cDate of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.Doi Number

MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique

Fawaz Mahiuob Mohammed Mokbal[1], Wang Dan[1], Azhar Imran[2], Lin Jiuchuan[3] , Faheem Akhtar[2,4],and Wang Xiaoxi[5]

[1]College of Computer Science and Technology, Beijing University of Technology, China (e-mail: fawaz@email.bjut.edu.cn, wangdan@bjut.edu.cn)
[2]College of Software Engineering, Beijing University of Technology, China (e-mail: azharimran63@gmail.com, fahim.akhtar@iba-suk.edu.pk)
[3]Key Lab of Information Network Security of Ministry of Public Security (The Third Research Institute of Ministry of Public Security), China（e-mail:linjiuchuan@ stars.org.cn）
[4]Department of Computer science, Sukkur IBA University, Sukkur 65200, Pakistan. (e-mail: fahim.akhtar@iba-suk.edu.pk)
[5]State Grid Management College, Beijing 102200, China (e-mail: wxxwxx0904@163.com)

This work was supported by the Opening Project of Key Lab of Information Network Security of Ministry of Public Security（The Third Research Institute of Ministry of Public Security(C 17613)

ABSTRACT Dynamic web applications play a vital role in providing resources manipulation and interaction among clients and servers. The features presently supported by browsers have raised business opportunities, by supplying high interactivity in web-based services, like web banking, e-commerce, social networking, forums, and at the same time, these features have brought serious risks and increased vulnerabilities in web applications that enable Cyber-attacks to be executed. One of the common high-risk cyber-attack of web application vulnerabilities is Cross-Site Scripting (XSS). Nowadays, XSS is still dramatically increasing and considered as one of the most severe threats for organizations, users, and developers. If the ploy is successful, the victim is at the mercy of the cybercriminals. In this research, a robust artificial neural network-based multilayer perceptron (MLP) scheme integrated with the dynamic feature extractor is proposed for XSS attack detection. The detection scheme adopts a large real-world dataset, the dynamic features extraction mechanism, and MLP model, which successfully surpassed several tests on an employed unique dataset under careful experimentation, and achieved promising and state-of-the-art results with accuracy, detection probabilities, false positive rate, and AUC-ROC scores of 99.32%, 98.35 %, 0.3%, and 90.02% respectively. Therefore, it has the potentials to be applied for XSS based attack detection in either the client-side or the server-side.

INDEX TERMS Artificial Neural Network, Cross-Site Scripting attack, Detection; Multilayer Perceptrons, Web Application Security.

I. INTRODUCTION

Web technology has beenincreased exponentially in daily volume and interactions involving web-based services, such as self-driving finance in web banking, Chatbots /AI assistants and recommendation engines in e-commerce, social networking sites such as Facebook, Twitter, forums, blogs, and much more. This technology has become an integral part of our daily and private lives, and at the same time, web applications became primary targets of cybercriminals. Cybercriminals exploit the poor code experiences of web developers, weaknesses within the code, improper user input sanitization, and non-compliance with security standards by the software package developers [1]. Additionally, the vulnerabilities may be found in reusable software components (i.e., third-party libraries, open-source software, etc.) which are heavily used to develop web applications, also vulnerable cyber-defense system where attackers regularly develop their offensive tactics by devising new ways to bypass the defense systems, and exploiting the vast sophistication of AI technology to facilitate their pernicious tasks [2]. It mandates the development of more sophisticated web application cyber-defense systems, which can be accomplished using the latest AI concepts with high precision to tackle the new web-based attacks.

One of the common high-risk cyber-attack to web application vulnerabilities is Cross-Site Scripting (XSS), which has placed web applications, users, and even the industrial field at high risk [3]. According to the National Vulnerabilities Database (NVD) database [4], the number of reported issues are dramatically increasing, especially in 2018. XSS-based vulnerabilities are considered to be the second most common vulnerability occurring [5, 6], and still on the top of 10 attacks in 2017 in OWASP [7].

XSS is a security vulnerability that can affect web application. If present in any web application, this vulnerability can allow cybercriminals to add their malicious code into the HTML pages displayed to receiving end-users. Primarily, there are three types of XSS based attacks that include stored-XSS, Reflected-XSS, and DOM-based XSS attacks [8, 9]. The Stored/Persistent-XSS occurs when malicious code takes place in the web application database. The attacker could inject malignant code into vulnerability server by (e.g., message forums or blog posts), the harmful code is stored in the application server, and, once the victim visits or navigator the infected application, they are served with the malignant code as a part of the legitimate web page. In the end, the victim's browser finishes executing the malicious code.

The Non-Persistent/Reflected-XSS occurs during the website echoes, which backs a portion of the request to the browser, the attacker has to trick the victim into clicking an evil link (i.e., through malignant JavaScript (JS) on another page or a phishing an email), that triggers the XSS-attack. Once the victim clicks the link, an HTTP-Req to the web server is sent together with the malignant code as a component of it. The response from the web server includes the malignant code part (i.e., reflected). Therefore, the malignant code will execute indifferently by a victim's browser. In DOM Based XSS, the downright malicious code flows from source to sink takes place in the browser (i.e., DOM objects) XSS-based attacks may damage and change the behavior or appearance of the enterprise website, stealing sensitive companies' information or private users' data, and performing actions on behalf of the user [10].

The defense mechanism against these attacks could be on either the Server-side, Client-side, or both. In terms of analyzing, there are three approaches to defense against XSS-based [11] are:
1) Static analysis approach, which reviews web application code inlcudes source code, binary code, or byte-code to find out how the control/ data would flow at runtime before the program executes.
2) Dynamic analysis analyzes the data obtained throughout program execution to discern vulnerabilities. Dynamic analysis is usually performed at a testing time during development or runtime once the software package is released.
3) The hybrid approach combines both the static and dynamic analysis approaches.

Although some solutions have already been proposed, such as mitigation tools, detection/prevention methods for XSS attacks, moreover, existing techniques still can't sufficiently detect XSS malicious code [12-14]. Current research works attempt to add intelligence to increase detection accuracy by using AI concepts, which is evident from the extensive use of machine learning techniques in the detection of cyber-attacks [15]. However, the classical machine learning techniques are represented with shallow detection rate and are incapable of detecting tiny mutants of existing malicious attacks such as zero-day attacks and etc [16]. Although most of those attacks are the small variants of the already well-known malicious code (close to 99% mutations), even the so-called a unique attacks (1%) rely on the previous concepts and logic. In other words, malignant behaviors that deviate sufficiently in feature from those seen before would fail to be classified; hence, it will override the undetected system. The success of deep neural network within the big data field considerable,and it can also be used to combat cyber threats because mutations of attacks are like small changes in image pixels. It implies that deep neural network learning in security learns a real face (benign or malignant) of learning data on even with small changes or mutations by exploiting the resiliency and capability of the deep neural network to small changes in data by producing high-level invariant representations of the training dataset.

In this research, an ANN-based detection scheme is introduced for the XSS-based web-applications attack.
A large dataset has been constructed and used for the training and testing of the detection scheme, along with proposing a novel technique for features extraction and a Multilayer Perceptron for the detection task. The detection scheme successfully surpassed several tests on a newly employed testing dataset and achieved promising and state-of-the-art results with detection probabilities, false positive rate, and AUC-ROC scores of 98.35 %, 0.3%, and 90.02% respectively.
The main contributions of this paper can be summarized as follows:

- An extensive real-world data composed of 138,569 unique records to detect XSS attack have been constructed comprehensively and uniquely.
- A dynamic-features extraction technique has been proposed, which acts as a layer for extracting and providing training and testing dataset to feed the neural network model in dynamic behavior.
- Introduced a robust, high precision and low complexity deep neural network scheme, making it easier to deploy along with the dynamical feature extraction model, which is platform independent of detecting XSS-based attack.
- The proposed scheme is tested on the new test set composed of 27,714 samples to demonstrate the capability of our model even XSS-based zero-day attack detection.

The rest of this paper is organized as follows,
Section II discusses related work with introducing research gaps that our proposed scheme is capable of entertaining. Section III presents the key details about literature and mechanism of this research that includes, raw data construction, feature extraction technique, and Multilayer Perceptron model for detection. Section IV presents the experimental design and evaluation mechanism of this research. Section V presents the results and discussion.

Finally, section VI concludes this research by highlighting some future research directions with the significance of this research.

