. The performance of these techniques depends on the choice of the classifier. The best performance reached 100% of precision using Stacked AutoEncoder classifier, while the less achieved accuracy is 66.67% using neural network multi-layer feed forward with a small dataset size. The different performances metrics (Accuracy, Precision, Recognition Rate) obtained prove that the machine learning algorithms are among the most suitable techniques for security problem and detection of the SQLI attack.

IV. CONCLUSION AND DISCUSSION

According to the main security consortiums, OWASP [2] and SANS [60], the injection vulnerabilities continue to be the most spread and dangerous attacks on web applications. A diversity of works have been proposed, and new techniques are emerging to deal with this kind of attacks. However, these huge amounts of work make the selection of the best solution, that fit a given web application, a difficult task. This paper tries to resolve this later problem by giving a general overview, and taxonomy of the main and recent proposed solutions. Indeed, a classification of the SQLI attack countermeasures was presented and discussed. We also highlighted the main characteristics and weaknesses of each solution.

Table IV presents a comparison between the discussed SQL injection solutions. For each solution we enumerated its capabilities in terms of Detection (D), Prevention (P), Generate Report (R), Modelization (M), and Classification (C). Moreover, we indicate the SQLI attack types addressed by the proposed solution.

It is worth noting that some techniques such as [8]–[12] are based on the modelling of the dynamic SQL query and then comparing it with a set of legal models of SQL queries. These techniques have the advantages of simplicity and easy implementation, however, having all the legal queries models is a difficult task. The lack of enough legal queries models can lead to a high false positive rate and therefore might put the web application out of service. Another category of techniques are based on the randomization of the SQL query such as [13]–[16]. The idea of these techniques is to randomize token of the SQL query. This randomization can be achieved by concatenating a random number or by adding a cryptographic salt. This randomization is removed once the query is considered as valid. The main drawback of these techniques is that the randomization technique can change the intended user query and it also limits the possible user input.

TABLE III: Machine Learning Techniques for SQL injection Solutions

Techniques	Classifier(s)	Performance	Dataset
HIPS [41]	Bayesian Naif Bayesian Multinomial	Accuracy= 97.6%	SQL injections data collection framework
SQLI-IDS [43]	Backpropagation Neural Network	Overall accuracy= 96.8%	Dataset 13,000 URL addresses including 500 benign URLs and 12,500 malicious URLs
Sheykhkanloo et al. [44]	Neural Network Based Model	Accuracy= 95%	Dataset 25000 URL addresses including 12250 benign URLs and 12250 malicious URLs
Verbruggen et al. [45]	Decision Tree, SVM, Random Tree, Jribber, Neural Network, Random Forest	Recognition Rate= 98.6% for Neural Network	6 different datasets contains 1876 malicious packets and 11444 normal packets.
Wang et al. [48]	Stacked AutoEncoder	Precision =100%	0.3 million TCP flow data collected from internal network
Ingre e al. [49]	Decision Tree	Accuracy= 83.7%	NSL-KDD [50]
Moosa et al. [51]	Neural network Multi-layer Feed Forward	Accuracy= 66.67%	300 SQL injection signatures and around 200 XSS signatures collected from different websites
Joshi et al. [52]	Naive Bayes	Accuracy= 93.9%	178 Codes including 101 normal codes and 77 malicious codes
SQLiGOT [53]	SVM	Accuracy= 96.23%	4610 injected sequences and 4884 genuine sequences
Modsecurity with ML [54]	K-NN (K=3), SVM, Random Forest	Precision=97%	CSIC-2010 [56], DRUPAL [57], and PKDD2007 [58]
TbD-NNbR [59]	Neural Network	Accuracy= 99.23%	1655 queries tested, 451 malicious queries and 1204 legal queries.

Monitoring and auditing based techniques such as [17]–[22] generally try to audit and analyse code to detect possible vulnerable parts of code. Some techniques make prevention by replacing these vulnerable parts of code. Others, just detect the SQLI attack. The advantage of these techniques is that they can work offline before the deployment of the web application and therefore they can prevent and detect SQLI attack before the deployment of the web application. However, these techniques might present a high false negative rate as some attacks can be launched at runtime, and therefore are undetectable in the offline mode.

Some techniques use information theory and detect the presence of the attacks by measuring the entropy of the query [23]–[27]. These techniques can be bypassed by hackers who could produce malicious queries with acceptable value of entropy.

