XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization

Preprint *in* Journal of Information Security and Applications · March 2021

DOI: 10.1016/j.jisa.2021.102813

5 authors, including:

Fawaz M. M. Mokbal
Beijing University of Technology
14 PUBLICATIONS **54** CITATIONS

SEE PROFILE

XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization

Fawaz Mahiuob Mohammed Mokbal[1,2], WANG Dan[1], WANG Xiaoxi[3], ZHAO Wenbin[1], and FU Lihua[1]

[1] College of Computer Science, Beijing University of Technology, Beijing, 100124, China

[2]Faculty of Computer Science, ILMA University, Karachi, Pakistan

[3]State Grid Management Institute, Beijing, 102200, China

ABSTRACT

With the widespread popularity of the Internet and the transformation of the world into a global village, Web applications have been drawn increased attention over the years by companies, organizations, and social media, making it a prime target for cyber-attacks. The cross-site scripting attack (XSS) is one of the most severe concerns, which has been highlighted in the forefront of information security experts' reports. In this study, we proposed XGBXSS, a novel web-based XSS attack detection framework based on an ensemble-learning technique using the Extreme Gradient Boosting algorithm (XGboost) with extreme parameters optimization approach. An enhanced feature extraction method is presented to extract the most useful features from the developed dataset. Furthermore, a novel hybrid approach for features selection is proposed, comprising information gain (IG) fusing with sequential backward selection (SBS) to select an optimal subset reducing the computational costs and maintaining the high-performance of detector' simultaneously. The proposed framework has successfully exceeded several tests on the holdout testing dataset and achieved avant-garde results with accuracy, precision, detection probabilities, F-score, false-positive rate, false-negative rate, and AUC-ROC scores of 99.59%, 99.53 %, 99.01%, 99.27%, 0.18%, 0.98%, and 99.41%, respectively. Moreover, it can bridge the existing research gap concerning previous detectors, with a higher detection rate and lesser computational complexity. It also has the potential to be deployed as a self-reliant system, which is efficient enough to defeat such attacks, including zero-day XSS-based attacks.

Keywords: Attack Detection, Cross-Site Scripting attack, Extreme Gradient Boosting, Machine learning, Hybrid Features Selection, Web Application Security.

1. Introduction

Cross-Site Scripting (XSS) vulnerabilities are one of the common high-risk cyber-attack of web applications, which have made the users, web applications, and even the industrial field alike at high risk [1]. The web application with these security vulnerabilities could permit cybercriminals to inject their malicious malware scripts into the webpages displayed to different end-users. Hence, it can be exploited to cause damages such as completely changing the appearance or behavior of the organization website, stealing sensitive enterprises' information or sensitive user' information, acting on behalf of the true user, and much more [2]. Various prevention and mitigation schemes have been proposed to counter XSS-based attacks either for client-side, server-side, or on the pair sides, using different analyzing methods such as static, dynamic, or hybrid [3]. However, the proposed solutions using existing traditional methods for such attack detection became insufficient due to the sophisticating and increasing forms of XSS payloads [4]; also, most of them are not scalable over time and having an un-ignorable case of false positives [5]. Recently, XSS had grabbed the distinction of the most widespread attack vector in 2019. According to PreciseSecurity research, nearly 40 % of all attacks recorded by security experts is XSS attacks. They noted that almost 75 % of prestigious companies across North America and Europe had targeted over 2019 [6]. Besides, the overall number of new XSS vulnerabilities in 2019 (2,023) increased by 30.2% compared to 2018 (1,554) and by 79.2% compared to 2017 (1,129) as per National Vulnerabilities Database (NVD) [7]. In 2018, the security report was released by the state of application security that considered XSS-based vulnerabilities as the second most common vulnerabilities [8]. It was enlisted among the top three attack vectors used against web applications in 2018 as per Akamai's State of the Internet Security Report [9], and still, it is on the pinnacle of 10 attack vectors in 2017 as per OWASP [10].

As mentioned above, a rapid increase in risk diverts our attention toward the modern world's challenges because of XSS-based attacks. Furthermore, it is also evident that the sophisticated XSS-based attacks have begun to emerge significantly, effectively, and pose a real threat to both companies and individuals. Artificial Intelligence (AI) techniques are becoming mainstream that readily available commercial technology for organizations and cyber-criminals to take full benefits in this regard. [11]. Therefore, the development of an efficient and robust cyber-defense mechanism with the latest AI schemes that have the potential to accurately and precisely detect these sophisticated XSS-based attacks is of keen interest to researchers and web-based communities.

Researchers are endeavoring to feature intelligence to improve detection probabilities employing AI techniques, which is evident from the considerable employ of machine learning (ML) techniques while detecting cyberattacks [12]. The security detection systems are trained by utilizing a dataset of previously known behaviors, and each group of behavior is recognized to be either malignant or benign. Although AI adds significant value in solving such problems, based on the literature review, Machine Learning (ML) and Deep Learning (DL) based XSS-detectors still

suffer from substantial fundamental shortage lies in remarkable missing cases (i.e., false-negative rate (FN)). FNs are a big concern than false positives (FP) since many attacks will override the detection system without being detected, the ending results in real threats and compromised security system. In fact, this issue is due to the shallow detection rate with these systems (i.e., the rates of attack detection are still far from 100%). Although the importance of this FN index in such security systems, the neglect of it was noted in most relevant research work, while most of them focus on FP. Besides, the majority of these proposed models also still have at least one of the essential issues, which is either (i) time-consuming to handle a huge amount of data and/or (ii) produce an un-ignorable case of false positive (FP rate). Hence, the question of whether advanced ML approaches can be used to increase the detection capabilities against XSS attacks is an important one for possibly strengthening the defenses against this kind of cyberattacks.

In this research, a novel detection framework, namely XGBXSS, is proposed to overcome the above-discussed shortages. The proposed framework is comprised of three main principles including (i) the improved feature extraction process proposed in our previous work [13], by enhancing the model's ability to scale up the search for features using the dictionary search function,(ii) fusing information gain (IG) with sequential backward selection (SBS) to perform a novel hybrid feature selection approach for select optimal subset while maintaining powerful performance in all measurements, (iii) and adopt ensemble learning using XGBoost-based algorithm with an extreme optimization approach.

Furthermore, this research also features an in-depth experimental evaluation of the proposed scheme using various performance evaluation metrics. The proposed framework has achieved avant-garde results on the newly employed testing dataset with accuracy, precision, detection probabilities, FP rate, FN rate, and Area under the ROC Curve (AUC) scores of 99.59%, 99.50 %, 99.02%, 0.20%, 0.98%, and 99.41%, respectively. The principal contributions of this research could be summarized as the following:

- We proposed a novel ensemble XGBoost framework along with extreme parameters optimization on a realistic and up-to-date XSS dataset, which covers 160 features for the detection task to provide higher accuracy and higher detection rate.
- Fused information gain (IG) with a sequential backward selection (SBS) approach is also introduced to select the most optimal features from the dataset, aiming to decrease the computational requirements with an improvement in the performance simultaneously.
- We derivative the best subset of features composed of 30 features capable of characterizing XSS scenarios efficiently.
- This research also features an in-depth experimental evaluation of the proposed framework using various performance evaluation metrics, which proved that the proposed an XGBXSS detection is capable of featuring robustness, high precision, high detection probability, and less in computational complexity that makes it easier, lighter and faster to deploy.

The remaining of this paper is systematized as follows. Section 2 discusses related work and highlights the previous studies' gaps, which are our proposed framework's primary focus. Section 3 offers the key details about this research's mechanism and techniques, including the Enhanced Feature Extraction (EFE) model, hybrid feature selection method, and model construction for the detection task. Section 4 presents the strategy of experimental design and extreme parameter optimization of this research. While Section 5 shows details of results and discusses the comparative analysis of the proposed framework with various existing techniques reported in the literature. Finally, Section 6 concludes this research, focusing on its significance and highlighting key future study directions.

2. Related work

Many traditional mechanisms against XSS attacks were proposed to be applied on either client-side [14], the server-side [15], or both [3] and are analyzed using various approaches on different attacks vectors. These analysis methods could be (i) static analysis, which is the review of web application code including source code, byte-code, or binary code, to disclose how the data or control will flow at runtime before the application is being executed [16]. However, due to the complexity and technical limitations, some of the static analysis approaches cannot observe the existence of input-validation routines and results in the generation of false positives. (ii) The second analysis method is dynamic, which investigates the data acquired for application execution duration to look at vulnerabilities [17]. Dynamic analysis is mostly achieved in the testing period at some stage in the development or runtime once the software package is released. However, the shortcoming of dynamic analysis is that it can only detect vulnerabilities inside the execution paths and cannot locate vulnerabilities in the code's not yet executed components. (iii) The other one is the hybrid method, which combines each of the static and dynamic evaluation processes [18]. However, the solutions methods that have been presented, including mitigation tools, detection, or prevention for such assaults, became insufficient and inadequate [4]. Also, there is no single defense solution that can successfully and effectively mitigate XSS attacks [19] or can fully mitigate flaws that exist within the program's source code [20]. Moreover, the majority of them are incompetent and powerless to detect sophisticated-XSS attacks [21][22]. The protection approaches to eliminate such attacks by employing conventional