II. RELATED WORK

Numerous researches have been proposed concerning cyber-attacks; still, cybercrimes and fraudulent activities of cybercriminals are increasing; authors in [17, 18] investigated various types of cyber frauds and introduced useable and feasible solutions.

The Sanitization of JavaScript attack payloads is deemed to be as an essential approach to obstruct the exploitation of XSS-based attack on web-based applications. In [19], the author proposed a cloud-based framework that restrains the DOM-based XSS vulnerabilities. The idea lies in improving and optimizing the context-sensitive sanitization process of HTML5 attack vector with injecting decisive code into the nested context of suspected variables. In [20], the authors introduced a JavaScript SANitizer framework that uses the capabilities of clustering with sanitization mechanisms to diminish the effect of JS vulnerabilities with the determination of their level of similarity in context. Furthermore, in [21], the author proposed to use a PHP-Sensor to identify the XSS vulnerabilities within web-based applications. The primary intention is to observe that the HTTP request and response might carry self-propagation XSS payloads. Thus, PHP-Sensor observes the scripts present in an outgoing HTTP Request and checks whether those scripts exists within the response. However, defense mechanisms to prevent such attacks using traditional methods of detection are extremely complicated. There is neither a single solution which may effectively mitigate XSS attack [12], nor have the capability to reduce existed flaws in the source code of applications [13]. Furthermore, existing techniques may also fail to detect XSS malicious for several reasons; and even most of them are not able to detect sophisticated-XSS attacks [14].

Furthermore, based on the widespread use of Artificial Intelligence (AI) technique in numerous application domains with higher detection ability, several researchers also engaged themselves for the detection of XSS attacks using machine learning (ML) technique. Likarish *et al.* [22] proposed 65 features to detect the obfuscated malicious JavaScript and performed classifications using Naive Bays, alternating decision tree (ADTree), support-vector machines(SVM), and RIPPER (Repeated Incremental Pruning to Produce Error Reduction) to evaluate features' effectiveness. However, the best precision score of 0.920 is achieved with SVM classifier. Therefore, the low precision and reduced detection rate which leads to a high false positive rate does not meet the purposes of such a model in the real world. The researchers in [23] proposed a machine learning approach to detect XSS in social networks. They extracted the web pages' features manually; classified them in four groups and each group is composed of multiple features where three features are related to the online social network; they used Adaptive Boosting (AdaBoost) and ADTree algorithms with 10-fold cross-validation; the highest precision and recall are obtained with AdaBoost algorithm, the scores are 0.941 and 0.939 respectively. Morover, the results represented a shallow detection rate, and most importantly, the high false positive rate was up to 4.20%. Similarly, in [24], the detection of XSS attack based on a machine learning approach is proposed. All the 25 features are divided into three groups, including URLs, webpage, and SNSs. They claimed that with Random Forest algorithm, the accuracy and false positive rate increased up to 0.9720 and 0.087, respectively. However, the False Positive rate is still high which is equal to 8.7%.

Furthermore, deep learning schemes are also used to overcome the limitations of the classical machine learning approaches as they can produce high-level representations of features and can even predict hidden features using training data. Wang et al. [25] applied the stacked denoising autoencoder model for malicious code detection on a web-based application and used sparse random projections for dimensions reduction. The results were excellent in term of accuracy (about 95%). However, the detection rate is between 93% and 95%, and the false positive rate was 4.2%, which is considered very high.

The authors in [26] proposed an unsupervised end-to-end deep learning scheme. They used the Robust Software Modeling Tool (RSMT) as a monitor to capture features and used autoencoders to learn low dimensional representations. Performance evaluations were performed by applying supervised and unsupervised techniques. However, RSMT has caused overhead issues; also, experiments performed using supervised ML technique was poor (around 0.728 at best case for XSS using SVM). Moreover, the unsupervised deep learning performance was not efficient (around 0.906).

The study in [27] proposed a deep recurrent neural classification system called ScriptNet to detect either malicious JavaScript or VBScript by utilizing a combination of both static and dynamic analysis. Despite the complexity of the model, the results were not convincing (best-performing LaMP model has a 65.9% true positive rate and best CPoLS model has a 45.3 % true positive rate, all with 1.0% False positive rate). Thus, applying only word vectors are not suitable for JavaScript because it uses encapsulation, code rearrangement, rubbish strings insertion, and alternative techniques.

Another research in [28] proposed a DeepXSS detection, which contains decoding, generalization, and tokenization techniques. They used word2vec for XSS feature extraction, which was given as an input to the deep learning model, based on LSTM for training. They achieved about 0.995 precision, 0.979 Recall, and a false positive rate tends to 0.019. However, the webpage includes JavaScript and HTML code, which is primarily not a standard encoding; therefore, adopting word vectors make the training process incredibly time-consuming, tedious, and not suitable.

III. DETECTION METHODOLOGY

The XSS-based attack occurs due to security bugs in the websites, because of the features originating from the dynamic web applications and supported by web browsers such as HTML tags, hyperlinks, scripts, and advanced functions. These features are attractive and useful, but they also bring severe risks and increased security vulnerabilities for web applications. Cybercriminals usually exploit these vulnerabilities. Based on this problem, this research proposes a new detection scheme, which consists of three main pillars including quality of data, appropriate feature vectors that wholly and genuinely characterize the XSS anomalous phenomenon and adopt an ANN technique for detection. For this purpose, three modules have been developed and work together in one working environment (Python) to form a final framework. These three main pillars are shown in the schematic of the proposed scheme given in Fig. 1, and the details are as follows:

A. COLLECTING RAW DATA

Currently, there is no open dataset available for XSS based attacks [29]. Building a new digital dataset of such attacks containing malicious and benign instances for training and testing a model is quite challenging. Therefore, to create a new dataset, one approach is to crawl all the web applications with different webpages. The size of the web applications precludes this strategy from being productive. Alternatively, one can crawl only a part of the web applications. However, crawling the web in some fixed, and settled manner is problematic because it potentially biases the dataset, which is not comprehensive nor miscellaneous, and the model could not generalize it. Besides, it is not an obvious way to argue that a sufficient subset of the websites is representative. For this, a novel approach is used to create a large real-world dataset and to make it comprehensive and miscellaneous to tackle the issues concerned with it. The approach is described as follows.

For benign samples, our crawler employed a robust python scraping framework [30] and implemented a random walk along with a random jumping method to crawl a sufficient variety of portions of the websites. A mathematical equation is used to select the probability of sample X, where X is crawled by the Formula (1) presented by [31].