Ontology based countermeasures such as [28]–[35], [40] try to avoid signatures based techniques weaknesses by giving good models of the SQLI attack types. Modelling the SQLIA is the main tasks of the ontology, some works intend to model attack, while others are enriched by semantic rules to give hand to the ontology model to prevent or detect the SQLIA. Ensuring the semantic relations between the different parts of the attack signature presents the main advantage of using ontology as a detection technique.

In addition, several researchers rely on using machine learning algorithms [41], [43]–[45], [48], [49], [51]–[54], [59]. With ML, the proposed models able to learn without being explicitly programmed. Choosing a well-ordered dataset that is close to reality leads to a successful completeness of the training and testing phases of the classifier. Hence, it can effectively detect the SQLIA.

Although these techniques can detect or prevent the SQL injection attack, there still gaps in their effectiveness in handling the web based threats. As a future work, we are designing an SQLI attack detection scheme based on machine learning that is lightweight and can fit a large number of requests per second. Moreover, we considered that there is a lack of an updated and standard dataset that can be used by researchers to evaluate their work and compare it with the existing solutions. Therefore, we are planning to provide such dataset to be a useful tool for researchers working in this field.

REFERENCES

[1] Statista. (2019) Number of web attacks blocked daily worldwide 2015-2018. [Online]. Available: https://www.statista.com/statistics/494961/web-attacks-blocked-per-day-worldwide/

[2] OWASP, "Owasp top ten project," https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project, 2019, accessed on April 2019.

[3] G. Deepa, P. S. Thilagam, F. A. Khan, A. Praseed, A. R. Pais, and N. Palsetia, "Black-box detection of xquery injection and parameter tampering vulnerabilities in web applications," *International Journal of Information Security*, vol. 17, no. 1, pp. 105–120, 2018.

[4] Y. Fang, J. Peng, L. Liu, and C. Huang, "Wovsqli: Detection of sql injection behaviors using word vector and lstm," in *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*. ACM, 2018, pp. 170–174.

[5] Q. Li, F. Wang, J. Wang, and W. Li, "Lstm-based sql injection detection method for intelligent transportation system," *IEEE Transactions on Vehicular Technology*, 2019.

TABLE IV: SQL injection Solutions Comparison

Techniques	Detection (D) Prevention (P) Generate Report (R) Modelization (M) Classification (C)	Tautologies	Logically Incorrect query	Alternate encoding	Union queries	Piggy Backed query	Stored	Inference
CANDID [8]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SQLStor [9]	P	No	No	No	No	No	Yes	No
Tautology CHECKER [10]	D	Yes	No	No	No	No	No	No
SEPTIC [11], [12]	D/P	No	No	No	No	Yes	Yes	No
Random4 [14]	D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
OBFUSCATION [13]	D	No	No	No	No	No	No	No
SQL Shield [15]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
AutoRand [16]	D/P	Yes	No	No	Yes	Yes	No	No
PSIAQOP [17]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic Analyzer [18]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CCSD [19]	D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DE-PRE [20]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WDIMT [21]	D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JoanAudit [22]	D/R	Yes	Yes	Yes	Yes	Yes	Yes	Yes
$\mu 4SQLi$ [23]	D	Yes	Yes	Yes	No	Yes	Yes	Yes
Information Theory [26]	D	Yes	Yes	Yes	Yes	Yes	No	Yes
MAC based [25]	D	Yes	Yes	Yes	Yes	Yes	No	No
Das et al. [27]	D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Denker et al. [29], [30]	M	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Kim et al. [31]	M	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dobsen et al. [32]	D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Undercoffer et al. [33]	M/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Raskin et al. [34]	M	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Abderazed et al. [35]	M/D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SecWAO [28] [39]	M/D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
NSSA [40]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HIPS [41]	C/D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SQL-IDS [43]	C/D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Verbruggen et al. [45]	C/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SAE [48]	C/D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingre et al. [49]	C/D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Moosa et al. [51]	C/D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Joshi et al. [52]	C/D	Yes	No	Yes	Yes	No	No	No
SQLiGOT [53]	C/D	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Modsecurity with ML [54]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TbD-NNbR [59]	D/P	Yes	Yes	Yes	Yes	Yes	Yes	Yes

[6] W. G. 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. IEEE, 2006, pp. 13–15.

