Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 5360472, 9 pages
https://doi.org/10.1155/2017/5360472

Research Article

Defending Malicious Script Attacks Using Machine Learning Classifiers

Nayeem Khan, Johari Abdullah, and Adnan Shahid Khan

Department of Computer Systems & Communication Technologies, Faculty of Computer Science & Information Technology, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia

Correspondence should be addressed to Nayeem Khan; 15010049@siswa.unimas.my

Received 27 October 2016; Accepted 29 December 2016; Published 7 February 2017

Academic Editor: Paul Honeine

The web application has become a primary target for cyber criminals by injecting malware especially JavaScript to perform malicious activities for impersonation. Thus, it becomes an imperative to detect such malicious code in real time before any malicious activity is performed. This study proposes an efficient method of detecting previously unknown malicious java scripts using an interceptor at the client side by classifying the key features of the malicious code. Feature subset was obtained by using wrapper method for dimensionality reduction. Supervised machine learning classifiers were used on the dataset for achieving high accuracy. Experimental results show that our method can efficiently classify malicious code from benign code with promising results.

1. Introduction

Due to the rapid advancement of modern computers and network infrastructure and our dependence on the Internet and its services which are increasing, web applications which are accessed through web browsers have become a primary target for cyber criminals. As per a report released by Symantec in 2016 [1], about 430 million unique pieces of malware were discovered showing a growth of 36% against the year 2014. Cyber criminals use the malicious code and malicious URLs to attack individuals and organizations. The sole purpose of such attacks is for personal, financial, and political gains through damaging or disrupting normal operation of computer and systems, performing phishing attacks, displaying unwanted advertisements, and extorting money. Thus detection of such malicious code attacks becomes a topmost security challenge.

Open Web Application Security Project (OWASP) has ranked cross-site scripting (XSS) as the 2nd most dangerous vulnerability among top ten vulnerabilities. Currently, XSS holds a share of 43% among all the reported vulnerabilities. XSS is a type of injection attack in which malicious scripts are injected around benign code in a legitimate webpage in order to access cookie, session, and other secret information. Web applications are used to transport malicious scripts to perform the attack. The target of XSS attack is a client side whereas SQL injections target server-side [2]. XSS attack is a vulnerability at the application layer of network hierarchy, which occurs by injecting malicious scripts to break security mechanism. About 70% attacks are reported to occur at application layer. Web browsers are the most susceptible application layer software for attacks. The purpose of the web browser is to get the requested web resource from the server and displayed in browser's windows. The format of the supplied resources is not restricted to HyperText Markup Language (HTML) but can also be portable document format (PDF), image, and so on. Attackers run malicious JavaScript in a web browser to target users. Malicious and obfuscated URLs also serve as a carrier for XSS attacks [3].

JavaScript is a programming/scripting language, used in web programming for making web pages more interactive, adds more features, and improves the end user experience. JavaScript also helps in reducing the server-side load and helps in shifting some computation to end user side. The JavaScript is commonly used for client-side scripting to be used in web browsers. When a user requests for a certain web page through a web browser, the server responds to the request and sends back web page which might have

JavaScript embedded into it. Then the JavaScript interpreter inside the web browser at the user's side starts executing it directly and automatically unless explicitly disabled by the user. Unfortunately, JavaScript's code provides a strong base for conducting attacks, especially drive-by-download attacks, which unnoticeably attack clients amid the visit of a web page. In contrast to different sorts of network-based attacks, malicious JavaScripts are difficult to detect [4]. JavaScript attacks are performed by looking into the vulnerability and exploiting by using JavaScript obfuscation techniques to evade the detection. Thus detecting of such malicious JavaScript's attacks at the real time becomes imperative.