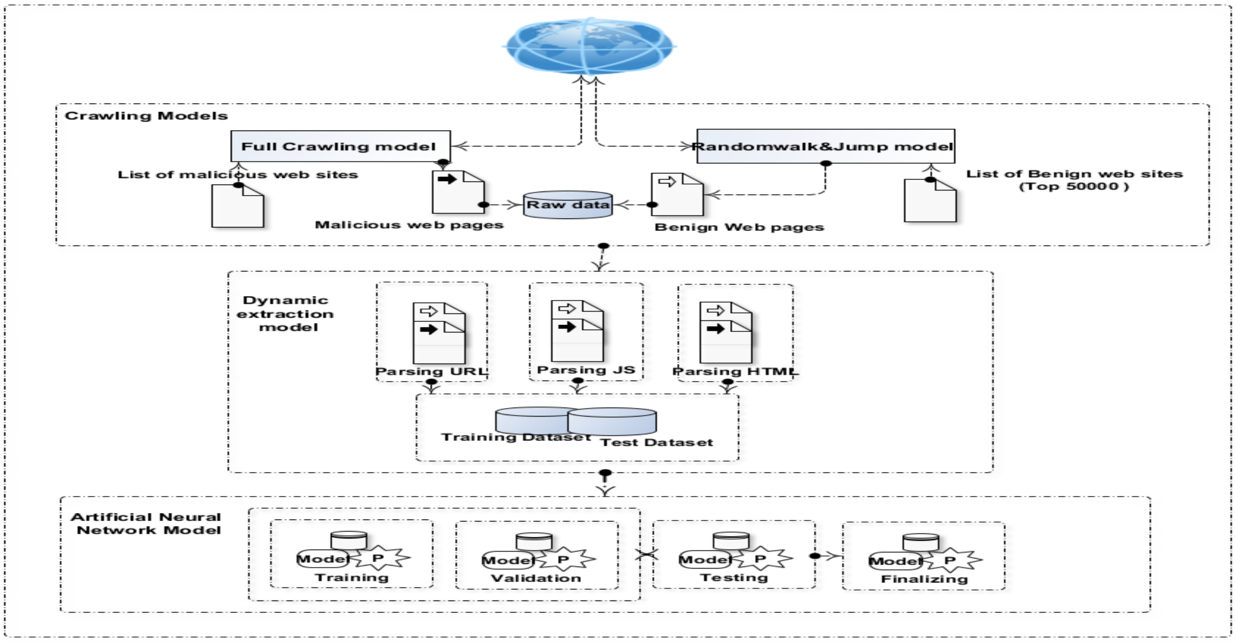

FIGURE 1. THE FRAMEWORK OF THE PROPOSED SCHEME

$$\Pr(X) = \Pr(\hat{X}) * \Pr(X|\hat{X}) \tag{1}$$

Where X is sampled, and $\hat{X}$ is crawled. To estimate $\Pr(X|\hat{X})$, they used the probabilities of each observation in inversely proportional to its PageRank. Based on their method, the samples selecting probabilities are developed with a random walk and jumping method. All crawler rules were written inside the spider class, and the top 50,000 of websites ranking in Alexa feed the initial seed of crawler [32]. Our crawler was configured to obtain maximum samples of 150,000 pages, the number of independent parallel walks is 100, and the jumping probability range between [0,1]. Because of the random walk and jumping, it is possible to have a small number of duplicate samples. These duplicate samples are deleted to form a resultant sample of 148,157 records. Moreover, uniform random sample pages are selected from the crawled portion to obtain 100,000 samples from resultant samples, which makes it uniform, comprehensive and miscellaneous, to be an optimal case for training and testing model.

For malicious samples, due to the limited availability of samples, another crawler is designed to crawl the raw data from XSSed [33] and Open Bug Bounty [34] in some fixed and deterministic manner. However, the duplicate samples and the pages of more than 25MB in size are excluded. The total of 38,569 malicious samples is considered for this research.

B. DYNAMIC FEATURES EXTRACTION MODEL

In several cases of Natural Language Processing using deep neural network technique, the text samples are represented in the form of word vectors, but in case of an XSS attack, malicious code is often a JavaScript code. Thus, JavaScript has non-standard encoding; there exist many senseless strings and Unicode symbols. Additionally, JavaScript uses data encapsulation, code rearrangement, rubbish strings insertion, and alternative techniques where the word vectors lead to generating large dimensions is the very time-consuming case of XSS.

This research introduces a novel dynamic feature extraction model which is introduced to get proper feature vectors that characterize the XSS anomaly. The model is integrated with the BeautifulSoup library [35] in combination with the html5lib parser [36] to pull data out of HTML raw files, and Esprima JavaScript parser [37] to tokenize and extract the abstract syntax tree from JavaScript code. The goal of this model is to dynamically find and extract the essential characteristics of both benign and malicious code from raw data provided by the crawling model, and subsequently, these features set are offered to the neural network as a digital dataset in a dynamic way. To accomplish it the model was developed uniquely, containing three sub-models, each of which has a set of features to deal with. These sub-models are HTML-based features sub-model, JavaScript-based features sub-model, and URL-based features sub-model, which is described as follows.

1) HTML-BASED FEATURES SUB-MODEL

The HTML tags are used for constructing the attack vector (i.e., *div, script, iframe, meta, embed, link, SVG, frame, form, style, video, img, textarea, etc*.,) and the attacker has many ways to embed these tags into JavaScript using the corresponding attributes to insert the malicious code [11, 38]. Additionally, most of them support the pseudo-protocol form such as JavaScript: [code] and it could be used in the same manner as a URL when calling an external resource (i.e., redirection) [39]. The protocol represents the body of the attributes which could be any JavaScript code either malicious that calls JavaScript interpreter. The attributes exploited to insert the malicious code (i.e., *href, HTTP-equiv, lowsrc, src, formation, background, target, action, classid, etc*.,) can trigger JavaScript events. The event acts as an interaction between JS and HTML, which can trigger based on the user's actions (*onclick, onmouseover*) or even through the web browser itself (*onload, onerror*). Since the event itself can execute JavaScript code, the attacker may use these features to carry XSS attacks. Moreover, introducing JavaScript in an event method *(<IMG SRC=# onmouseover="alert('XSS')">)* will likewise apply to any HTML tag type injection that utilizes HTML elements. It will thus allow any relevant event for the tag to be substituted (e.g., *onclick, onblur*) and results in an extensive amount of variations for many injection vectors and can bypass most domain filters [40].

Three core features related to HTML are selected in this study, including tags, attributes, and events. The proposed model is designed to extract features and JavaScript code from HTML raw files in combination with the html5lib parser. The html5lib parser gives us the optimal representation way and is extremely broad as it parses the pages in the same manner, as a web browser does. Furthermore, it also performs the HTML5 parsing algorithm, which is heavily impacted by modern browsers and based on the WHATWG HTML5 specification [41]. Once the tree structure of HTML data for the webpage is created, the process of navigating and searching parse tree is required, i.e., tree traversal. For this task, a BeautifulSoup python library is used to create an object for each page which can be executed in parallel with html5lib. Since the object of this library contains all the data in a tree structure, the tags, attributes, and event of each webpage are programmatically extracted as shown in Algorithm 1.

Algorithm 1: Parsing HTML Documents
Input: set of web pages WP $\{wp_1, wp_2 \dots wp_n\}$
Output: Html-based Feature Vector (H_{FV})

```
TG[ ] ← list representing HTML tags
AT[ ] ← list representing HTML attributes
EV[ ] ← list representing HTML events
KE[ ] ← list representing keywords_evil
P ← Null
$H_{FV}$ {} ← Ø
for each $wp_i \in WP$ do
// parsing each web page using html5lib and BeautifulSoup for tree traversal
    P←parse_html ($wp_i$)
// Collect and count each TG, AT, EV, KE from each node in tree
      for $each\ node_i \in P$ do
          for each $t_i \in TG$ do
              $H_{FV}[t_i] = \sum t_i$ , $t_i$ in node $_i$
          for each $a_i \in AT$ do
              $H_{FV}[a_i] = \sum a_i$ , $a_i$ in node $_i$
          for each $e_i \in EV$ do
              $H_{FV}[e_i] = \sum e_i$ , $e_i$ in node $_i$
          for each $k_i \in KE$ do
              $H_{FV}[k_i] = \sum k_i$ , $k_i$ in node $_i$
      end for
    $H_{FV}[hl] = len(p)$  // html length
end for
```

Returns features vector (H_{FV}) for future model uses

2) JAVASCRIPT-BASED FEATURES SUB-MODEL

JavaScript codes live within a webpage and interact with the DOM. It can either provide additional features of the webpage or create an application within the webpage. The source code written in any programing language can be represented as an abstract syntactic structure called Abstract Syntax Tree (AST).

Each node of the tree represents a construct in the sourcecode. The resulting syntax tree is beneficial for different purposes, from program transformation to static program analysis [42].

The JavaScript code could be either benign or malignant based on its semantics rather than the syntax. JavaScript-based features are extracted through the following three main procedures in this research.

1) The set of features related to an XSS attack are identified as *min_length, max_length* (which are higher in malignant), *min_define_function, min_function_calls* (which is more advantageous for benign), *methods* (which can cause XSS attacks like *getElementsByTagName, write, alert, eval, prompt, confirm, fromCharCodefetch*), *Dom-Objects (windows, location, and document)*, and *location _properties (cookies, document, referrer)*.

2) Since some of the JavaScript codes are involved in HTML elements, they can be invoked in various ways from HTML, for instance from tag (e.g., <*script*>), from attribute of some tags (e.g., *href= "JavaScript:" of 'a' tag*) and/ or event (e.g., *onSubmit=" JavaScript:" onload=" JavaScript"*) etc., as described in W3C Consortium lists [12]. The JavaScript codes are extracted from all possible places that can be retrieved from them.

3) The dynamically extracted code is distributed into a series of tokens and syntax tree using Esprima parser is produced, which acts in the same manner as JavaScript engine does. Hence, lexical and syntactical analysis of JavaScript code can be performed at the same time.

However, it has been observed in some cases that the JavaScript used by XSS could be broken and cause parser stop. To deal with this, force a parser to continue parsing, and produce a syntax tree even the JavaScript does not represent a valid code. The parser is configured to a tolerant mode and set the tokens flag to be true in the configuration object to keep the tokens that found during the parsing process. To traversing the entire tree, the generator function was made to take Esprima node and yielding all child nodes at any level of the tree, which gives us the full ability to visit all branches of the tree.