detection strategies are also complicated [23].
Numerous works have been published that employ AI techniques to find-out and block XSS-based attacks. The author in [24] used Support Vector Machine (SVM), Alternative Decision Tree (ADTree), and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) to determine the capabilities of the proposed 65 features for the detection of embedded malicious JavaScripts (JS) and achieved the precision score of 0.92 with the SVM classifier. However, the research did not mention the FN rate, in addition to the low detection rate and high FP rate. The author in [25] used ADTree and AdaBoost schemes with ten-folds cross-validations to detect an XSS based attack in social networks and achieved the precision and recall scores of 0.941 and 0.939 with the AdaBoost scheme. However, the results represented a shallow detection rate, with a high FP rate, i.e., almost equivalent to 4.20%. The researchers in [26] divided the obtained 25 features into three groups, i.e., URLs, SNSs, and webpages. The intention is to detect an XSS based attack using machine learning schemes, and the authors declared that with the Random Forest detector, the accuracy could be increased up to 0.9720 and false positive rate up to 0.087. However, the FP rate is equal to 8.7%, which still considered high. Further, the research did not mention the FN rate (real threat).Furthermore, researchers have also exploited deep learning (DL) schemes to increase the detection rate of said attacks using advanced features of the deep learning concept, which lies in its potential to output high-level representations of these attributes and predict hidden attributes using the training data. The researchers in [27] detected malicious codes using stacked denoising autoencoder model and reduced the dimensionality using sparse random projections. The results were visible in terms of accuracy (i.e., 0.9490) with a detection rate and false positive rates of 0.9480 and 4.2%, respectively. In [28], the author proposed to use an unsupervised deep learning scheme, i.e., Robust Software Modeling Tool (RSMT) to monitor and capture features that were representative of the application behavior during the runtime. Various supervised and unsupervised techniques were used to evaluate the performance of the proposed schemes; where, RSMT has caused overhead issues; moreover, supervised ML technique induced poor results (i.e., achieved a maximum score of 0.728 for the detection of an XSS with the SVM classifier). Additionally, the unsupervised DL scheme performance was not adequate, i.e., around 0.906. The researcher in [29] proposed to use Script Net (i.e., a deep recurrent neural classification system) for the detection of malicious JavaScript or VBScript through static and dynamic analysis. The method used an antivirus engine for analyzing scripts and labeling of data. The results of the model were not convincing, where the LaMP model achieved a detection rate of 65.9%, and the CPoLS model achieved the TPR of 45.3 % with a false positive rate (FPR) of 1.0%. Another researcher proposed to use DeepXSS (a deep learning model) [30] that contains decoding, generalization, and tokenization process. The proposed deep learning model (LSTM) was given with word2vec as input for extracting XSS features for the training. The model achieved precision, recall, and a false positive rate of 0.995, 0.979, and 0.019, respectively, with a higher FPR of 1.9%, which is not a good sign of an efficient detection scheme; however, webpages also contain JavaScript and HTML codes, which makes word vectors unsuitable for the desired detection task. Lastly, deep learning schemes require enormous data for the development of an efficient and state of the art detection model whereas, some of its well-known schemes, i.e., CNN or RNN are also not suitable to entertain such attacks. The primary reason is that a malicious code of an XSS attack is mostly a JS code. Hence, JS has un-standard encoding; there exist many incongruous, unmeaning strings and Unicode symbols.
Moreover, JS uses data encapsulation, drivel strings insertion, code rearrangement, and substitute procedures, which leads to various sophisticating forms of XSS payloads that are difficult to enumerate. Therefore, it is difficult to rely on a particular pattern to detect such attacks. Furthermore, the reported results of deep learning schemes are still unsatisfactory and require higher computational cost with an intensified complexity for the deployment of such techniques. In [31], the authors proposed a hybrid analysis (static and dynamic) approach to detect malicious web pages. The 25 features are extracted which belong to three groups (URL, HTML, and JavaScript). Using a decision tree algorithm for binary classification, the precision, detection rate, F-score, and ROC curve of 95.2%, 88.2%, 91.6%, and 94.79% respectively are achieved. However, this result is not sufficient to avoid the risk of attacks, the detection rate is low compared to the precision, and consequently, the false-negative will be high. This means that many attacks will pass through the system undetected.
In [13], the authors proposed a scheme based on deep neural networks to deal with the XSS attacks. The proposed detection scheme includes a large dataset, feature extraction method and Multilayer Perceptron (MLP) for the detection task. Even though the reported results are impressive with the accuracy, detection rate, FP rate, and AUC scores of 0.9932, 0.9835, 0.003, and 0.9902 respectively, the rate of attack detection is still far from 100% (i.e., 98.35 %); where we aim in this research to bridge such an existing research gap.

3. Detection Methodology

In recent years, web-based XSS attacks are the most crucial concern for security analysts. They are caused due to existing security bugs in the websites that are dawned by the features provided by dynamic web applications such as hyperlinks, scripts, HTML tags, and numerous useful functions. However, these features are extremely beneficial for developing an efficient web-based application but exhibit serious security concerns that need immediate attention. Based on these concerns, we propose to use a novel framework, which is a platform-

independent and can overcome these concerns with a maximum detection rate and minimum FN and FP rate. The proposed framework is comprised of three main principles, which include the derivation of an improved feature extraction process proposed in [13], derivation of a useful feature vector with the capability of the characterization of XSS scenarios using the fused information gain with sequential backward selection (SBS) as a hybrid approach, and adopt ensemble learning using XGBoost-based algorithm with extreme optimization approach for classifying and detecting malicious codes. Fig.1 presents the proposed framework comprehensively. The details of the proposed framework are as follows.

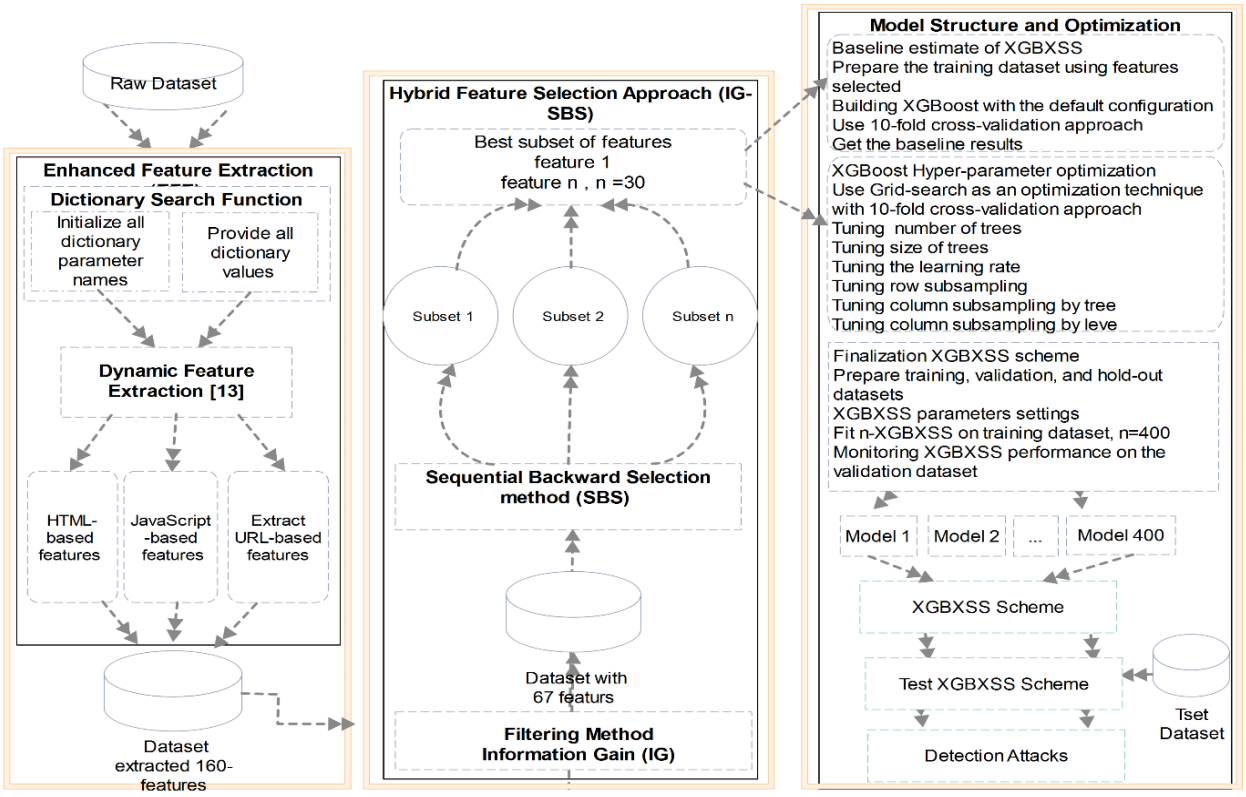


Fig. 1 - The schematic framework of the proposed scheme

3.1. Enhanced Feature Extraction (EFE) Model

To avoid the risk of a lack of a qualitative and adequate training dataset in developing an XSS detection framework using ML, Besides, in the security domain, it is necessary to include the most potential possibilities that might be used in the attacking vectors. For this purpose, The dynamic feature extraction of our previous work in [13], is enhanced in this research. The enhancing aims are to consider the majority of the possible cases that can be used in the XSS attack vector. Thus, the overall performance of the detection framework will be improved. The feature extraction model's main improvement is to extend features vectors search space to find the most related features and extract them digitally from the entire raw data. To do so, we used a brute-force search method on the entire raw dataset with the help of a dictionary argument as a map of the model parameter name and a list of values to searching for and extraction. After that, using our proposed feature selection technique, the best optimal subset of features can be obtained. The dictionaries have been created to include (i) web page-based features, and (ii) URL-based features as shown in the enhanced Algorithm 1. As a result, labeled samples covering 160 potential features for the detection task are extracted digitally to provide accurate and high detection rates and minimum FN and FP. Later, the best subset of 30 features is extracted. The details of this model are described in the subsequent subsections.

3.1.1. Raw Dataset

This research uses a realistic and up-to-date XSS raw dataset that was proposed in our previous work [13]. The dataset has been created by crawling the top 50,000 websites ranked by Alexa using a random walk integrated with jump algorithm for benign sample, and by crawling XSSed [32] and Open Bug Bounty [33] for malicious samples. The raw dataset is composed of 138,569 webpages in total, in which 100,000 samples are benign and 38,569 samples are malicious.

3.1.2. Web page-based features

The aforementioned model deals with two groups of features. One is the features of the HTML document including HTML-based features, and the other is JS that exists inside the document that interacts with it.

3.1.2.1. HTML-based features

To extend HTML-based features search space, we provided a dictionary vectors to extract in the param_EFE argument. Dictionary vectors act as a map of the EFE_model parameter name and a list of values to examine and extract. These parameters are the HTML tag that could be used to construct the attack payload and imbed JS-based malicious codes [3][16]. (i.e., script, form, style, iframe, div, SVG, frame, textarea, embed, video, link, img, meta, etc.,).In addition, pseudo-protocol (i.e., e JavaScript: [code]) that probably employed to invoking an outer resource identically as a URL (i.e., redirection) [34]. HTML attributes that can be exploited to inject the harmful script (i.e., href, src, target, lowsrc, http-equiv, action, background, classid, etc.,) to trigger JS events. The events influence an HTML and JS and interact with them, which might be triggered based on the actions of a user (e.g., onkeypress, onclick) or perhaps by the browser actions (e.g., onerror, onload). As the JS script code itself can be executed by an event and using JS within an event routine (i.e., <img src=# onerror="alert ('XSS')">), therefore, it can be applied to any HTML type injection that using HTML elements, which can authorize any relevant event for the tag to be substituted (e.g., onblur, onclick) which leads to an in-depth quantity of variations for several injected payload vectors and may bypass most the domain filters. The attacker to hold XSS attacks can easily by exploiting such a way.
The EFE features extraction model is designed and integrated with the html5lib parser, where it is capable to parses the webpage in the same fashion as a web browser [35], and the BeautifulSoup [36], which is employed to navigate and search the parse tree (tree traversal), which can be carried out in parallel with html5lib. Based on this approach, we obtained the object of BeautifulSoup that contains the complete tree-structured data. Therefore, all the features of each webpage are extracted programmatically in a dynamic manner.