[7] M. S. Aliero, I. Ghani, S. Zainudden, M. M. Khan, and M. Bello, "Review on sql injection protection methods and tools," *Jurnal Teknologi*, vol. 77, no. 13, 2015.

[8] 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, p. 14, 2010.

[9] S. Mamadhan, T. Manesh, and V. Paul, "Sqlstor: Blockage of stored procedure sql injection attack using dynamic query structure validation," in *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*. IEEE, 2012, pp. 240–245.

[10] R. J. Manoj, A. Chandrasekhar, and M. A. Praveena, "An approach to detect and prevent tautology type sql injection in web service based on xschema validation," *International Journal Of Engineering And Computer Science ISSN*, pp. 2319–7242, 2014.

[11] I. Medeiros, M. Beatriz, N. Neves, and M. Correia, "Hacking the dbms to prevent injection attacks," in *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy*. ACM, 2016, pp. 295–306.

[12] ——, "Septic: Detecting injection attacks and vulnerabilities inside the dbms," *IEEE Transactions on Reliability*, 2019.

[13] R. Halder and A. Cortesi, "Obfuscation-based analysis of sql injection attacks," in *2010 IEEE Symposium on Computers and Communications (ISCC),*. IEEE, 2010, pp. 931–938.

[14] S. Avireddy, V. Perumal, N. Gowraj, R. S. Kannan, P. Thinakaran, S. Ganapthi, J. R. Gunasekaran, and S. Prabhu, "Random4: an application specific randomized encryption algorithm to prevent sql injection," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom),*. IEEE, 2012, pp. 1327–1333.

[15] P. Mehta, J. Sharda, and M. L. Das, "Sqlshield: Preventing sql injection attacks by modifying user input data," in *Information Systems Security*. Springer, 2015, pp. 192–206.

[16] J. Perkins, J. Eikenberry, A. Coglio, D. Willenson, S. Sidiroglou-Douskos, and M. Rinard, "Autorand: Automatic keyword randomization to prevent injection attacks," in *International Conference on Detection of*

Intrusions and Malware, and Vulnerability Assessment. Springer, 2016, pp. 37–57.

[17] E. Al-Khashab, F. S. Al-Anzi, and A. A. Salman, "PSIAQOP: preventing sql injection attacks based on query optimization process," in *Proceedings of the Second Kuwait Conference on e-Services and e-Systems*. ACM, 2011, p. 10.

[18] R. M. Nadeem, R. M. Saleem, R. Bashir, and S. Habib, "Detection and prevention of sql injection attack by dynamic analyzer and testing model," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 8, pp. 209–214, 2017.

[19] T.-Y. Wu, C.-M. Chen, X. Sun, S. Liu, and J. C.-W. Lin, "A countermeasure to sql injection attack for cloud environment," *Wireless Personal Communications*, vol. 96, no. 4, pp. 5279–5293, 2017.

[20] P. P. Churi and K. Mistry, "DE-PRE tool for detection and prevention from input validation attacks on website," *Circulation in Computer Science*, vol. 2, no. 5, pp. 23–27, June 2017. [Online]. Available: https://doi.org/10.22632/ccs-2017-252-21

[21] M. Masango, F. Mouton, P. Antony, and B. Mangoale, "Web defacement and intrusion monitoring tool: WDIMT," in *2017 International Conference on Cyberworlds (CW)*. IEEE, 2017, pp. 72–79.

[22] J. Thomé, L. K. Shar, D. Bianculli, and L. Briand, "JoanAudit: a tool for auditing common injection vulnerabilities," in *11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM, 2017.

[23] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for sql injection vulnerabilities: an input mutation approach," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 259–269.

[24] D. G. Kumar and M. Chatterjee, "Detection block model for sql injection attacks," *International Journal of Computer Network and Information Security*, vol. 6, no. 11, p. 56, 2014.

[25] ——, "Mac based solution for sql injection," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 1–7, 2015.

[26] H. Shahriar and M. Zulkernine, "Information-theoretic detection of sql injection attacks," in *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*. IEEE, 2012, pp. 40–47.

[27] D. Das, U. Sharma, and D. Bhattacharyya, "Defeating sql injection attack in authentication security: an experimental study," *International Journal of Information Security*, vol. 18, no. 1, pp. 1–22, 2019.