The complete processing of above steps are presented in the follwoing Algorithm 2 which is used to get a JavaScript-based Feature vector (JS_{FV}) for the future model.

Algorithm 2: Parsing JavaScript codes
Input: set of webpages WP {$wp_1, wp_2 … wp_n$}
Output: JavaScript-based Feature vector for each webpage (JS_{FV})

```
DO[ ] ← list representing domObjects
JP[ ] ← list representing JS properties
JM[ ] ← list representing JS methods
TG[ ] ← list representing HTML tags
EV[ ] ← list representing HTML events
AT[ ] ← list representing HTML attributes
P ← Null
js ← Null
JS_strings = []
$JS_{FV}$ {} ← Ø
for each $wp_i \in WP$ do
    P←parse_html($wp_i$)
    with P do
     for $t_i \in TG$ do
      // Extract JS code from <script>
        if ($t_i$ == script and AT[src] == false) then
          js = $t_i$.string
          if (js ≠ Ø) then
             JS_strings.append(js)
        //Extract JS code from javascript: links
        if ($t_i$ == a and AT[href] == true) then
          js = $t_i$.string
          if (js ≠ Ø) then
             JS_strings.append(js)
        //Extract JS code from form
        if ($t_i$ == form and AT[action] == true) then
          js = $t_i$.string
          If (js ≠ Ø) Then
             JS_strings.append(js)
        //Extract JS code from iframe
        if ($t_i$ == iframe and AT[src] == true) then
          js = $t_i$.string
          if (js ≠ Ø) then
             JS_strings.append(js)
        //Extract JS code from frame
        if ($t_i$ == frame and AT[src] = true) then
          js = $t_i$.string
          if (js = Ø) then
             JS_strings.append(js)
     end for
  // Generate tokens, Extract Abstract Syntax Tree of JavaScript code
    E-Object = esprima.parseScript(JS_strings)
    // Generated tree level (node)
    for each $TL_i \in$ E-Object[body] do
      // subnode in the current level
      for each $ND_i \in TL_i$ do
         // Function Declaration e.g. function fun(){…}
         if ($ND_i$[type] in {FunctionDeclaration}) then
            $JS_{FV}$[js_define_function]+=1
         // function calls e.g. var fun = function(){…}
         else if ($ND_i$[type] in
                 {CallExpression, FunctionExpression}) then
            $JS_{FV}$[js_function_calls] += 1
      end for
    end for
    tokens = E-Object[tokens]
    for $TK_i$ in tokens do
      if ($TK_i$[type] == Identifier) then
        If ($TK_i$[value] ∈ DO) then
          $JS_{FV}$[DO_Tki[value]]+=1
        else if ($TK_i$[value] ∈ JP) then
          $JS_{FV}$[JP_Tki[value]]+=1
        else if ($TK_i$[value] ∈ JM) then
          $JS_{FV}$[JM_Tki[value]]+=1
      else if ($TK_i$[value] == string) then
          Stringlist.append(TKi[value])
    end for
    if Stringlist >0 then
      $JS_{FV}$[min_length] = min(len(Stringlist))
      $JS_{FV}$[max_length] = max(len(Stringlist))
end for
Returns features vector ($JS_{FV}$) for future model uses
```

3) URL-BASED FEATURES SUB-MODEL

URL parameters are the critical suspicious of non-persistent XSS injection point [43]. Cyber-criminals are always trying to obfuscate or encode URLs in different ways to bypass filtering tools or mask malicious code to trick or redirect users. A full URL parsing algorithm to obtain the various features that can be exploited by attackers in this research is shown in Algorithm 3. The features considered are as follows:

(1) URL redirection that was used to trick users and redirect the current page to another web page controlled by an attacker

(i.e., *document.URL, document.URLUnencoded, document.baseURI, document.documentURI,location, window.location, window.history, window.navigate, window.open, self.location, top.location*); (2) IPs; (3) domain numbers; (4) Keyword parameters that are frequent on URL as the variable names for allocating user input values (i.e., *search, login, signup, query, contact, URL, redirect*). (5) The tags whose presence in the URL indicates a high probability of an existence XSS (*script, iframe, meta, SVG, img, style, etc*.,) and (6) properties of HTML (i.e., *href, src, formaction*, etc.,) which can host JavaScript code used in URL. (7) The URL length that has been used effectively with phishing detection (8) special characters (i.e., "< ", ">" and "/") which mostly used to perform XSS attacks [26]; (9) in keyword evil most of the intruders reveals their signature.

Each URL of each webpage crawled is decoded to extract URL features and to make it as a string. Then, Regular Expressions is performed to match text patterns and extract these features, which is shown in Algorithm 3.

```
Algorithm3: Parsing URL Addresses
Input: Pages URL
Output: URL-based Feature vector (U_FV)
URD[ ] ← list representing URL redirection
UFP[ ] ← list representing URL frequent parameters
KE[ ] ← list representing keywords evil
TG[ ] ← list representing HTML tags
EV[ ] ← list representing HTML events
AT[ ] ← list representing HTML attributes
$U_{FV}$ {} ← Ø
$U_{Str}$ []
for each page[URL_i] do
    Collect encode-length
    $U_{Str}$ = urldecode(URL_i)
    $U_{FV}$ [url_length] = Len($U_{Str}$)
    for $t_i \in TG$ do
        if ($t_i$ exist in $U_{Str}$) Then
            $U_{FV}$ [url_tag[t_i]] = True
        else
            $U_{FV}$ [url_tag[t_i]] = False
    end for
    // check tag, event, and attributes existing in URL
    for each $a_i \in AT$ do
        if ($a_i$ exist in $U_{Str}$) then
            $U_{FV}$ [url_ parameter[a_i]] = True
        else
            $U_{FV}$ [url_ parameter[a_i]] = False
    end for
    for each $e_i \in EV$ do
        if ($e_i$ exist in $U_{Str}$) then
            $U_{FV}$ [url_ event[e_i]] = True
        else
            $U_{FV}$ [url_ event[e_i]] = False
    end for
    // find any URL redirection parameters
    if (any($urd_i \in URD$ exist in $U_{Str}$)) then
        $U_{FV}$ [URD] = True
    else
        $U_{FV}$ [URD] = False
    if (document.Cookie exist in $U_{Str}$) then
        $U_{FV}$ [url_cookie] = True
    else
        $U_{FV}$ [url_cookie] = False
    for each $ufp_i \in UFP$ do
        // Collect URL frequent parameters existing in UStr
        $U_{FV}[ufp_i] = \sum ufp_i$ , $ufp_i$ in $U_{Str}$
    for each $k_i \in KE$ do
        // Collect URL keywords evil existing in UStr
        $U_{FV}[k_i] = \sum k_i$ , $k_i$ in $U_{Str}$
    for each $domain_i$ found in $U_{Str}$ do
        // Collect the number of domain
        $U_{FV}[number\_domain] += 1$
    for each $IP_i$ found in $U_{Str}$ do
        // Collect a number of IPs
        $U_{FV}[number\_IP] += 1$
end for
Returns features vector (U_FV) for future model uses
```

C. ARTIFICIAL NEURAL NETWORK MODEL

Artificial Neural Network (ANN) is inspired by a human brain operation [44]. It is composed of many layers including, the input layer, the hidden layer(s) and the output layer. In this research, an ANN-based multilayer perceptron (MLP) algorithm which characterized by the dynamic features of XSS-based attack detection is used.

The model has m-layer (4 - 2 hidden layers) that figures the dataset input-output pairs $(\overrightarrow{x_i}, y_i)$, where y_i denote to one-dimensional output $y_i \in \{0,1\}$, where 0= benign and 1= malicious, on n-dimensional input $\vec{x} = \{x_1, x_2, \dots x_n\}$ as shown in Fig. 2. The dataset input-output pairs size N referred $X = \{(\overrightarrow{x_1}, y_1), \dots, (\overrightarrow{x_N}, y_N)\}$. The neurons at hidden layers have f_h activation functions (i.e., ReLU=max (0, x)) and the neuron at the output layer has f_z activation function (i.e., sigmoid= $1/(1 + e^\wedge(-x))$). The layers are fully connected in which every neuron depends on the outputs of all the neurons in the previous layer [45]. MLP model training is done through a Back-propagation algorithm [46].