3.1.2.2. JavaScript-based features

The codes of JavaScript exist inside webpages and can interact with the DOM; JS has a powerful capability to either give extra features of the webpage or can develop an application inside the webpage that can operate directly within the web browser. The JS-based features extraction relies on three main stages as follows.
(i) We initialized and identified the complete set of features that could be used with JavaScript and are also related to an XSS attack using dictionaries. The potential features could be methods with the list of (getElementsByTagName, eval, write, prompt, confirm, alert, and fromCharCodefetch) that can induce XSS based attacks, min_function_calls (i,e., useful for benign), min_define_function, max_length, min_length (which are shorter in benign and longer in malicious), location_properties (i.e., cookies, document, and referrer) and Dom-Objects (i.e., location, windows, and document).
(ii) Extracted JS codes from all potential places within HTML elements. Since JS codes are mostly embedded in HTML elements which could recall in varied ways from an HTML document, as an example, tag (e.g., <script>), from the attribute that belongs to some tags (e.g., href= “JavaScript:” of ‘a’ tag), and/ or event (e.g., onload= “JavaScript”, onSubmit= “JavaScript: ”), etc.
(iii) In the final stage, each extracted webpage JS code is handled and converted into a series of tokens and syntax trees by employing Esprima parser [37], where it is acting in the same fashion JS engine proceed. Hence, lexical and syntactical analysis of JS code could be executed simultaneously. The Esprima parser is adjusted to a tolerant mode, and the tokens flag is set to be valid to keep the tokens that are obtained throughout the parsing process. Furthermore, the generator function is created to traverse the complete tree that takes the Esprima node and yields all child nodes of each tree level. Thus, this way has given us the complete ability for the traversal of all branches of the tree.

3.1.3. URL-based features

URL parameters are considered an essential suspect injection points for a non-persistent XSS injection [38]. The obfuscate/encode URLs in several ways is a common tactic by Cyber-criminals to bypass filtering tools or mask malignant code to trick or redirect users. In the same way, to increase the URL feature search space, we provided a dictionary vectors in the param_EFE argument as a map of the EFE_model parameter name and a list of values for the examination and extraction. The details of dictionary vectors that could be potential features use for our framework are as follows:
i) URL redirection (i.e., could exploit it to mislead users to redirect from the legal web page to a different webpage dominant by the Cyber-criminal) with the values of (document.URL, document.documentURI, document.URLUnencoded, document.baseURI, location, window.location, window.navigate, window.history, window.open, top.location, and self.location). ii) Domain knowledge features (i.e., domain numbers and IPs. iii) The Keyword parameters with the values of (login, signup, search, query, contact, redirect, URL) that usually appear on URL as the parameters name for assigning user input values. iv) The tags with the list of (i.e., script, style, iframe,

SVG, meta, img, etc), whose appearance within URL refers to a rising probability of an existence XSS. v) Properties of HTML with the list of (i.e., src, href, formaction, etc.) which may host JS malicious code employed within URL. vi) The Length of the URL, which is used during phishing detection. vii) Special characters with the list of (i.e., "< ", ">" and "/") that largely utilized to perform XSS attacks [40], and viii) keyword evil that is mostly used by attackers to reveals their signature.

The URL-based features extraction sub-model takes each URL of each webpage to decode and transform them as a string. Later, the Regular Expressions technique is carried out to correspond text patterns to extract URL-based features for the long-run model.

***Algorithm** 1: Enhanced Feature Extraction (EFE)*
***Input**: set of webpages WP {wp1,wp2...wpn}*
***Output**: dataset D with its set of features $X = \{x_0, x_1, \ldots, x_n\}$ for each webpage*

HTL{} ← Dictionary of HTML (tags)
HAL{} ← Dictionary of HTML (attributes)
HEL{} ← Dictionary of HTML (events)
DOL{} ← Dictionary of Dom Objects
JPL{} ← Dictionary of JS properties
JML{} ← Dictionary of JS methods
LUR {} ← Dictionary URL redirection
UFPL{} ← Dictionary URL frequent parameters
KEL{} ← Dictionary keywords evil
$H_{FV}\{\} \leftarrow \emptyset$
$JS_{FV}\{\} \leftarrow \emptyset$
$U_{FV}\ \{\} \leftarrow \emptyset$
***For** each html page $wp_i \in WP$ raw data file **do**:*
Create a tree structure of HTML data for the web page using html5lib parser
Navigating and searching the parse tree (tree traversal) using BeautifulSoup
Call Dictionary search:
Count each HTML tag, attribute and event from each node in tree
Compute the HTML length
***While** tags(script form, link, iframe, frame) , (event) **do**:*
Extract JavaScript_code;
Parse JavaScript_code as JS-Object using Esprima
Generate tokens of JS-Object
Extract Abstract Syntax Tree of JS-Object
For** each noed in tree **do
***For** each leaf in tree do*
If** $(leaf_i[type] in \{FunctionDeclaration\})$ **Then
Set [define_function]+=1
Elsif** $(leaf_i[type] in \{CallExpression, FunctionExpression\})$ **Then
Set [function_calls] += 1
End for
End for
For** each $tokens_i$ in JS-Object[tokens] **do
If** $(tokens_i[type] == Identifier)$ **Then
If** $(tokens_i[value] \in DOL)$ **Then
Set [DOL_($tokens_i$) [value]]+=1
Elsif** $(tokens_i[value] \in JPL)$ **Then
Set [JPL_($tokens_i$) [value]]+=1
Elsif** $(tokens_i[value] \in JML)$ **Then
Set [JML_($tokens_i$) [value]]+=1
Elsif** $(tokens_i[value] == string)$ **Then
Set Stringlist.append ($tokens_i$ [value])
End for
If** Stringlist >0 **Then
$JS[min\_length] = min\ (len(Stringlist))$
$JS[max\_length] = max\ (len(Stringlist))$
EndIf
End for
Return Webpage-based Features vector H_{FV} , JS_{FV} for future model uses
For** each $WP_i[URL]$ **do
Decode ($WP_i[URL]$)
Collect URL-length

Call Dictionary search:
Set each (tag, event, and attributes) =True if existing in URL
Set redirection parameters $\in$ $dic.LUR$ *= True if existing in URL*
Set for any document.Cookie = True if existing in URL
Count keywords_evil $\in$ $dic.KEL$ *existing in URL*
Count all frequent_parameter $\in dic.UFPL$ *existing in URL*
Count all domain/IP existing in URL
Return URL-based Features vector U_{FV} *for future model uses*

3.2. Hybrid Feature Selection Approach (IG-SBS)

Even though qualitative and adequate training dataset are primary attention while adopting ML techniques, too many features can also be a real hindrance (i.e., many data points could be noisy). Furthermore, complex models can be challenging to deploy in the real world. Reducing the dimensions of the dataset is an essential aspect in reducing the Size of the model; it helps in minimizing the computational cost of the model, making it easy to deploy. To derivative the best subset of features that capable of the characterization of a suitable XSS phenomenon, a hybrid approach is proposed. The hybrid approach includes fused Information Gain with the Sequential Backward Selection method.

3.2.1 Information Gain (IG)

The procedure aims to reduce the full dataset dimensions as the initial stage by employing IG as a filtering method. This method's key principle concept is to rank the most relevant subset of features by calculating the entropy for every feature, in which lower entropy of feature is a larger IG suggests. The subset of features with the highest IG is taken for further processing by passing them to the sequential backward selection stage. The formal baseline definition of Shannon's entropy for the IG gain calculation as:

$$\mathrm{E(x)} = -\sum_{\mathrm{k}\,\in \mathrm{target}} (\mathrm{P(x=k)} * \log_2(\mathrm{P(x=k)})) \quad (1)$$

Where $P(x = k)$ is the probability that the target feature takes a specific value k. we need to check which of the attribute most precisely splits data, which means the remains dataset with the lowest entropy (impurity). To get the remaining entropy, each attribute is used to split the dataset alongside these attributes' values. Accordingly, the entropy of the dataset calculates once a dataset has split along the attribute values. Next, we deduct this value from the originally computed entropy of the dataset to check how much this attribute splitting reduces the original entropy. Let assume D represents a training dataset along with its samples and corresponding features. If n is the number of targets and the training set includes d_i samples of target k. To calculate the estimated information desired to classify a given sample is given in formula 2.

$$I(d_1, d_2 \cdots d_n) = -\sum_{i=1}^{n} \frac{d_i}{D} \times log_2\left(\frac{d_i}{D}\right) \quad (2)$$

An attribute X with values $\{x_1, x_2, \dots, x_m\}$ can splitting the training set into V subsets $\{d_1, d_2, \dots, d_v\}$, where $d_j$is the subset that has the value $a_j$for attribute X. Further, let d_j contain d_{ij} samples of target k. The entropy of attribute X can compute as formula 3. Accordingly, the IG for attribute X is collected as formula 4.

$$E(X) = \sum_{j=1}^{v} \frac{d_{1j} + \cdots + d_{nj}}{D} \times I(d_1, d_2 \cdots d_n) \quad (3)$$

$$IG(X) = I(d_1, \cdots, d_n) - E(X) \quad (4)$$

Table 1 summarized the subset of 67 features with high IG, which is considered the input of the second stage of the feature selection process.

3.2.2 Sequential Backward Selection method (SBS)

Although the IG is deemed the most popular filter method, the bias towards features with a large range of possible values is the main problem [39]. These features return an entropy value that tends to zero, therefore obtained higher gain values than any other. Accordingly, these features obtain a score that maybe not reflected their importance to the training sample. To alleviate this bias, features subset outcome from the IG stage will be exposed to additional evaluation and reduction process by employing the warping method (i.e., SBS) to select the optimal subset of features. Consequently, improved the generalization capability and reduced complexity of the model could be achieved. SBS is a heuristic search algorithm-based warping method that uses a learning algorithm's performance to assess the usefulness of a feature set. The algorithm reconstructs the dataset into a subset of selected features. Therefore, it essentially allowed us to evaluate every feature subsets based on its significant impact on performance (i.e., starting with all and ending with essential features subsets). Initially, the algorithm begins to compute the criterion function for the complete features n. Then sequentially drop the feature x^- that least reduces the value of the objective function $J(Y - x^-)$ and compute the criterion for the size of candidate feature subsets $n - 1$. This procedure is repeated until the termination criterion is satisfied. The selection procedure of the hybrid IG-SBS

approach is abridged in algorithm 2. Table 2 summarizes the most various subsets along with their accuracy measurement during the selection process of the optimal subset. The subset of 30 features has been taken as the optimal subset, as it reduces the computational cost while maintaining performance.