[28] M. Busch and M. Wirsing, "An ontology for secure web applications." *Int. J. Software and Informatics*, vol. 9, no. 2, pp. 233–258, 2015.

[29] G. Denker, L. Kagal, and T. Finin, "Security in the semantic web using owl," *Information Security Technical Report*, vol. 10, no. 1, pp. 51–58, 2005.

[30] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara, "Security for daml web services: Annotation and matchmaking," in *International Semantic Web Conference*. Springer, 2003, pp. 335–350.

[31] A. Kim, J. Luo, and M. Kang, "Security ontology for annotating resources," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2005, pp. 1483–1499.

[32] G. Dobson and P. Sawyer, "Revisiting ontology-based requirements engineering in the age of the semantic web," in *Proceedings of the International Seminar on Dependable Requirements Engineering of Computerised Systems at NPPs*, 2006, pp. 27–29.

[33] J. Undercoffer, A. Joshi, and J. Pinkston, "Modeling computer attacks: An ontology for intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 113–135.

[34] V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg, "Ontology in information security: a useful theoretical foundation and methodological tool," in *Proceedings of the 2001 workshop on New security paradigms*. ACM, 2001, pp. 53–59.

[35] A. Razzaq, Z. Anwar, H. F. Ahmad, K. Latif, and F. Munir, "Ontology for attack detection: An intelligent approach to web application security," *computers & security*, vol. 45, pp. 124–146, 2014.

[36] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," 1997.

[37] N. Guarino and C. Welty, "Evaluating ontological decisions with ontoclean," *Communications of the ACM*, vol. 45, no. 2, pp. 61–65, 2002.

[38] CVE, "Common vulnerabilities and exposures," https://cve.mitre.org/, 2016, accessed on May 2016.

[39] M. Busch, "Evaluating & engineering: an approach for the development of secure web applications," 2016.

[40] G. Xu, Y. Cao, Y. Ren, X. Li, and Z. Feng, "Network security situation awareness based on semantic ontology and user-defined rules for internet of things," *IEEE Access*, vol. 5, pp. 21 046–21 056, 2017.

[41] A. Makiou, Y. Begriche, and A. Serhrouchni, "Improving web application firewalls to detect advanced sql injection attacks," in *Information Assurance and Security (IAS), 2014 10th International Conference on*. IEEE, 2014, pp. 35–40.

[42] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[43] N. M. 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*. ACM, 2015, pp. 258–266.

[44] ——, "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.

[45] R. Verbruggen and T. Heskes, "Creating firewall rules

with machine learning techniques," Ph.D. dissertation, Kerckhoffs institute Nijmegen, 2015.
[46] "Dataset kdd cup," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[47] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.
[48] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, 2015.
[49] B. Ingre, A. Yadav, and A. K. Soni, "Decision tree based intrusion detection system for nsl-kdd dataset," in *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, 2017, pp. 207–218.
[50] "Dataset nsl-kdd," https://www.unb.ca/cic/datasets/nsl.html.
[51] A. Moosa, "Artificial neural network based web application firewall for sql injection," *World Academy of Science, Engineering and Technology*, vol. 4, 2010.
[52] A. Joshi and V. Geetha, "Sql injection detection using machine learning," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, 2014, pp. 1111–1115.
[53] D. Kar, S. Panigrahi, and S. Sundararajan, "Sqligot: Detecting sql injection attacks using graph of tokens and svm," *Computers & Security*, vol. 60, pp. 206–225, 2016.
[54] . P. G. Betarte and R. Martnez, "Web application attacks detection using machine learning techniques," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 1065–1072.
[55] I. T. Holdings, "Modsecurity: Open source web application firewall," http://www.modsecurity.org/, Accessed on April 2019.
[56] "Dataset csic-2010," http://www.isi.csic.es/dataset/.
[57] "Dataset drupal," https://www.drupal.org/project/dataset.
[58] "Dataset pkdd2007," http://www.lirmm.fr/pkdd2007-challenge/.
[59] T. K. George, K. P. Jacob, and R. K. James, "Token based detection and neural network based reconstruction framework against code injection vulnerabilities," *Journal of Information Security and Applications*, vol. 41, pp. 75–91, 2018.
[60] CWE/SANS, "CWE/SANS top 25 most dangerous software errors," http://www.sans.org/top25-software-errors/, accessed on April 2019.