Let w_{ij}^k denotes to the weight of the link between i^{th} neuron of l_k layer and j^{th} neuron (if k ≥1) of l_{k-1} layer, $\vec{x}$ denotes to the input vector to the model, and z_i^k be the output of neuron i in layer l_k. We introduce an extra weight parameter for each neuron, b_i^k representing the bias for i neuron in the layer l_k, and r_k represent the number of neurons in layer l_k. The s_i^k representing the product of summation add with the bias for a neuron i in layer l_k. Thus, $\overrightarrow{w_i^k}$ representing the weight vector for neurons i in the layer l_k, so be it as $\overrightarrow{w_i^k} = \{w_{1i}^k, \dots, w_{rki}^k\}$. Also, $\overrightarrow{z^k}$ be the output vector for the layer l_k, so be it as $z^k = z_1^k, \dots, z_{rk}^k$. The output z from the MLP model is obtained through a feed-forward phase. It is started by initializing the input layer l_0, set the values of the outputs z_i^0 for neurons in the input layer l_0 to their related inputs in the vector $\vec{x} = \{x_1, \dots, x_n\}$. For each hidden layer in order from l_1 to l_{m-1}, calculate the weight sums and outputs as Formula (2) and Formula (3).

$$s_i^k = \overrightarrow{w_i^k}.\overrightarrow{z^{k-1}} + b_i^k$$
$$s_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k . z_j^{k-1} \; for \; i = 1, \dots, r_k \qquad (2)$$

$$z_i^k = f_h(s_i^k) = \begin{cases} s_i^k, & s_i^k > 0 \\ 0, & else \end{cases} \quad for \; i = 1, \dots, r_k \qquad (3)$$

Where f_h is the activation function (ReLU) at hidden layers. The computing output $\hat{y}$ for output layer l_m done as Formula (4) and Formula (5).

$$s_1^m = \overrightarrow{w_1^m}.\overrightarrow{z^{m-1}} + b_1^m = b_1^m + \sum_{j=1}^{r_{m-1}} w_{j1}^k . z_j^{k-1} \qquad (4)$$

$$\hat{y} = z_1^m = f_z(s_1^m) = \frac{1}{1+e^{-s_1^m}} \tag{5}$$

Where f_z activation function for the output layer is a logistic function (sigmoid), the learning depends on iteratively adjusting the values of $\overrightarrow{w_i^k}$ and b_i^k so as to minimize the L loss function. However, the loss function in our case is Cross-Entropy called a logarithmic function given as Formula (6).

$$J = L(\hat{y}_i, y_i) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i) \tag{6}$$

Where $\hat{y}_i$ indicates the computed output of the MLP model on input $\overrightarrow{x_\imath}$, whereas y_i indicates to the actual value for input-output pair $\overrightarrow{x_\imath}, y_i$. The goal is minimizing J concerning respect to all w_{ij}^k and b_i^k through gradient descent process. Accordingly, Adam optimizer is used to adjust the parameters w_{ij}^k and b_i^k with α as a learning rate, will produce delta equations for each iteration $\Delta w_{ij}^k = -\alpha \frac{\partial L(J)}{\partial w_{ij}^k}$, $\Delta b_i^k = -\alpha \frac{\partial L(J)}{\partial b_i^k}$, which send backward from layer m, the output layer, to layer one, the input layer to update weights [47].

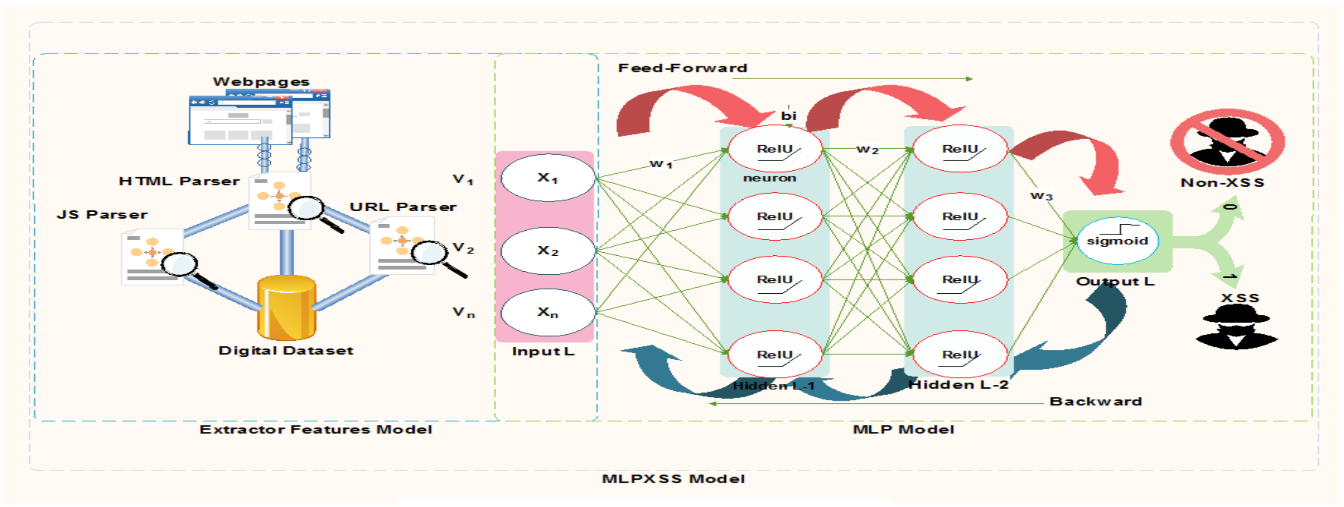

FIGURE 2. MLPXSS DETECTION SCHEME.

IV. EXPERIMENTAL DESIGN AND EVALUATION

A. THE DATASET

The final dataset that has been created in a randomized and uniform manner consist of 138,569 samples in total, where the benign samples are 100,000, and the malicious samples are 38,569 with dimensional of 41 features. Ideally, the ultimate model estimation should be performed on a held-out test dataset that never used before, neither for training the model nor for tuning the model parameters, so that they supply an unbiased sense of model effectiveness [48]. However, estimating score may get on a single validation set and is unlikely to reflect the model performance in general. Therefore, the dataset was split randomly and separated by the dynamic feature extractor model into two parts with a partition ratio of 80%:20%. The first part is training dataset including 11,0855 samples labelled as [0 Benign, 1 Malicious] and the second part is a Hold-Out test dataset containing 27,714 samples used only to estimate the performance of the final and fully-trained model. The dataset subdivisions are shown in Table 1.

TABLE 1. DATASET SUBDIVISIONS

Dataset	Benign	Malicious	Total
Training	80,033	30,822	110,855
Testing	19,967	7,747	27,714
Total	100,000	38,569	138,569

B. PERFORMANCE EVALUATION METRICS

For the binary classification problem (benign or malignant), the confusion matrix is employed as an evaluation parameter [49]. In a confusion matrix, TN points that a benign case was correctly labelled as benign, FP denotes that a benign case was incorrectly labelled as XSS attack. As for the performance metrics, FN indicates that an XSS was incorrectly identified as benign, TP represents that an XSS is correctly identified as an attack. For a comprehensive evaluation, the proposed scheme performance was tested and evaluated in various measurements such as Accuracy overall, Misclassification Rate (Error Rate), Precision, and Detection rate (DR), False Positive, and F-score. The performance of the proposed scheme is evaluated as follows, as shown in Formulas (7)-(13).

$$Accuracy\ overall = \frac{(TP+TN)}{(TP+TN+FP+FN)} \tag{7}$$

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

$$Precision = \frac{TP}{(TP+FP)} \tag{9}$$

$$Detection\ Rate\ (DR) or\ Recall = \frac{TP}{(TP+FN)} \tag{10}$$

$$FP\ Rate\ or\ Fall-out = \frac{FP}{(TN+FP)} \tag{11}$$

$$F-Score = 2\left(\frac{(Recall \times Precision)}{(Recall + Precision)}\right) \tag{12}$$

$$Area\ Under\ the\ Curve = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{13}$$

C. EXPERIMENTAL ENVIRONMENT

To construct and select a model that accurately captures the regularities in its training data and skillfully generalizes unseen data. The performance evaluation of any model with single parameter accuracy is not sufficient to judge its performance. Since a single large hidden layer is enough for an approximation of most functions [50], we first start with the shallow model, including one hidden layer. The size of the hidden layers was determined by taking the mean of the number of neurons at the input layer plus a number of neurons at the output layer as a baseline. More specifically, the initial model has an input layer with 41 neurons, one hidden layer with 22 neurons and an output layer with a single neuron. All the other hyper-parameters were fixed such as Adam optimizer, activation functions ReLU, sigmoid for hidden and output layers, respectively, and the epoch equals to 100.