***Algorithm** 2. Hybrid Feature Selection Approach (IG-SBS)*
***Input**: dataset D with its set of features* $X = \{x_0, x_1, \ldots, x_n\}$,
Size of the feature set n, Size of target feature subset d.
***Output**: Subset of Features with Model Performance*

IG_ procedure (X):
compute estimated information desired to classify a given sample

$$I(d_1, d_2 \cdots d_n) = -\sum_{i=1}^{n} \frac{d_i}{D} \times log_2\left(\frac{d_i}{D}\right)$$

while $1 \leq i \leq n$ **do**
Compute entropy for feature X_i

$$E(X_i) = \sum_{j=1}^{v} \frac{d_{1j} + \cdots + d_{nj}}{D} \times I(d_1, d_2 \cdots d_n)$$

Compute information gain for attribute X_i

$$IG(X_i) = I(d_1, \cdots, d_n) - E(X_i)$$

$X \leftarrow$ *the n features with the highest scores*
Return $X = \{x_1, x_2, \ldots, x_n\}$
SBS_ procedure(X):
Start with the full IG *feature* set $Y \leftarrow X$, k = n
For $j = 1 \rightarrow n - d$
Split data Y to x_train, x_test, Train classifier on x_train with existing features,
Compute classification performance, accuracy AC_k on x_test.
Remove the worst feature $x \leftarrow argmax\left(J\left(Y_k - \{x_j\}\right)\right), x_j \in Y_k$
Update $Y_{k-1} \leftarrow Y_k - \{x\}$
$k = k - 1$
End
Return $Y_k = \{y_j | j = 1,2,\ldots,k\ ; x_j \in Y\}$, *where* $k = (0,1,2,\ldots,n-d)$

Table 1: Features Obtained by Filtering Method

No.	Feature name	No	Feature name	No	Feature name	No	Feature name	No	Feature name
F0	url_length	F14	html_tag_embed	F28	html_attr_href	F42	html_event_onload	F56	js_method_getElementsByTagName
F1	url_special_characters	F15	html_tag_link	F29	html_attr_longdesc	F43	html_event_onmousedown	F57	js_method_getElementById
F2	url_tag_script	F16	html_tag_svg	F30	html_attr_profile	F44	html_event_onmouseout	F58	js_method_alert
F3	url_tag_iframe	F17	html_tag_frame	F31	html_attr_src	F45	html_event_onmouseover	F59	js_method_eval
F4	url_attr_src	F18	html_tag_form	F32	html_attr_usemap	F46	html_event_onmouseup	F60	js_method_fromCharCode
F5	url_event_onload	F19	html_tag_div	F33	html_attr_http-equiv	F47	html_event_onsubmit	F61	js_method_confirm
F6	url_event_onmouseover	F20	html_tag_style	F34	html_event_onblur	F48	html_number_keywords_evil	F62	js_min_length
F7	url_cookie	F21	html_tag_img	F35	html_event_onchange	F49	js_file	F63	js_min_define_function
F8	url_number_keywords_param	F22	html_tag_input	F36	html_event_onclick	F50	js_pseudo_protocol	F64	js_min_function_calls
F9	url_number_domain	F23	html_tag_textarea	F37	html_event_onerror	F51	js_dom_location	F65	js_string_max_length
F10	html_tag_script	F24	html_attr_action	F38	html_event_onfocus	F52	js_dom_document	F66	html_length
F11	html_tag_iframe	F25	html_attr_background	F39	html_event_onkeydown	F53	js_prop_cookie		
F12	html_tag_meta	F26	html_attr_classid	F40	html_event_onkeypress	F54	js_prop_referrer		

F13	html_tag_object	F27	html_attr_codebase	F41	html_event_onkeyup	F55	js_method_write

Table 2: The Various Main Subsets with Their Performance Measurement

No.F	Subsets of features	Accuracy
67	All 67 features	99.3553%
41	f0, f1, f2, f4, f8, f9, f10, f11, f12, f13, f14, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31, f33, f35, f37, f38, f42, f43, f44, f45, f48, f49, f51, f52, f55, f56, f58, f59, f62, f64, f65, f66	99.3553%
40	f0, f1, f2, f4, f8, f9, f10, f11, f12, f13, f14, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31, f33, f35, f38, f42, f43, f44, f45, f48, f49, f51, f52, f55, f56, f58, f59, f62, f64, f65, f66	99.3289%
38	f0, f1, f2, f4, f8, f9, f10, f11, f12, f13, f14, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31,f35, f38, f42, f44, f45, f48, f49, f51, f52, f55, f56, f58, f59, f62, f64, f65, f66	99.3433%
35	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31,f35, f38, f42, f45, f48, f49, f51, f52, f55, f56, f58, f59, f62, f64, f65, f66	99.3385%
34	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31,f35, f38, f42, f45, f48, f49, f51, f52, f55, f56, f58, f62, f64, f65, f66	99.3409%
33	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31,f35, f42, f45, f48, f49, f51, f52, f55, f56, f58, f62, f64, f65, f66	99.3337%
32	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f23, f24, f25, f28, f31, f42, f45, f48, f49, f51, f52, f55, f56, f58, f62, f64, f65, f66	99.3433%
30	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f24, f25, f28, f42, f54, f48, f49, f51, f52, f55, f56, f58, f62, f64, f65, f66	99.3505%
28	f0, f1, f2, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f24, f25, f28, f45, f48, f49, f51, f52, f55, f56, f58, f62, f65, f66	99.3216%
25	f0, f1, f4, f8, f9, f10, f11, f12, f15, f18, f19, f21, f22, f24, f25, f28, f45, f48, f51, f55, f56, f58, f62, f65, f66	99.3289%
20	f0, f1, f4, f8, f9, f12, f15, f18, f19, f24, f25, f28, f45, f48, f51, f56, f58, f62, f65, f66	99.3168%
15	f0, f1, f8, f9, f12, f15, f18, f19, f28, f48, f56, f58, f62, f65, f66	99.2783%
10	f0, f1, f8, f9, f15, f19, f56, f58, f62, f65	99.1725%

3.3. XGBoost model

XGBoost is an open-source optimized distributed gradient boosting system, designed to be highly efficient, accurate, flexible, and easy to use [40]. It has previously solved numerous data science problems and has already been utilized to win various machine-learning competitions [41]. In this paper, a scalable system is implemented using machine-learning algorithms under the Gradient Boosting framework, i.e., also known as Gradient Boosted Decision Trees (GBDT) and Gradient Boosted Machine (GBM), which are ensemble learning techniques. Ensemble learning introduces a systematic resolution to aggregate the prognostic power of multiple classifiers (learners) to obtain sustainable results instead of relying on a single classifier that results in establishing a unique model. The ensemble model includes multiple weak classifiers. Each weak classifier is a decision tree. The trees are constructed sequentially, such that each subsequent tree aims to scale back the errors of the previous tree. Every tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals. The tree can be defined as Formula (5).

$$\sum_{k=1}^{K} f_k \ , f_k \in \mathcal{F}, k = 1,2 \dots K \qquad (5)$$

Where *K* is the number of decision trees in the model and f_k is prediction from the decision tree that belongs to the space of the regression tree. Having all decision tree, we can make a prediction on dataset input-output pairs $(\overrightarrow{x_i}, y_i)$, where y_i denotes to one-dimensional output $y_i \in \{0,1\}$, where 0= benign and 1= malicious, on *n*-dimensional input vector $\vec{x} = \{x_1, x_2, \dots x_i\}$ by sum all prediction results of each tree as Formula (6).

$$\widehat{y_i} = \sum_{k=1}^{K} f_k\,(x_i) \qquad (6)$$

Where the x_i is the feature vector for i^{th} data point. Similarly, we trained the model at each iteration, where one tree is added to the model until it reaches the maximum number of predefined trees. Therefore, the prediction at the t^{th} time can be defined as Formula (7).

$$\widehat{y_i}^{(t)} = \sum_{k=1}^{t} f_k\,(x_i) \qquad (7)$$

The training model has two main function including loss function L and the regularization function Ω. L controls the productive power, and Ω controls the simplicity. Hence, the objective function can be defined as Formula (8).

$$Obj = L + \Omega \tag{8}$$

The loss function L in our model is the logistic regression and is calculated as Formula (9).

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{9}$$

Where $\hat{y}_i$ refers to the computed output of the XGBXSS model on input $\overrightarrow{x_i}$, and y_i indicates the actual value for input-output pair $\overrightarrow{x_i}, y_i$. Further, the regularization function Ω is given as Formula (10).

$$\Omega = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \tag{10}$$

Where T is the number of leaves on the tree, and the w_j^2 is the rank/ score on the j^{th} leaf. The gamma γ and λ are parameters to control the degree of regularization. With the combination of Formula (9) and (10), the objective function of Formula (4) can be re-defined as Formula (11).

$$Obj^t = \hat{y}_i^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$\hat{y}_i^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{i=1}^{t} \Omega(f_i) \tag{11}$$

Where $\hat{y}_i^{(t)}$ denote to the model training at round t, which is composed of the prediction of t^{th}steps to the previous steps $\hat{y}_i^{(t-1)}$ and current prediction $f_t(x_i)$, this is called additive training (boosting) technique.

Given the objective $Obj(y, \hat{y})$, the XGBXSS framework used gradient descent, which is an iterative technique to optimize f_t function at each epoch/iteration. Accordingly, the framework improves the productive $\hat{y}$ along the direction of the gradient to minimize the objective. The improvement can be performed by considering both the first and second order gradient as $\partial_{\hat{y}^{(t)}} Obj^t$ and $\partial^2_{\hat{y}^{(t)}} Obj^t$to converge faster as Formula (12).

$$Obj^{(t)} = \sum_{i=1}^{N}[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \sum_{i=1}^{t} \Omega(f_i), \quad \begin{cases} g_i = \partial_{\hat{y}^{(t-1)}} Obj^{t-1} \\ h_i = \partial^2_{\hat{y}^{(t-1)}} Obj^{t-1} \end{cases} \tag{12}$$

Where $g_i$is gradient and h_i is the second order gradient of loss function. We eliminated the constant terms to get the new objective at t^{th} step, which is defined as Formula (13).

$$Obj^{(t)} = \sum_{i=1}^{n} g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) + \Omega(f_t) \tag{13}$$

Where our goal is to find f_t and to optimize it. However, in the decision tree, each data point flows to one of the leaves flowing in the node's direction. Hence, we need to define the complexity of the tree as Formula (14).

$$\begin{cases} f_t(x_i) = w_{q(x)} \\ I_j = \{i \,|\, q(x_i) = j\} \end{cases} \tag{14}$$

Where the $q(x)$ is the indexing function that is assigned to each x (data point) to $q(x)^{th}$ leaf and the w is the corresponding score on $q(x)^{th}$ to data point x. Moreover, I_j is the index set that contains the indices of the data points that are assigned to the j^{th} leaf. Hence, we can regroup the objective function at Formula (13) considering leaf by leaf with an index set which points to the data points that are assigned to the j^{th} leaf, rather than considering the sum of all functions by the data points n separately as Formula(15).

$$Obj^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T \tag{15}$$

Thus, we get the sum on T independent quadratic function, which the best weight/score w in each leaf j and the resulting objective value $Obj^{(t)}$ can be obtained by Formula(16).

$$\begin{cases} w_j^* = -\dfrac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \\ Obj^{(t)} = -\dfrac{1}{2}\sum_{j=1}^{T} \dfrac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \end{cases} \quad (16)$$

The leaf weight/score w_j is related to g_i , h_i and λ, that pointer to first and second orders gradient of loss function along with regularization parameter. To find the tree's good structure, the algorithm considers the best splitting point greedily based on its objectives, leading to optimizing it. Moreover, at each time the algorithm does the split, a leaf changed to an internal node. Let I is the set of indexes of data points assigned to this node, and I_L, I_R are sets of indexes of data points assign to two new leaves (right and left). Therefore, from equation (16), the best value of objective $Obj^{(t)}\ in\ j^{th}$ leaf is achieved. Furthermore, the gain could be calculated by some of right and left of new leaves minus the previous leaf as Formula (17).

$$\text{gain} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \quad (17)$$

If the gain is positive, the algorithm keeps these new leaves, and if the gain is negative, the algorithm will delete these fresh leaves and trains another new leaf. This process is called pruning, which aims to reduce the complexity of the tree.