Security solution providers who play a vital role in providing tools for defending attacks are fundamentally based on two correlative approaches, signature-based and heuristic-based detection approaches. The signature-based approach is based upon the detection of unique string patterns in the binary code. Signature-based attack detection approach fails when a new type of attack occurs. In this approach, the instance of new attack needs to be captured first in order to create a signature from it and then subsequently update the definitions of the detection tool on the client side. As the time gap between the capturing of new attack instance and the updating at the client side is high, it may make millions of client-side machines vulnerable to attacks. Cyber criminals take advantage of this time gap and vulnerability and simultaneously launch the attack. Consequently, this approach is unacceptable in an environment where new attacks are expected to come. Another tradition approach for attack detection is heuristic-based detection. Heuristic-based detection is based upon the set of expert decision rules to detect the attack. The advantage of using this attack approach is that it cannot only detect modified or variant existing malware but can also detect the previous unknown attack. The downside of using this approach is that it takes a long time in performing scanning and analysis, which drastically slows down the security performance. Another problem of the approach is that it introduces many false positives. False positive occurs when a system wrongly identifies code or a file as malicious when actually it is not. Some of the heuristic approaches include file emulation, file analysis, and generic signature detection.

Researchers recently have enlisted machine learning in the detection of malware. The advantage of using machine learning is that it can determine whether a code or a file is malicious or not in a very small time without the need of isolating it in a sandbox to perform the analysis. Machine Learning is able to detect previously unknown malware with predictive capabilities and especially useful for the detection of increasingly polymorphic nature of malware. To perform the classification of a benign and a malicious code, machine learning classifiers are used in detection through learning from patterns.

Keeping in view the impact that an attack can cause on the client side, detection of JavaScript at a real time thus became necessary. Several approaches have been proposed for classification and detection of malicious JavaScript code from the benign code of a website such as those of Rieck et al. [5], Curtsinger et al. [6], and Fraiwan et al. [7], the problem of these approaches is time overhead in detection. Other approaches such as GATEKEEPER [8] and Google Caga [9] use a method of executing random JavaScripts from the code in a protected environment but can detect the limited type of attacks.

Antivirus software, whether online or offline, uses signature-based techniques for detection of malware, thus requiring a constant update for new malware signatures from vendors for detection. The time gap between the detection of new malware and updating the virus signature at the client side may take a long time. It thus leaves the systems open for vulnerabilities until the virus definition has not been released by the vendor and updated by the client. Thus antiviruses are ineffective in current scenario for defending zero-day attacks where new malware variants are put in wild every day. In this paper, we introduced an interceptor for detection of malicious JavaScript attacks coming towards the client side based on machine learning for detection of malware especially previously unknown malware variants. The proposed approach is light weight in nature with minimal runtime overheads. Detection is based on the static analysis of a code for extracting features from given JavaScript to be fed into classifier for the classification process. Experiments were conducted by using a dataset of 1924 instances of JavaScript with 409 as malicious and 1515 as benign. Machine language classifiers such as Naïve Bayes, SVM, J48, and k-NN were individually tested for their performance of the selected features. The proposed approach is browser and platform independent and is depicted in Figure 1. This study is a continuation of our previous study [10]. The rest of the paper is organized as follows: Section 2 discusses the architecture of the proposed study. Section 3 explains the classification model. Section 4 provides details about experiments and analyzes their results. Section 5 deals with the related work. Section 6 provides conclusion and future work.

2. Architecture of Proposed Approach

Keeping in view the need for the real-time detection of malicious JavaScript code, we proposed an approach for detection of such attacks; the proposed approach consists of an interceptor between browser and server for detection of malicious JavaScript code. In this approach, all the traffic between server and client is exchanged through the interceptor to check for possible attacks in the source code to be executed by the browser. There are no direct communication channels between browser and server. This interceptor is proposed to protect the client side from being attacked or delivered with the malicious mode. The interceptor uses a static analysis for detection of malicious JavaScript attack. The interceptor will be installed as a plugin on the browser. Normally, when a client intends to visit a website by typing the URL into the address bar, an HTTP GET request is sent to the web server for lookup of the requested web page and if found response is generated, and the cookie is set in the browser. In this approach, the response from the server is passed via an interceptor to find a malicious code. Features are extracted from the source code of the web page which is fed into the