The entire experimental was done on the operating platform LinuxMint-19-tara, Kernel 4.15.0-42-generic, 16 GB RAM, Intel© Xeon© CPU E-5-2620 v3 @ 2.40GHz, GPU NIVIDA (Quadro K220). The Python framework version is 3.6.7 and Keras version (2.2.4) with Tensorflow (1.12.0).

A k-fold cross-validation approach is used as a baseline estimation of our model on training dataset only by using a wrapper method for MLP models to be used as classification estimator in scikit-learn library. This approach involves randomly splitting the training dataset up into a set of k-folds, of approximately equal sub-datasets size. Then for each unique fold, it takes the k sub-dataset as a test dataset, and the remaining $k - 1$ sub-datasets is fit to model as a training data set. Finally, the k outcomes from the folds are averaged to make a single estimation [49].The importance of this approach lies in allowing each observation give to be utilized in the test dataset one time and used to train the model k-1 times. However, for the initial estimate and bias-variance tradeoff, an average accuracy of the ten evaluations is selected and computed the standard deviation to look at the variance. The results obtained are 0.989788 as mean accuracy, and the variance equals to 0.001022, which means that our model falls into the low bias and low variance category of bias-variance tradeoff categories as shown in Table 2.

Although for some functions, a single hidden layer is optimal, a single-hidden-layer-solution may be quite inefficient for others as compared to solutions with more layers [45]. Furthermore, the hidden layers are increased to two and kept all other hyper-parameters fixed as in a shallow model. The same estimated approach is used for this, which computes the mean accuracy and the standard deviation. The results obtained with additional hidden layer are more promising, where the mean accuracy equal to 0.991195 and variance is 0.000759 as seen in Table 2. It can be observed that by increasing the hidden layer to two layers, it also increased the accuracy and decrease the variance.

The model design is stabilized to obtain optimum performance, and the number of hidden layers is fixed to 2. Hence, the effectiveness of most Hyper-parameters of our model was studied including several combinations of these values such as the number of neurons in hidden layers with 22,32 and 42, mini-batch size with 32,64,128, the number of epochs with 100,200 and 300, and with the optimizers such as Adam, rmsprop, and SGD. Furthermore, standardize the dropout rate for regularization in an attempt to limit over-fitting and therefore, improve the model's ability to generalize. The Grid-search has been used as a model hyper-parameter optimization technique with 10-fold cross-validation approach. Table 3 shows the effect of the number of neurons at the hidden layer and epoch's number as well as the best values that were optioned over this testing. Table 4 shows the drop rate evaluated along with different optimizer, while the batch size is fixed to 60, the number of neurons at each hidden layer is fixed to 42 and number of the epochs are fixed to 300 with 10-fold cross-validation approach.

TABLE 2. THE BIAS-VARIANCE TRADEOFF VALIDATION ON SHALLOW AND DEEP MLP MODEL

K-Fold=10	1	2	3	4	5	6	7	8	9	10	Mean Accuracy	Variance
Shallow (%)	99.03	98.99	98.82	98.96	99.12	98.96	98.80	99.03	98.95	99.12	98.98	0.10
Deep (%)	99.10	99.11	99.02	99.03	99.25	99.09	99.11	99.19	99.08	99.23	99.12%	0.08

TABLE 3. EFFECT OF THE NUMBER OF NEURONS AT HIDDEN LAYER

N.Epoch	N.Neurons	Batch Size	Mean loss	Mean Accuracy
100	22	60	0.000716	0.991637
100	32	60	0.000380	0.992043
100	42	60	0.000788	0.992125
200	22	60	0.000849	0.992016
200	32	60	0.000800	0.992215
200	42	60	0.000863	0.992305
300	22	60	0.000805	0.991917
300	32	60	0.000971	0.992251
300	**42**	**60**	**0.000681**	**0.992531**

TABLE 4. EFFECT OF DIFFERENT OPTIMIZERS AND DROP RATE

Optimizer	Dropout rate	Mean loss	Mean Accuracy
Adam	0.0	0.000743	0.991809
rmsprop	0.0	0.000824	0.990943
SGD	0.0	0.001019	0.981597
Adam	**0.1**	**0.000484**	**0.992494**
rmsprop	0.1	0.000773	0.990609
SGD	0.1	0.001412	0.981092
Adam	0.2	0.000571	0.992368
rmsprop	0.2	0.000656	0.990203
SGD	0.2	0.001618	0.980424

V. RESULTS AND DISCUSSION

After tuning the model with selected parameters, the final model was configured to achieve optimal results. More specifically, the final model has two hidden layers, one input that taken the input vector from the DEF model, and the other is the output layer. Neurons at the input layer, hidden layer1, hidden layer 2, and output are 41, 42, 42, and 1 respectively.The activation function at hidden layers is rectified linear function (ReLU), whereas the activation function for the output layer is the logistic function (sigmoid). The model used cross-entropy as the loss function; furthermore, the model takes the minibatch approach as the updating rule for large-scale dataset weights. The mini-batch dataset was set up to 64. The Adam optimizer is used to minimize the loss function with learning a rate set to the default value= 0.001, the epochs are fixed to 300, and the drop rate is equal to 0.1.

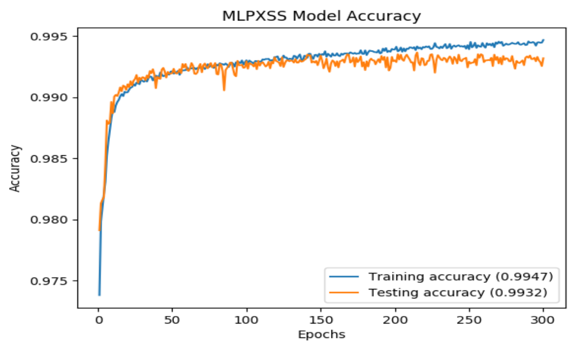

FIGURE 3. MODEL ACCURACY ON TRAINING AND TESTING DATASET

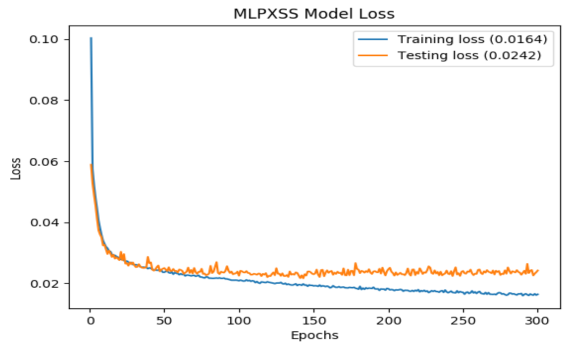

FIGURE 4. MODEL LOG LOSS VALUES OVER TIME

Furthermore, to make a real sense of the model accuracy, which can help in developing greater confidence in the developed model, the proposed model is tested with a new large-scale-real-world dataset. Thus, we fitted the final model with on the whole training dataset and tested on the held-out test dataset.

The results obtained on held-out test dataset achieved 99.32% of accuracy, as shown in Fig. 3 and model loss over time in Fig. 4. While the precision is 99.21 % for XSS class and detection rate up to 98.35%. The false positive rate is tend-to-zero, which represent 0.31 %. We obtained the global value of quality for the complete taxonomy of our model using the weighted averages (micro average), and macro averages of both classes. Table 5 shows the full report generated based on the confusion matrix for testing model performance and Fig. 5 shows the confusion matrix.

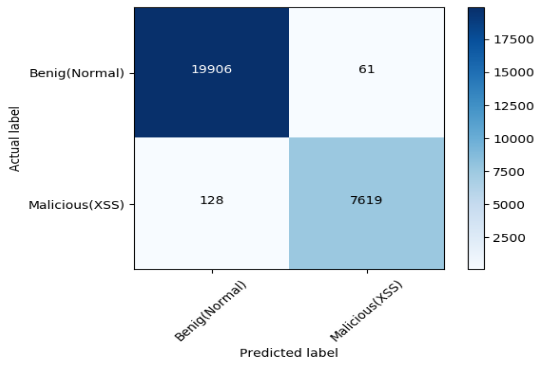

FIGURE 5. CONFUSION MATRIX ON TEST DATASET

To demonstrate the detection performance of our proposed scheme to avoid false classification, and clarify the discriminative robustness of our scheme, the ROC (receiver operating characteristic) curve of mode testing are plotted as shown in Fig. 6 (a) and Fig. 6 (b). The proposed method shows state of the art ROC curve for our model, with an area under the curve (AUC) equals to 0.9902. While, the False Positive rate is tend-to-zero, which equal to 0.0031, and the corresponding True Positive Rate tends to one, which equals to 0.9969 on the testing dataset. This implies that our proposed scheme is beneficial in detecting XSS-based attacks. Furthermore, it has a real ability to detect zero-day XSS-based attacks efficiently and effectively.