In the next section, we will discuss the implementation environment and design of the experiments includes the initial structure established and the baseline experiment estimator. Finally, the parameters of the detection framework are extremely optimized to form the final model.

4. Experimental Design and Optimization

4.1. The dataset

The dataset consists of 138,569 samples, where 100,000 are benign, and 38,569 are malicious samples with 30-dimensional features that were selected. To the development of a robust and accurate estimation model and provide an unbiased sense of proposed model efficiency, the dataset has been split randomly and separately into three parts with a partition ratio of 60%: 20%: 20% for training, validation, and testing sets. The training set includes 83,142 samples labeled as [0: Benign, 1: Malicious], the validation set contains 27,714 samples, and the holdout set consists of 27,713 samples, which is used only to estimate the effectiveness of the final and fully trained model. Table 3 shows the detailed partitions of the dataset.

Table 3: Dataset Subdivisions

Name	Benign	Malicious	Total
Training dataset	60,155	22,987	83,142
Validation dataset	19,871	7,843	27,714
Test dataset	19,974	7,739	27,713
Total dataset	100,000	38,569	138,569

4.2. Performance evaluation metrics

In this research, an accuracy, precision, detection rate (DR)/ true positive (TP), misclassification rate (Error Rate), false positive (FP), false (FN) negative, F-score, and AUC curve are selected to evaluate the performance of the proposed framework which is based on confusion matrix [51]. In a confusion matrix, true negative (TN) represents whether the benign case is correctly classified as benign or not. FP or type I error represents a benign case which is wrongly labeled as an XSS attack. FN or type II error represents an XSS attack that is wrongly identified as benign. TP refers that an XSS is successfully identified as an attack. The detailed derivation of the selected performance metrics appears in Formulas (18) to (25).

$$Precision = \frac{TP}{(TP\ +\ FP)} \quad (18)$$

$$Recall\ or\ Detection\ _{Rate}(DR)\ = \frac{TP}{(TP\ +\ FN)} \quad (19)$$

$$FP_{Rate}\ = \frac{FP}{(TN\ +\ FP)} \quad (20)$$

$$FN_{Rate} = \frac{FN}{(TP + FN)} \quad (21)$$

$$F - Score = 2\left(\frac{(Recall \times Precision)}{(Recall + Precision)}\right) \quad (22)$$

$$Accuracy_{overall} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (23)$$

$$Misclassification_{Rate} = \frac{(FP + FN)}{(TP + TN + FP + FN)} \quad (24)$$

$$Area\ Under\ the\ Curve\ (AUC) = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \quad (25)$$

4.3. Model Parameter Optimization

XGBoost is considered one of the performant models for the classification of tabular data that utilizes a gradient boosting algorithm with several advanced functions. The model is dynamically designed to feed on the tabular dataset coming from the feature extraction process. However, constructing a model using XGBoost is straightforward, but improving the model using XGBoost is difficult. Furthermore, establishing an efficient classification model using the XGBoost algorithm requires parameter tuning [40]. Therefore, we subdivided the experimentations to tune these parameters using two steps. (i) Initially, we started with the default configuration using k-fold cross-validation approach provided in scit-learn as a baseline estimate of the XGBXSS detection framework on the training dataset to establish the basis of the estimation of our detection framework.

This technique comprises randomly partitioning the training dataset into a set of k-folds (i.e., k=10) of equal subsets. The model is trained on the k-1 fold with one held back fold to be used for testing. Furthermore, it is reiterated so that each k-fold of the dataset is allowed to be held back as a test set [42]. Finally, the algorithm ends up with k-different performance rates that are averaged to make a single estimation utilizing a mean and standard deviation, as shown in Table 4. The default parameters for XGBoost in the Python API Reference can be summarized as learning rate = 0.3, n_estimators = 100, max_depth = 3, subsample = 1 and colsample_bytree=1. The entire experimentation was carried out using LinuxMint-19-tara OS, Kernel 4.15.0-42-generic, 16 GB RAM, Intel© Xeon© CPU E-5-2620 v3 @ 2.40GHz, GPU NIVIDA (Quadro K220). The XGboost version (0.82), and the Python framework version is 3.6.7.

From the results of Table 4, we can see the effectiveness of the model performance, where it achieved the mean accuracy of 0.9918 and the variance of 0.009. Therefore, the framework falls into the low-bias low-variance category of bias-variance tradeoff categories. (ii) In the second stage, to further improve the model, Grid-search is employed as a hyperparameter optimization technique with a 10-fold cross-validation approach [43]. The Grid-search is an efficient approach, which is widely used for parameter tuning that methodically builds and evaluate a model for each combination of the algorithm with parameters specified in a grid. We provided a dictionary of hyper-parameters to evaluate in the param_grid argument as a map of the model parameter name and an array of values to examine. These parameters are the number of trees (n_estimators with values from 100 to 800 with an increment of 100 at each iteration), Size of trees (max_depth within values of 3,5,7,9, and 11), learning rate also called shrinkage or eta within values of (0.0001, 0.001, 0.01, 0.1, 0.2, 0.3 and 0.4), row subsampling (subsample) within values from 0.1 to 1.0, and column subsampling (colsample_bytree within values between 0.1 and 1.0 incremented by 0.1). Moreover, based on our interest in probability estimates, the log-loss is the best as the calibration statistic function to be minimized for training the detector. It measures the relative uncertainty between the classes that our framework predicts and the actual labels of the true classes, where it penalizes the divergence between actual and estimated probabilities. Hence, all tuning parameters are applied along with the log loss function as objective to minimize loss of the model. Furthermore, we also studied the effect of the number of trees and the size of trees because of the relationship between n_estimators in the model and every tree's max_depth. The more deep-rooted trees would lead to fewer trees, whereas the inverse needs a higher number of trees with less complicated trees (e.g., decision stumps) to realize similar results. The relationship between each series of max depth values for given n_estimator is shown in Fig. 2, where the best number of trees is 400 with a depth value of 3, having a log loss of 0.013599. Also, gradient boosted decision trees are featuring quickly to learn and overfit training data because it involves building and joining trees to the model sequentially. However, one efficient way to slow down learning in the gradient boosting paradigm is the use of weighting. The weighting is also known as an eta or the shrinkage factor or the learning rate. We also studied the effect of the learning rate versus the loss-function, as shown in Fig. 3. However, the best step size shrinkage utilized during the update to prevent over fitting was found to be 0.1 with a max_depth of 5 and number of trees (n_estimators) equals to 400, with a performance log loss of 0.012783. Also, the learning rate shrank the feature weights to make the boosting process more conservative, as shown in Fig. 4.

We also studied the effect of row subsampling and column subsampling on the XGBXSS framework. The former involves picking the sample of training dataset randomly without replacement, while the latter consists of using

randomly created samples of columns (or features) before constructing a tree using the boosted framework. We found that the best mean performance was achieved by tuning the subsample to 20% with a performance log loss of 0.020543, which is even better than 020913 that was seen while configuring the per-tree row sampling to 100% as shown in Fig. 5. It is attention-grabbing to notice that every subsample values' mean performance scores are significantly higher than the scores achieved without subsampling. Whereas in the case of column sampling, the best performance scores are with colsample_bytree =1.0 and colsample_bylevel =1.0, as shown in Fig. 6 and 7, respectively, reveals that column sampling on this problem does not add any value.
Furthermore, the next section discusses the finalization steps, training, evaluation, and testing results on a new dataset of the proposed framework.

Table 4: XGBXSS Model Bias-Variance Tradeoff with 10-Fold Cross-Validation on Default Configuration.

K-Fold =10	1	2	3	4	5	6	7	8	9	10	Mean Accuracy	Variance
Default Config	99.84	99.79	99.81	99.78	99.70	99.78	99.71	97.86	97.85	97.69	99.18	0.91