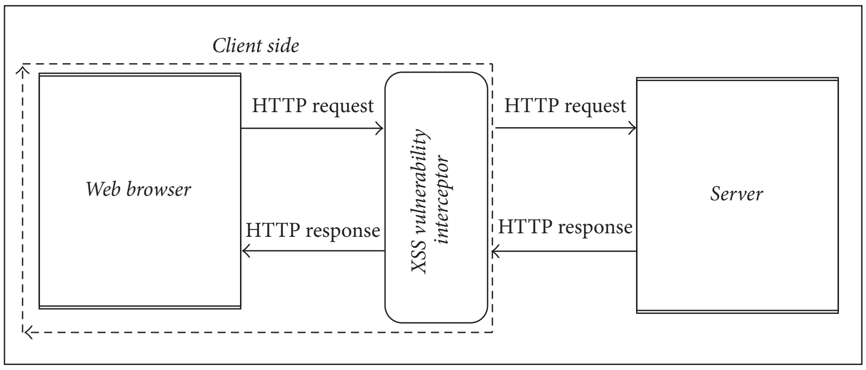

Figure 1: System figure of interceptor.

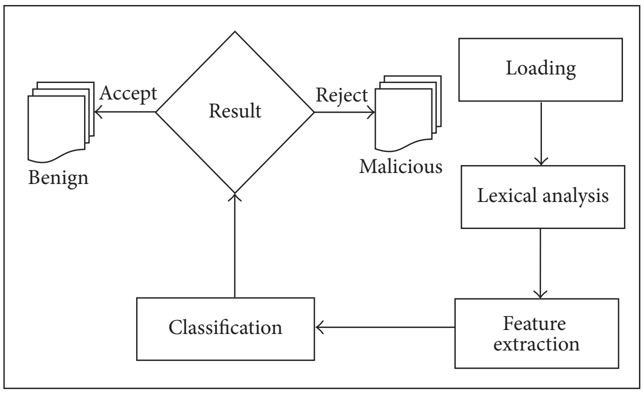

Figure 2: Schematic diagram of interceptor.

machine learning classification for decision. If any malicious JavaScript found, the page is blocked before it is interpreted by the browser.

The process of detection of malicious JavaScripts is done in few steps as shown in Figure 2. At the first step, the web browser sends a request to a web server for a specific page using its URL; all the subsequent web pages are cached by the loader into memory for the purpose to make succeeding requests for the same web domain faster while the background check of the whole domain has already been done in the background by interceptor in advance. Only the web pages up to level two will be loaded from the seed URL into the memory on a visit. Once the user starts browsing and diving deep into the website the progression is further extended automatically to next two levels with a maximum limit of 500 web pages per level, which sums up 7 MB of JavaScript code per level with the average of JavaScript code found on each web page being 15 KB as per [11]. The source code of the web page needs to be a breakdown by performing lexical analysis to generate a sequence of the tokens to be parsed. In this method, we used Yacc parser (yet another compiler-compiler) which is an LALR (Look Ahead left Right) algorithm based parser to generate tokens from JavaScripts [12, 13]. The parser reads and analyzes an input stream of JavaScripts which breaks them into component pieces. Each component piece obtained is a token which may be a keyword, string, number, or punctuation, which cannot be further broken. The tokens obtained after parsing are arranged according to the input structure of JavaScript code obtained from the web page and grammar rules of the parser. The benefit of using LALR type of parser is to generate a large parsing table from the input stream of JavaScripts from where a larger set of tokens can be drawn. The tokens obtained from parsing are stored in a file to be used in further processing. To evade the detection attackers employ obfuscation technique in an attempt to infuse malicious code. A wide range of obfuscation approaches are used such as encoding, escaped ASCII, string split, and hidden iFrame to avoid detection. As the obfuscation is becoming more and more sophisticated, any such obfuscated code will be detected with a high degree of accuracy and precision during lexical analysis. Extraction of features is a key step to perform the transformation of unstructured and semistructured contents into a structured form which is used for classification. The final step is to perform classification, which will accurately predict to which class a new script belongs, based upon the observations on the training set whose class is already known. Once a malicious script is detected the page will be blocked.