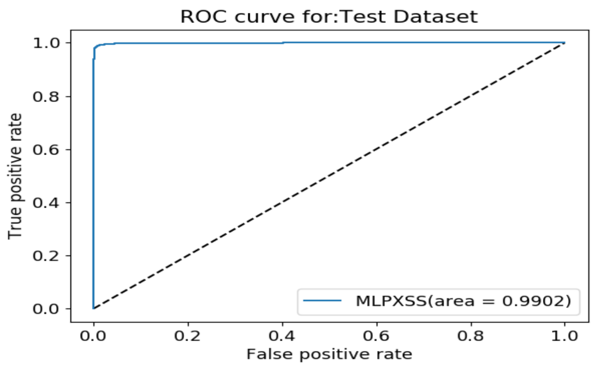

FIGURE 6. (a) MLPXSS MODEL ROC CURVE

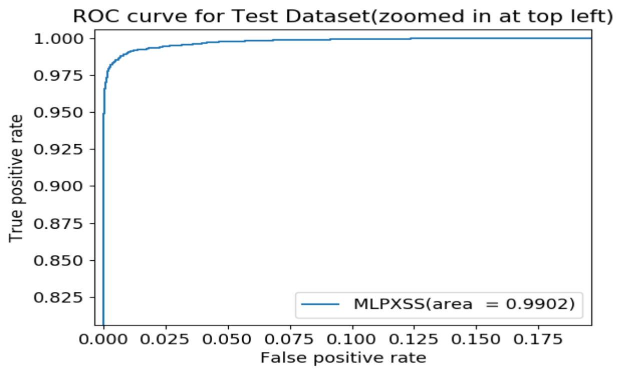

FIGURE 6. (b) MLPXSS MODEL ROC CURVE ZOOMED IN AT TOP LEFT

A. COMPARISON WITH OTHER CLASSIFIERS

For the strict verification of data quality, a new experiment is implemented using the same dataset with different machine learning algorithms such as nonlinear SVM Radial Basis Function (RBF) kernel with c value =1.0, k-nearest neighbor (k-NN) with neighbors =5 and ensemble method AdaBoost classifier along with proposed methodology motioned above. The goal is to verify whether the dataset employed by the proposed method has advantages and able to works on different techniques as well as to compare MPLXSS with different algorithms. The results of the various algorithms proved the efficiency and effectiveness of the proposed mechanism through which the data collection and extraction were obtained as shown in Table 6 (a) and Table 6 (b). Also, the results of the AdaBoost algorithm are approximate to the results of neural networks model in terms of accuracy and less than in terms of detection rate. Therefore, we can conclude that data quality and features extraction are the key components for precision and robustness of these classifiers since most of them are performing well on the same data set.

B. COMPARISON WITH PREVIOUS WORKS

To objectively evaluate the proposed scheme, MLPXSS is compared with the methods proposed by Wang *et al.* [23], Rathore *et al.* [24], and Fang *et al.* [28], since the malicious data coming from the XSSed. Moreover, the detected scheme is also compared with the methods proposed by Likarish *et al.* [22] and Wang *et al.* [25]. Since the XSS considerably utilize malicious JavaScript as a portion of the attack, furthermore, the code injection attack is closely associated with JavaScript code. All these proposed methods have been discussed in the related work section. Furthermore, the best case is selected for each method to be compared with our MLPXSS. Table 7 compares the results of our scheme with other proposed schemes. From the results, it can easily be foreseen that the proposed (MLP-XSS) scheme supersedes each of the previously proposed schemes with a significant margin. It maintained to achieve high Precision and high Recall, which is evident through F-score, which is another metric widely used to convey these both metric into one unified metric. Moreover, FPR is also near to zero and the AUC-ROC curve is also higher than 90.02%, which is one of the essential evaluation metrics for any classification model's performance that visualizes classification quality and presents a complete sense of the model performance.

TABLE 5. EXPERIMENTAL RESULTS ON TEST DATASET

Class	Precision	Detection Rate	F-score	Accuracy overall	FP Rate	Misclassification rat	ROC
Non-XSS	0.9924	0.9969	0.9953	0.9932	0.0031	0.0068	0.9902
XSS	0.9921	0.9835	0.9877				
weighted avg (micro avg)	0.9932	0.9932	0.9932				
macro avg	0.9928	0.9902	0.9915				

TABLE 6. (a) RESULT COMPARISON MLPXSS WITH OTHER CLASSIFIERS

Classifier	Non-XSS			XSS		
	Precision	Detection Rate	F-score	Precision	Detection Rate	F-score
MLPXSS	**0.9924**	**0.9969**	**0.9953**	**0.9921**	**0.9835**	**0.9877**
Gaussian NB	0.9756	0.9852	0.9804	0.9609	0.9365	0.9486
SVM	0.9733	0.9994	0.9862	0.9985	0.9293	0.9626
K-NN	0.9882	0.9957	0.9920	0.9888	0.9694	0.9790
AdaBoost	0.9920	0.9970	0.9945	0.9923	0.9793	0.9858

TABLE 6. (b) RESULT COMPARISON MLPXSS WITH OTHER CLASSIFIERS

Classifier	Accuracy overall	FP Rate	Misclassification rat	ROC
MLPXSS	**0.9932**	**0.0031**	**0.0068**	**0.9902**
Gaussian NB	0. 9716	0.0148	0.0284	0.9609
SVM	0.9798	0.0005	0.0202	0.9644
K-NN	0.9884	0.0043	0.0116	0.9826
AdaBoost	0.9921	0.0030	0.0079	0.9882

TABLE 7. RESULT COMPARISON MLPXSS WITH OTHER METHODS

Approaches	Accuracy overall	Precision	Recall/Detection rate	F-score	FP Rate	Misclassification rate	AUC-ROC
Likarish et al. [22]	-	92.00%	74.20%	76.40%	13.05%	-	80.58%
Wang et al. [23]	-	94.10%	93.90%	93.90%	4.20%	-	94.85%
Rathore et al. [24]	97.20%	97.70%	97.10%	97.40%	8.70%	2.80%	94.42%
Wang et al. [25]	94.82%	94.90%	94.80%	94.80%	4.14%	5.18%	95.33%
Fang et al. [28]	-	99.50%	97.90%	98.70%	1.90%	-	98.00%
MLPXSS	**99.32%**	**99.21%**	**98.35%**	**98.77%**	**0.31%**	**0.68%**	**99.02%**

VI. CONCLUSIONS

In this research, the ANN-based scheme is proposed to detect the XSS-based web-applications attack. Three models are designed in a novel manner. The first model is concerned with the quality of the raw data and random crawling. The second model deals with extracting digital data as features of raw data and providing neural networks with these digital features, and the third is ANN-based multilayer perceptron model that takes the digital data, processes and classifies the final prediction result of XSS-attack problem. Our model performs a prediction of security threats such as XSS attack, which can reflect in the form of a warning to users who can cancel the subsequent treatment of the pages. It acts as a security layer either for the client-side or the server-side. The experiments were conducted effectively on the test dataset, and the comparison was performed on existing methods. Based on the results, it can be concluded that the proposed scheme outperformed the state-of-the-art techniques in different aspects of measurements. In our future work, we will improve and apply this scheme to detect XSS attacks in the real-time detection system.

REFERENCES

[1] B. K. Ayeni, J. B. Sahalu, and K. R. Adeyanju, "Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System," *J. Comput. Networks Commun.*, vol. 2018, pp. 1–10, Aug. 2018.

[2] WhiteHat, "2018 Application Security Statistics Report Security. [Online]. Available: https://info.whitehatsec.com/Content-2018StatsReport_LP.html , Accessed on: Oct. 1, 2018

[3] O. Andreeva et al., "Industrial Control Systems Vulnerabilities Statistics", Kaspersky Lab, Report, 2016. [Online] Available: https://media.kasperskycontenthub.com/wpcontent/uploads/sites/43/2016/07/07190426/KL_REPORT_ICS_Statistic_vulnerabilities.pdf

[4] National Vulnerability Database (NVD), Vulnerabilities, [Online]. Available: https://nvd.nist.gov/vuln/

[5] H. Adams *et al.*, "2018 Application Security Research Update", *Micro Focus®Fortify Software Security Research Team*, 2018. [Online] Available: https://www.microfocus.com/en-us/assets/security/application-security-research-update-2018, Accessed on: Jun.5, 2018.