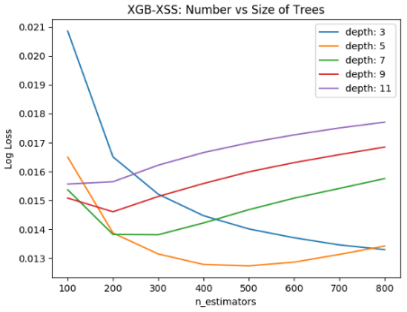


Fig. 2 - No. of Trees Vs Size of Trees

Fig. 3 - Learning Rate Vs Log Loss

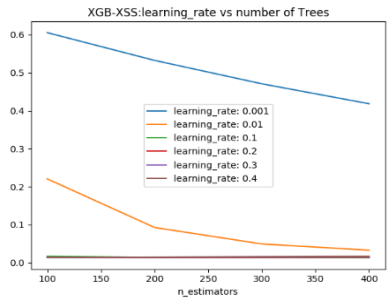


Fig. 4 - Learning Rate Vs No. of Trees

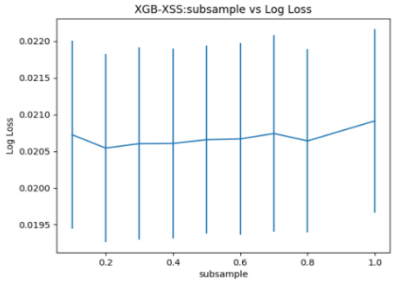


Fig. 5 - Row subsampling Vs Log Loss

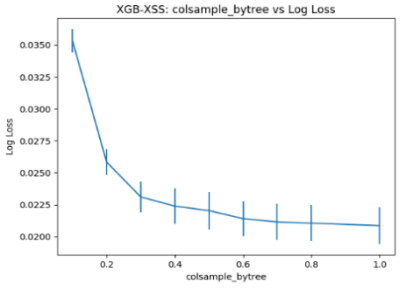


Fig. 6 - Column subsampling by Tree Vs Log Loss

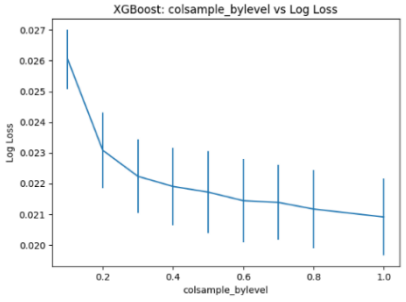


Fig. 7 - Column subsampling by Level Vs Log Loss

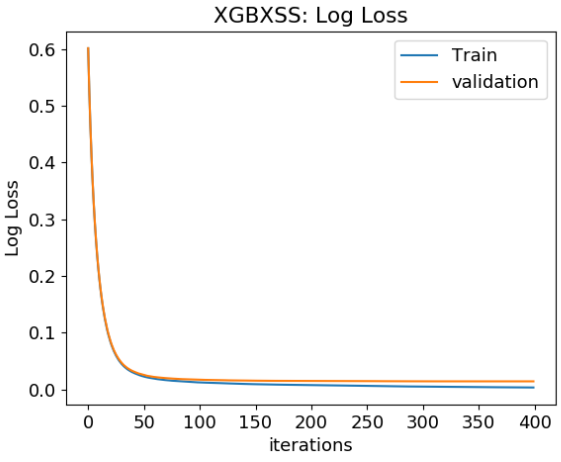


Fig. 8 - XGBXSS Log Loss VS Iterations

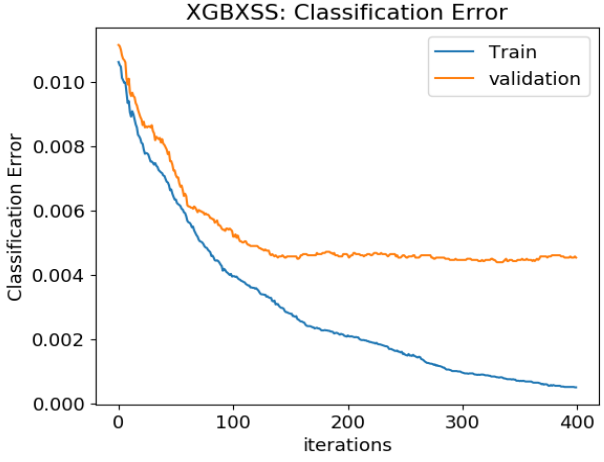


Fig. 9 - XGBXSS Classification Error VS Iterations

5. Results and Discussion

5.1. Finalization of the detection framework

The parameters of the proposed framework are calibrated to achieve better performance results, where parameters are configured to achieve optimal results that involve parameter settings, as shown in Table 5, whereas the other parameters are kept rigid to default. Later, by carefully and rigorously observing the experimentation results of the XBGXSS framework, the performance was monitored on the validation dataset to verify the number of calibrated trees and adopt an early stopping technique once performance on the validation dataset begins to deteriorate. Since the number of iterations and the window of the number of iterations/epochs over which no improvement is observed must be set and predefined, we specified this window size to 10% of the total number of training iterations/epochs to be 40 within the log loss as an optimization function. Fig. 8 shows the log loss vs. iterations/epochs on the training and validation dataset, and Fig. 9 shows the classification error vs. iterations/epochs on the training and validation datasets. Besides, to show the general idea and to take an in-depth look at the model, Fig. 10 shows the first tree (No.1) in the framework, and Fig. 11 shows the last tree (No. 400) in the framework.

Table 5: Parameters of XGBXSS framework

Parameters	Values	Parameters	Values
n_estimators	400	max_depth	5
learning_rate(eta)	0.1	gamma	0
Subsample	0.20	colsample_bytree	1.0
Objective	binary: logistic	eval_metric	logloss
early stopping rounds	40	colsample_bylevel	1.0

After completing the training of XGBXSS, the framework was further tested on a new large-scale real-world dataset; this held-out testing dataset is composed of 27,713 samples, of which 7,739 are malicious and 19,974 benign samples.

5.2. Results

As a result of applying proposed framework on the held-out testing dataset, we achieved accuracy, precision, detection probabilities, FP rate, FN rate, and AUC scores of 99.59%, 99.50 %, 99.02%, 0.20%, 0.98%, and 99.41% respectively for the XSS class. Furthermore, the macro averages and weighted averages of both classes are computed to obtain a global value of quality for the complete classification of our framework. Table 6 presents the comprehensive report generated, which is based on the confusion matrix for the testing of XGBXSS framework skills and achievement on both of the classes, i.e., XSS and Non-XSS. Fig.12 shows the result of the confusion matrix with a complete statistical view. We can observe that the XGBXSS framework successfully achieved a state of the art results. It also steadies significantly of achieving high precision and high recall simultaneously, which is evident through the F-score. Besides, FNR and FPR are also near to zero.

Furthermore, the detection framework's performance is also tested with the ROC curve (receiver operating characteristic) to clarify our XGBXSS attack detection framework's discriminative robustness. The ROC curve is one of the key evaluation metrics for any detector's skills and visualizes its accomplishment with a comprehensive summary of the model performance quality. It is evident from the test results on the held-out dataset that the detector is beneficial in detecting XSS-based attacks, with an area under the curve score of 99.41% as shown in Fig. 13 (a) and Fig.13 (b) for zooming at the top left, which implies that it has a powerful capability to detect XSS-based attacks effectively and efficiently.

Table 6: Experimental Results of XGBXSS on Test Dataset

Class	Precision	Detection Rate	F-score	Accuracy overall	FN Rate	FP Rate	Misclassification rate (Error Rate)	AUC
XSS	0.9950	0.9902	0.9926	0.9959	0.0098	0.0020	0.0041	0.9941
Non-XSS	0.9962	0.9980	0.9971					
weighted avg (micro avg)	0.9958	0.9959	0.9958					
macro avg	0.9956	0.9941	0.9948					

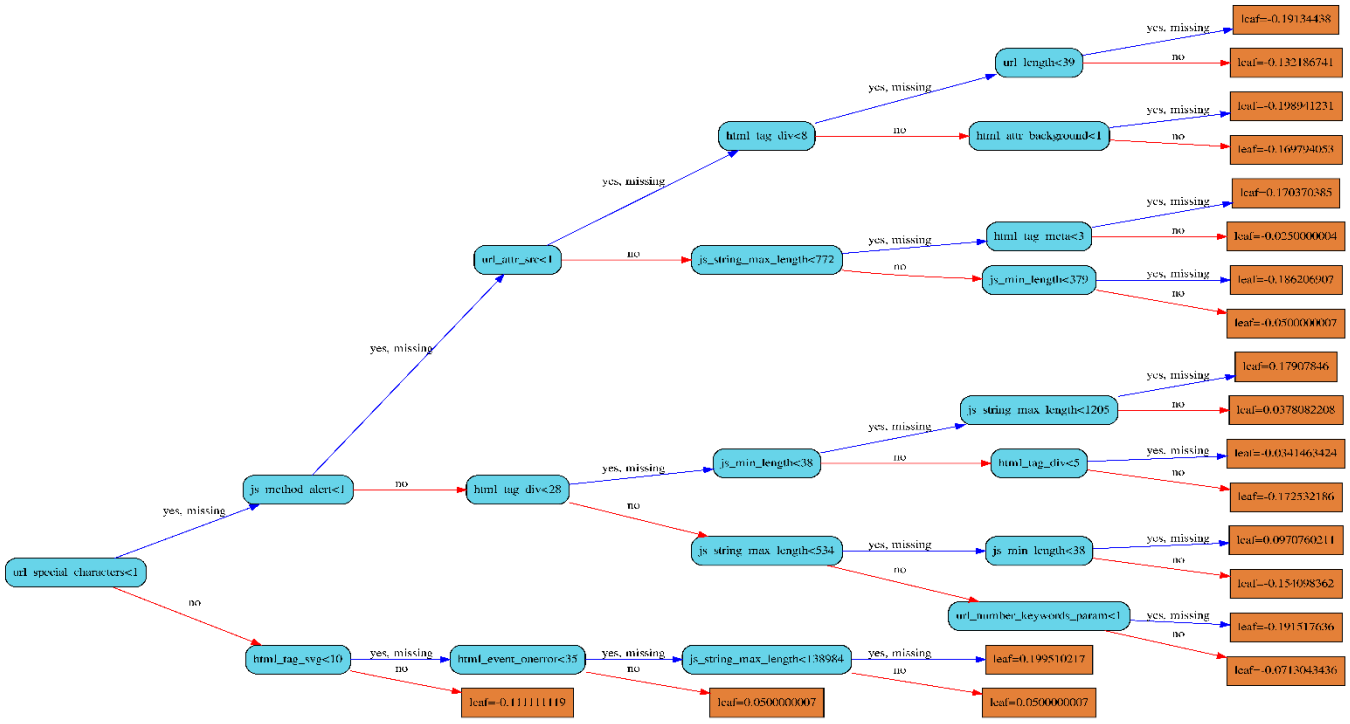


Fig. 10 - The First Decision Tree in the XGBXSS (No. 1)

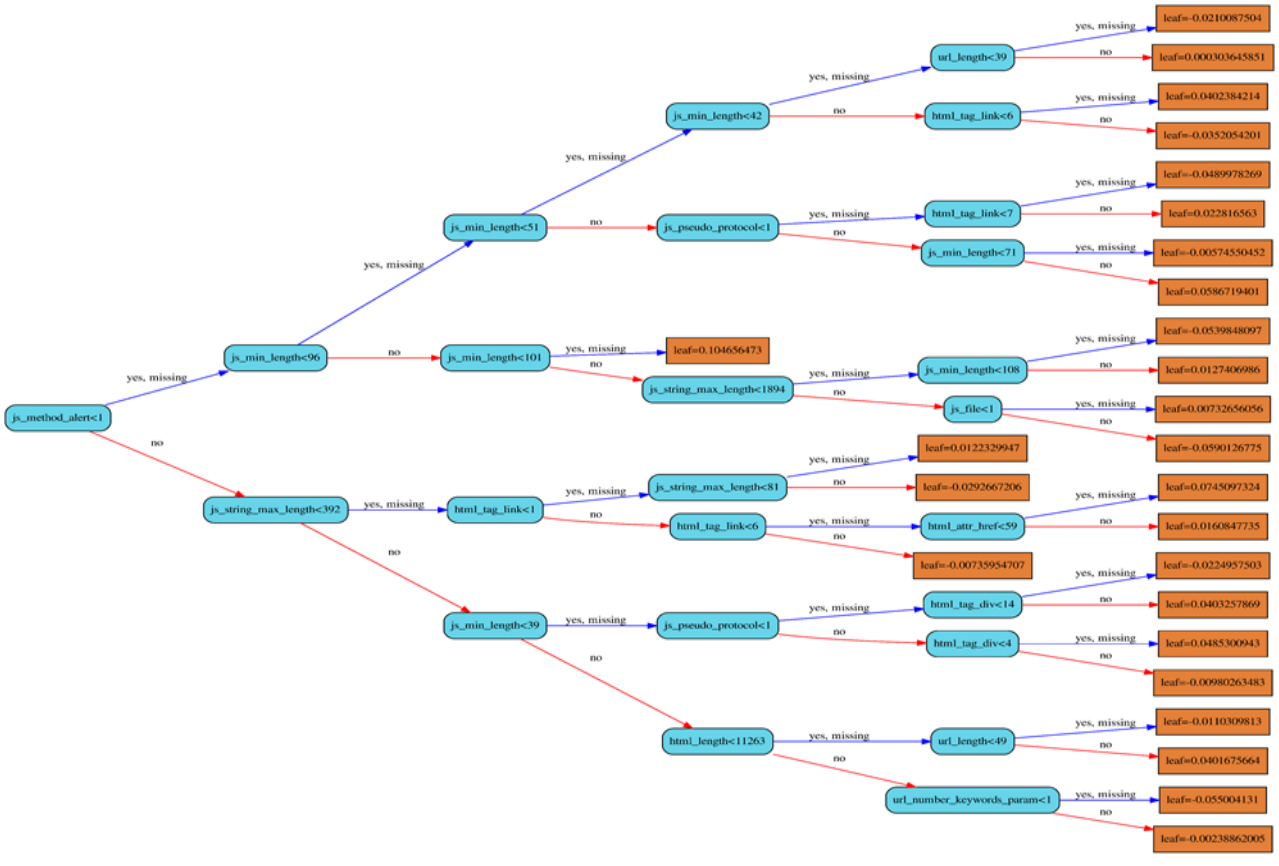


Fig. 11 - The Last Decision Tree in the XGBXSS (No. 400)