[6] S. Elad, S. Renny, M. Martin, and F. Amanda, "State of the Internet Security-Credential Stuffing Attacks Report", vol. 4, no. 4, Akamai, 2018.

[7] OWASP, OWASP Top 10 – 2017[Online]. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf, Accessed on: Feb. 1, 2018

[8] S. Gupta and B. B. Gupta, "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art," *International Journal of Systems Assurance Engineering and Management*, vol. 8. Springer India, pp. 512–530, 2017.

[9] S. Gupta, B. B. Gupta "BDS: browser dependent XSS sanitizer." In *Handbook of Research on Securing Cloud-Based Databases with Biometric Applications*. IGI Global, 2015. 174-191.

[10] H. B. Kazemian and S. Ahmed, "Comparisons of machine learning techniques for detecting malicious webpages," *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1166–1177, 2015.

[11] U. Sarmah, D. K. Bhattacharyya, and J. K. Kalita, "A survey of detection methods for XSS attacks," *J. Netw. Comput. Appl.*, vol. 118, pp. 113–143, 2018.

[12] R. Sajjad, H. Mamoona, G. Zartasha, A. Ansar, and J. Hasan, "Systematic Review of Web Application Security Vulnerabilities Detection Methods," *J. Comput. Commun.*, vol. 03, no. 09, pp. 28–40, 2015.

[13] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Inf. Softw. Technol.*, vol. 74, pp. 160–180, 2016.

[14] S. Lekies, K. Kotowicz, S. Groß, E. A. Vela Nava, andM. Johns, "Code-Reuse Attacks for the Web," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, 2017, pp. 1709–1723.

[15] J. Murphree, "Machine learning anomaly detection in large systems," in *2016 IEEE AUTOTESTCON*, 2016,pp. 1–9.

[16] A. Abeshu and N. Chilamkurti, "Deep Learning: The Frontier for Distributed Attack Detection in Fog-To-Things Computing," *IEEE Commun. Mag.*, vol. 56, no.2, pp. 169–175, Feb. 2018.

[17] B. B. Gupta. Computer and Cyber Security: Principles, Algorithm, Applications, and Perspectives. CRC Press, Taylor & Francis, 2018

[18] B.B. Gupta, D.P. Agrawal, S. Yamaguchi, Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security, IGI Global Publisher, USA, 2016.

[19] Gupta, B.B., Gupta, S. and Chaudhary, P., "Enhancingthe browser-side context-aware sanitization of suspicious HTML5 code for halting the DOM-based XSS vulnerabilities in cloud". *International Journal of Cloud Applications and Computing (IJCAC)*, *7*(1), pp.1-31,2017

[20] Gupta, S. and Gupta, B.B., JS-SAN: "defense mechanism for HTML5-based web applications against JavaScript code injection vulnerabilities". *Security and Communication Networks*, *9*(11), pp.1477-1495, 2016.

[21] S. Gupta, B.B. Gupta. "PHP-sensor: a prototype method to discover workflow violation and XSS vulnerabilities in PHP web applications." In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, p. 59. ACM, 2015.

[22] P. Likarish, E. Jung, and I. Jo, "Obfuscated malicious javascript detection using classification techniques," *2009 4th Int. Conf. Malicious Unwanted Software, MALWARE 2009*, pp. 47–54, 2009.

[23] R. Wang, X. Jia, Q. Li, and S. Zhang, "Machine learning based cross-site scripting detection in online social network," *Proc. - 16th IEEE Int. Conf. High Perform. Comput. Commun. HPCC 2014, 11th IEEE Int. Conf. Embed. Softw. Syst. ICESS 2014 6th Int. Symp. Cybersp. Saf. Secur.*, pp. 823–826, 2014.

[24] S. Rathore, P. K. Sharma, and J. H. Park, "XSSClassifier: An efficient XSS attack detection approach based on machine learning classifier on SNSs," *J. Inf. Process. Syst.*, vol. 13, no. 4, pp. 1014–1028, 2017.

[25] Y. Wang, W. Cai, and P. Wei, "A deep learning approach for detecting malicious JavaScript code," *Secur. Commun. Networks*, vol. 9, no. 11, pp. 1520–1534, Jul. 2016.

[26] Pan, Y., Sun, F., White, J., et al: 'Detecting web attacks with end-to-end deep learning', (Vanderbilt University, Melbourne, FL, USA, 2018), pp. 1–14.

[27] J. W. Stokes, R. Agrawal, and G. McDonald, "Neural classification of malicious scripts: A study with javascript and vbscript." *arXiv preprint arXiv:1805.05603*, 2018.

[28] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross Site Scripting Detection Based on Deep Learning," *Proc. 2018 Int. Conf. Comput. Artif. Intell. - ICCAI 2018*, pp. 47–51, 2018.

[29] A. E. Nunan, E. Souto, E. M. dos Santos, and E. Feitosa, "Automatic classification of cross-site scripting in web pages using document-based and URL-based features," in *2012 IEEE Symposium on Computers and Communications (ISCC)*, 2012, pp. 000702–000707.

[30] Developers S. Scrapy, Sep 2018. [Online] Available: https://scrapy.org/.

[31] M. R. Henzinger, A. Heydon, M. Mitzenmacher, andM. Najork, "On near-uniform URL sampling," *Comput. Networks*, vol. 33, no. 1, pp. 295–308, 2000.

[32] Alexa, alexa-static. [Online] Available: http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. Accessed on: Jul. 1, 2018.

[33] F. Kevin, and D. Pagkalos, "XSSed. com. XSS (Cross-Site Scripting) information and vulnerable websites archive". [Online] Available: http://www.xssed.com/archive

[34] Open Bug Bounty. [Online] Available: https://www.openbugbounty.org/

[35] L. Richardson, "Beautiful Soup Documentation. [Online] Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/,2017.Accessed on: Jul. 5, 2018.

[36] J. Graham, html5lib. [Online] Available: https://pypi.org/project/html5lib/

[37] A. Hidayat, *Esprima*, Oct 2018. [Online] Available: https://github.com/jquery/esprima.

[38] M. Mohammadi, B. Chu, and H. R. Lipford, "Detecting cross-site scripting vulnerabilities through automated unit testing," in *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*, 2017, pp. 364–373.

[39] B. Cui, Y. Wei, S. Shan, and J. Ma, "The generation of XSS attacks developing in the detect detection," *in Advances on Broad-Band Wireless Computing, Communication and Applications, Lecture Notes on Data Engineering and Communications Technologies 2*,Springer International Publishing AG, 2016, vol. 2, pp. 353–361.

[40] OWASP, XSS Filter Evasion Cheat Sheet. [Online]. Available: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet, Accessed on: Sep.10, 2018.

[41] HTML Living Standard, WHATWG, Sep 2018. [Online] Available: https://html.spec.whatwg.org/, Accessed on: Sep. 20, 2018.

[42] R. Yan, X. Xiao, G. Hu, S. Peng, and Y. Jiang, "New deep learning method to detect code injection attacks on hybrid applications," *J. Syst. Softw.*, vol. 137, pp. 67–77, 2018.

[43] M. Michael and M. S. Lam, "Automatic generation of XSS and SQL injection attacks with goal-directed model checking," in *17th USENIX conference on Security symposium*, 2008, pp. 31–43.

[44] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural networks: an introduction*, Springer Science & Business Media, 2012.

[45] N. Smithing, "Supervised Learning in Feedforward Artificial Neural Networks", Cambridge, MA: MIT Press, 1999.

[46] K. Gurney. "An introduction to neural networks", *CRC press*. London, UK, 2014.

[47] D. Kingma and J. Ba. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*, 2014.

[48] S.J. Russell and P. Norvig, "Artificial intelligence: a modern approach", Malaysia, Pearson Education Limited, 2016.

[49] G. James, W. Daniela, H. Trevor, and T. Robert, "An introduction to statistical learning", Springer Texts in Statistics, Springer, New York, 2013.

[50] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning", in preparation for MIT Press, 2016. [Online] Available: http://www.deeplearningbook.org.