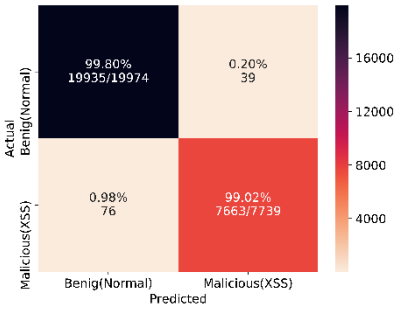


Fig. 12 - Confusion Matrix on Test Dataset

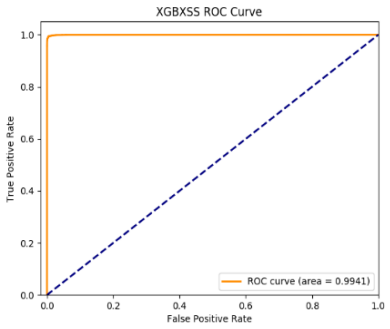


Fig. 13 (a) - XGBXSS Roc Curve

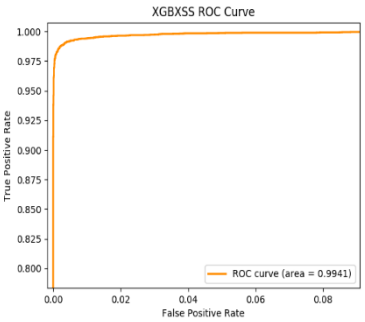


Fig. 13 (b) - XGBXSS Roc Curve zooming on the top left

5.3. Discussion

This section discusses the results that have been achieved by the XGBXSS detection framework, comparing it with widely used ML techniques on the same dataset. Further, XGBXSS is also compared with previous detectors. Moreover, this section also explains the significance of the proposed framework, including the reasons for achieving state-of-the-art results.

5.3.1. Comparison with other detectors techniques

To highlight the advantages of proposed detection, five different experiments are executed using well-known machine learning techniques. That is, SVM with Radial Basis Function (RBF) kernel having c value =1.0, KNN with n_neighbors =5, bagging method of random forest (RF) classifier, Adaptive Boosting (AdaBoost), and neural network (multi-layer perceptrons MLP). The parameters of all the above ML methods are tuned with GridSearch using 10-fold cross-validation. After the optimization stage, all of these algorithms were applied to the same dataset, which we extracted with the proposed methodology. The training was performed on the training dataset, the validation was performed on the validation dataset, and finally, the test was employed on the holdout testing dataset. Although the results are promising with all ML classifiers, we can see that the SVM, KNN, and RF classifiers are selective and do not classify many samples as malicious, whereas all of the samples they classified maliciously are genuinely malicious. They also missed some of the actual malicious samples, as they have a relatively low recall, but very high precision; therefore, they are relatively selective compared to XGBXSS. This fact can also be observed through a higher FN rate as compared to the FP rate in Table 7(b). With the same logic regarding AdaBoost and MLP detectors, they are considered less selective detectors and outperformed than SVM, KNN, and RF. This fact can be seen with detection rate and F-score measurements. However, their detection rates are still a bit far from 100%. Therefore, they also do not classify many samples as malicious, and thus, misses some of the actual malicious samples because it has a slightly low detection rate/recall. But still, they have very high precision and are relatively selective as compare to XGBXSS.

In contrast, XGBXSS, as compare to all detectors, shows more efficiency and advantages in different aspects, where it achieved a great optimal performance with all measurements as details shown in Table 7(a, b). The proposed detection framework is efficient and reliable, which is evident by observing the notable perfection and better performance metrics scores of various experiments considering both classes. Where it produced very high recall-high precision simultaneously that is also evident through F-score. F-score is a significant measure widely used to convey these measurements into one coherent metric. Furthermore, a very low FP and FN rates are near to zero. Additionally, the argument that emphasizes the XGBXSS framework's quality can be seen in the observed score of the area under the ROC curve, i.e., 99.41%. Finally, our proposed framework's performance speed performed existing approaches, where it finished training and testing on a huge dataset in record time compared to other detectors, by 0.22 minutes for the training and 0.01 minute for the testing task.

On the other hand, it should be noted that the recall/detection rates of all the algorithms we applied and compared are not diminutively at all. On the contrary, they are still excellent compared to the results reported in previous studies, which we will be discussing in the next part of our discussion. Our discussion of these results are from the perspective to what extent the detection rates of these attacks are far from approaching unit, where the results are 100% or closer to it; however, the absolute idealism (i.e., 100%) could be challenging to achieve but not impossible using AI techniques.

Table 7 (a): Result Comparison XGBXSS with Other Detectors on the Same Dataset

Detectors	XSS			Non-XSS		
	Precision	Detection Rate	F-score	Precision	Detection Rate	F-score
XGBXSS	**0.9950**	**0.9902**	**0.9926**	**0.9962**	**0.9980**	**0.9971**
SVM	0.9986	0.9273	0.9616	0.9726	0.9995	0.9859

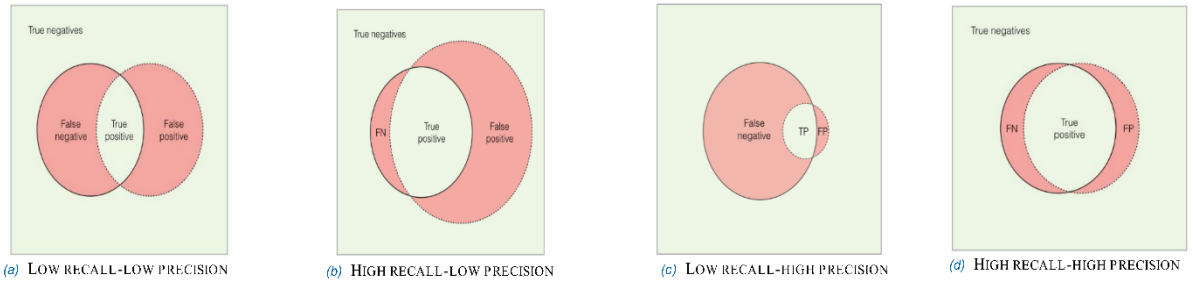
K-NN	0.9902	0.9780	0.9762	0.9857	0.9963	0.9910
RF	0.9962	0.9780	0.9870	0.9915	0.9985	0.9950
AdaBoost	0.9911	0.9829	0.9870	0.9934	0.9966	0.9950
MLP	0.9921	0.9835	0.9877	0.9924	0.9969	0.9953

Table 7(b): Result Comparison XGBXSS with Other Detectors

Detectors	Accuracy overall	FN Rate	FP Rate	Misclassification rat (Error Rate)	AUC	Training–Time(m)	Testing–Time(m)
XGBXSS	**0.9959**	**0.0098**	**0.0020**	**0.0041**	**0.9941**	**0.22**	**0.01**
SVM	0.9793	0.0727	0.0005	0.0207	0.9634	3.94	0.15
K-NN	0.9869	0.0373	0.0037	0.0131	0.9795	1.19	2.58
RF	0.9928	0.0220	0.0015	0.0072	0.9883	0.23	0.01
AdaBoost	0.9928	0.0171	0.0034	0.0072	0.9898	0.30	0.03
MLP	0.9932	0.0165	0.0031	0.0068	0.9902	6.932	2.165

5.3.2. Comparison with previous detectors

To further assess the advantages of the proposed XGBXSS framework, it is compared with the reported quantitative measures between different literature studies. Initially, XGBXSS is compared with the detectors proposed by [25][26][30][13], where XSSed is used to observe maliciousness in the data. Besides, the proposed detection framework is also compared with the detectors proposed by [24][27][31], where the XSS considerably utilizes malignant JS as a portion of the attack. All of those proposed models have been debated in the literature work section. The proposed scheme's comparative results with other detectors are presented in Table 8, which can also be seen in Fig.15. The proposed framework demonstrates the capability and efficiency to compete with other detectors with a high level of performance on various measurements. It has very high recall (detection)-high precision rate. Besides, it produced very low FP and FN rates, which are nearer to zero. The other argument that emphasizes the quality of our model can be seen in the area under the ROC curve (AUC) parameter, where it achieved a score of 99.41%. For the establishment of additional clarification, Fig.14, is presented to bring four different cases as mentioned by [44], where these four cases can represent the detection models that use artificial intelligence techniques.



(a) LOW RECALL-LOW PRECISION (b) HIGH RECALL-LOW PRECISION (c) LOW RECALL-HIGH PRECISION (d) HIGH RECALL-HIGH PRECISION

Fig. 14 - The Four Cases of Models Performance

The four cases are as follows, Fig.14 (a) represents low recall-low precision, Fig.14 (b) represents high recall-low precision, Fig.14(c) represents low recall-high precision, or Fig.14 (d) represents high recall-high precision (best case). In the case of Fig.14 (a), it leads to a high FP and FN, (Likarish et al., 2009; Wang et al., 2014; Rathore et al., 2017; Wang et al., 2016) as an example, with varying ratios for each of them. While in the case of Fig.14 (b), it leads to a high FP rate and low FN rate. In the case of Fig.14 (c), it leads to a high FN and low FP as in(Fang et al., 2018, Wang R et al.,2017, and Mokbal et al.,2018), with varying ratios for each of them. Moreover, in the case of Fig.14 (d), it is an ideal case, which leads to low FP and FN rates as in the XGBXSS detection framework. For this reason, most of the published papers do not correctly list the FN rate.

Table 8: Result Comparison XGBXSS with Other Previous Detectors

Detectors	Accuracy overall	Precision	Recall/Detection rate	F-score	FN Rate	FP Rate	Misclassification rate	AUC-ROC
Likarish et al., 2009 [24]	N/A	0.9200	0.7420	0.7640	N/A	0.1305	N/A	0.8058
Wang et al., 2014 [25]	N/A	0.9410	0.9390	0.9390	N/A	0.0420	N/A	0.9485
Wang et al., 2016 [27]	0.9482	0.9490	0.9480	0.9480	N/A	0.0414	0.0518	0.9533

Rathore et al., 2017 [26]	0.9720	0.9770	0.9710	0.9740	N/A	0.0870	0.0280	0.9442
Wang et al.,2017 [31]	N/A	0.9520	0.8820	0.9160	N/A	N/A	N/A	0.9479
Fang et al., 2018 [30]	N/A	0.9950	0.9790	0.9870	N/A	0.0190	N/A	0.9800
Mokbal et al., 2019 [13]	0. 9932	0.9921	0.9835	0.9877	N/A	0.0031	0.0068	0.9902
XGBXSS(This Paper)	**0.9959**	**0.9950**	**0.9902**	**0.9926**	**0.0098**	**0.0020**	**0.0041**	**0.9941**

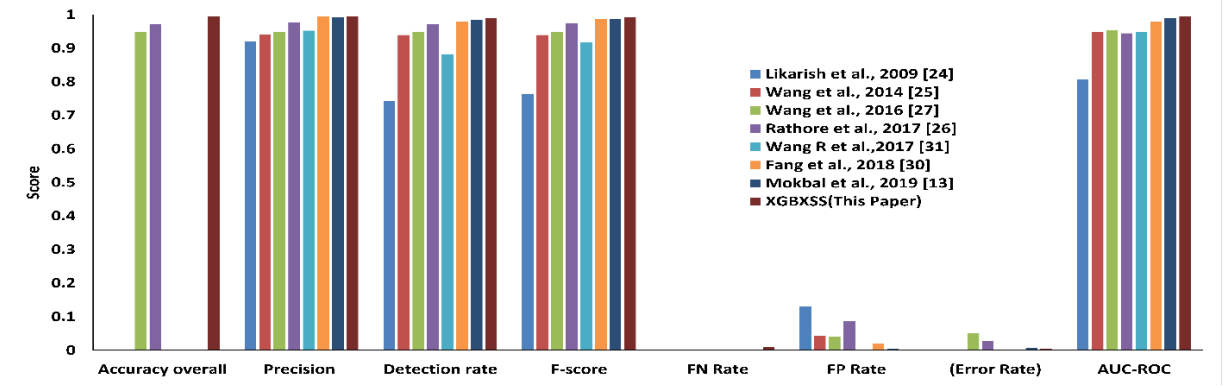


Fig. 15 - Result Comparison XGBXSS with Other Previous Detectors

6. Conclusions

This research proposes to use an XGBXSS detection framework for the detection of web-based XSS attacks. The detection framework has been proved efficient with the capability of achieving remarkable accuracy and detection rate with minimal False Positive and False Negative rates, i.e., almost equivalent to zero. The detection framework adopted a large dataset for the training and testing perspective with a proposed features extraction and selection technique, and ensemble learning technique for the detection task. Numerous analyses have been performed to test the proposed framework at various stages. The experimental results establish the efficiency and idealism of the XGBXSS framework with significant perfection and harmony of performance on multiple measurements of both classes compared to five well-knowing ML algorithms. The proposed detection framework can reduce the dataset dimensions up to 30 features while maintaining high-performance measures. More important is that it can address the FN and FP simultaneously, and this is what failed to achieve by most of the proposed models in the literature.

Furthermore, one of our proposed framework's discriminant features besides being robust with a high-precision, high-detection rate, and low complexity, is that it can easily be deployed as a complete system. The page could quickly be crawled by supplying the link in the header of the HTTP response for downloading the page (i.e., cashing). The feature extractor will perform full parsing to it produce the digital data that will feed the XGBXSS framework to establish the prediction process. Finally, the XGBXSS framework will perform a prediction to determine whether there was an attack or not. Hence, the prediction can reflect in the form of a message to users to accept or reject the subsequent treatment of the page if it is encountered with an XSS based attack. In our future work, we will try to improve the XGBXSS framework further to be able to detect other types of attacks targeting web applications.

References

[1] Andreeva O, Gordeychik S, Gritsai G, Kochetova O, Potseluevskaya E, Sidorov SI, et al. Industrial Control Systems Vulnerabilities Statistics. 2016.

[2] Nithya V, Lakshmana Pandian S, Malarvizhi C. A survey on detection and prevention of cross-site scripting attack. Int J Secur Its Appl 2015;9:139–52. https://doi.org/10.14257/ijseia.2015.9.3.14.

[3] Sarmah U, Bhattacharyya DK, Kalita JK. A survey of detection methods for XSS attacks. J Netw Comput Appl 2018;118:113–43. https://doi.org/10.1016/j.jnca.2018.06.004.

[4] Zhou Y, Wang P. An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. Comput Secur 2019;82:261–9. https://doi.org/10.1016/j.cose.2018.12.016.

[5] Rodríguez GE, Torres JG, Flores P, Benavides DE. Cross-site scripting (XSS) attacks and mitigation: A survey. Comput Networks 2019;166. https://doi.org/10.1016/j.comnet.2019.106960.

[6] Cross-Site Scripting (XSS) Makes Nearly 40% of All Cyber Attacks in 2019 - PreciseSecurity.com n.d. https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/ (accessed February 29, 2020).

[7] NIST.gov. National Vulnerability Database (NVD), Vulnerabilities n.d. https://nvd.nist.gov/vuln.

[8] Adams H, Hoole AM, Kamani S, Lemos R, Martin L, Mello J, et al. 2018 Application Security Research Update. 2018.

[9] Akamai. State of the Internet: Security | Credential Stuffing Attacks Report (Volume 4, Issue 4)| Akamai 2018;4.

[10] Owasp. OWASP top 10 - 2017 The Ten Most Critical Web Application Security Risks. OwaspOrg 2017;1:1–24. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf.

[11] WhiteHat. 2018 Application Security Statistics Report. 2018.

[12] Murphree J. Machine learning anomaly detection in large systems. AUTOTESTCON (Proceedings) 2016;2016-Octob:1–9. https://doi.org/10.1109/AUTEST.2016.7589589.

[13] Mokbal FMM, Dan W, Imran A, Jiuchuan L, Akhtar F, Xiaoxi W. MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique. IEEE Access 2019;7:100567–80. https://doi.org/10.1109/access.2019.2927417.

[14] Pajila PJB, Julie EG. Detection of DOM-Based XSS Attack on Web Application. vol. 33. Springer International Publishing; 2020. https://doi.org/10.1007/978-3-030-28364-3.

[15] Gupta S, Gupta BB. XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code. Arab J Sci Eng 2016;41:897–920. https://doi.org/10.1007/s13369-015-1891-7.

[16] Mohammadi M, Chu B, Lipford HR. Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing. 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur., IEEE; 2017, p. 364–73. https://doi.org/10.1109/QRS.2017.46.

[17] Guo X, Jin S, Zhang Y. XSS Vulnerability Detection Using Optimized Attack Vector Repertory. 2015 Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov., IEEE; 2015, p. 29–36. https://doi.org/10.1109/CyberC.2015.50.

[18] K.Patil D, R. Patil K. Client-side Automated Sanitizer for Cross-Site Scripting Vulnerabilities. Int J Comput Appl 2015;121:1–8. https://doi.org/10.5120/21653-5063.

[19] Rafique S, Humayun M, Gul Z, Abbas A, Javed H. Systematic Review of Web Application Security Vulnerabilities Detection Methods. J Comput Commun 2015;03:28–40. https://doi.org/10.4236/jcc.2015.39004.

[20] Deepa G, Thilagam PS. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Inf Softw Technol 2016;74:160–80. https://doi.org/10.1016/j.infsof.2016.02.005.

[21] Román Muñoz F, Sabido Cortes II, García Villalba LJ. Enlargement of vulnerable web applications for testing. J Supercomput 2018;74:6598–617. https://doi.org/10.1007/s11227-017-1981-2.

[22] Lekies S, Kotowicz K, Groß S, Vela Nava EA, Johns M. Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets. Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS '17, New York, USA: ACM Press; 2017, p. 1709–23. https://doi.org/10.1145/3133956.3134091.

[23] Prokhorenko V, Choo KKR, Ashman H. Web application protection techniques: A taxonomy. J Netw Comput Appl 2016;60:95–112. https://doi.org/10.1016/j.jnca.2015.11.017.

[24] Likarish P, Jung E, Jo I. Obfuscated malicious javascript detection using classification techniques. 2009 4th Int. Conf. Malicious Unwanted Softw., IEEE; 2009, p. 47–54. https://doi.org/10.1109/MALWARE.2009.5403020.

[25] Wang R, Jia X, Li Q, Zhang S. Machine Learning Based Cross-Site Scripting Detection in Online Social Network. 2014 IEEE Intl Conf High Perform. Comput. Commun. 2014 IEEE 6th Intl Symp Cybersp. Saf. Secur. 2014 IEEE 11th Intl Conf Embed. Softw. Syst, IEEE; 2014, p. 823–6. https://doi.org/10.1109/HPCC.2014.137.

[26] Rathore S, Sharma PK, Park JH. XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs. J Inf Process Syst 2017;13:1014–28. https://doi.org/10.3745/JIPS.03.0079.

[27] Wang Y, Cai W, Wei P. A deep learning approach for detecting malicious JavaScript code. Secur Commun Networks 2016;9:1520–34. https://doi.org/10.1002/sec.1441.

[28] Pan Y, Sun F, White J, Schmidt DC, Staples J, Krause L. Detecting Web Attacks with End-to-End Deep Learning. Acm 2019:1–14. https://doi.org/https://doi.org/0000001.0000001.

[29] Stokes JW, Agrawal R, McDonald G. Neural Classification of Malicious Scripts: A study with JavaScript and VBScript 2018:1–20.

[30] Fang Y, Li Y, Liu L, Huang C. DeepXSS: Cross Site Scripting Detection Based on Deep Learning. Proc. 2018 Int. Conf. Comput. Artif. Intell. - ICCAI 2018, New York, New York, USA: ACM Press; 2018, p. 47–51. https://doi.org/10.1145/3194452.3194469.

[31] Wang R, Zhu Y, Tan J, Zhou B. Detection of malicious web pages based on hybrid analysis. J Inf Secur Appl 2017;35:68–74. https://doi.org/10.1016/j.jisa.2017.05.008.

[32] XSSed.com. Cross Site Scripting (XSS) attacks information and archive n.d. http://www.xssed.com/ (accessed July 20, 2018).

[33] openbugbounty.org. Open Bug Bounty n.d. https://www.openbugbounty.org/ (accessed July 25, 2018).

[34] Cui B, Wei Y, Shan S, Ma J. The generation of XSS attacks developing in the detect detection. Adv. Broad-Band Wirel. Comput. Commun. Appl. Lect. Notes Data Eng. Commun. 2 Technol., vol. 2, Cham: Springer; 2017, p. 353–61. https://doi.org/10.1007/978-3-319-49106-6_33.

[35] Overview — html5lib 1.1-dev documentation n.d. https://html5lib.readthedocs.io/en/latest/ (accessed January 3, 2020).

[36] Richardson L. Beautiful Soup Documentation 2019.

[37] Ariya H. Esprima 2017.

[38] Martin M, Lam MS. Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking. USENIX Secur Symp 2008:31–43. https://doi.org/10.5555/1496711.1496714.

[39] Adi E, Baig Z, Hingston P. Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services. J Netw Comput Appl 2017;91:1–13. https://doi.org/10.1016/j.jnca.2017.04.015.

[40] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD '16, vol. 42, New York, USA: ACM Press; 2016, p. 785–94. https://doi.org/10.1145/2939672.2939785.

[41] Nielsen D. Tree Boosting With XGBoost. 2016.

[42] James G, Daniela W, Trevor H, Robert T. An Introduction to Statistical Learning with Applications in R. Vol. 112. New York: springer; 2013.

[43] Mantovani RG, Rossi ALD, Vanschoren J, Bischl B, de Carvalho ACPLF. Effectiveness of Random Search in SVM hyper-parameter tuning. 2015 Int. Jt. Conf. Neural Networks, vol. 2015- Septe, IEEE; 2015, p. 1–8. https://doi.org/10.1109/IJCNN.2015.7280664.

[44] Manning CD, Christopher D M, Schütze H. Foundations of statistical natural language processing 1999.