3. Building the Classification Model

3.1. Dataset. In order to perform classification with high accuracy, the quality of dataset is very important, which must contain the instances of benign and malicious JavaScripts so that the distribution of samples emulates distribution in the web. Dataset used for conducting experiments consists of total 1924 instances with 1515 instances of benign and 409 malicious instances. The dataset has been obtained from [14].

3.2. Feature Selection. A number of features can be extracted from JavaScripts, but not all of them will be helpful in accurate detection of malicious JavaScripts. In the feature selection process, a classification algorithm is encountered with a problem of selecting a relevant set of features as a primary focus while ignoring others in order to achieve high accuracy. To obtain high accuracy with a particular classification algorithm on a specific dataset, feature subset selection plays a crucial role in deciding how a classification

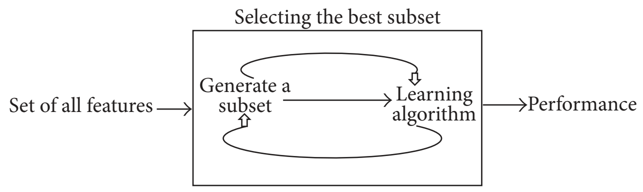

Figure 3: Feature selection wrapper method.

Table 1: Feature subset obtained by applying wrapper method.

Classifier	Feature subset
Naïve Bayes	7, 9, 15, 17.25, 26, 36, 70
J48	3, 7, 17, 69, 70
SVM	9, 12, 16, 17, 20, 53, 70
K-NN	7, 11, 69, 70

and the training set interact. Taking this into consideration, we performed feature subset selection using wrapper method in order to generate a feature subset which is likely to be more predictive with a specific classification algorithm. We have extracted 70 features of JavaScripts as shown in the Appendix. In wrapper method, the feature selection is done by an induction algorithm as a black box where the algorithm conducts a search for a good subset itself as a part of the evaluation process as shown in Figure 3 [15]. The accuracy of the induced classifier is estimated using accuracy estimation techniques [16]. In this study, we employed classifier subset evaluator along with the best first search engine to obtain an optimal feature subset. Classifier subset evaluator evaluates attribute subsets on training data or performs a seperate holdout test on the testing set and uses a classifier to estimate the "merit" of a set of attributes [17]. We obtained subsets by using a classifier subset evaluator along with the best first search engine. The subsets obtained for different classifiers are given in Table 1.

3.3. Classifiers. Researchers in the field of machine learning have proposed many algorithms for classification in the past. Some of the popular known algorithms are SVM, Naïve Bayes, decision tree, K-Nearest Neighbour (K-NN), Random Forests, and so on. The selection of algorithm depends on the size of the training set and also on the basis of accuracy, training time, linearity, the number of parameters, and the number of features. If the training set is small in size, then high variance low bias classifiers are used and if the training set is large for the low variance, high bias classifiers are used. Machine learning algorithms are divided into three categories based on their learning style. They are supervised, unsupervised, and semisupervised algorithms. The proposed approach will use 4 supervised machine learning classifiers, and their results will suggest which classifier to be used in the interceptor for the real-time detection malicious code.

3.3.1. Naïve Bayes Classifier. Naïve Bayes [18] classifier technique is based on Bayes theorem with independent assumption between predictors. Naïve Bayes model is easy to build when the dimensionality of the dataset is very large. In the context of learning the process, the classifier tokenizes training data to some tokens x_i ($i = 1, \ldots, n$) and counts the number of occurrences of x_i in each class. Based on this process the likelihood of each class is computed with some test data and classifies that test data for the class which has the highest likelihood.

Despite its simplicity, Naïve Bayes classifier is widely used as it surpasses more sophisticated classification methods. Bayesian classifier is based on Bayes theorem, which says

$$p\left(c_j \mid d\right) = \frac{p\left(d \mid c_j\right) p\left(c_j\right)}{p\left(d\right)}, \tag{1}$$

where $p(c_j \mid d)$ is probability of instance d being in class c_j, $p(d \mid c_j)$ is probability of generating instance d given class c_j, $p(c_j)$ is probability of occurrence of class c_j, and $p(d)$ is probability of instance d occurring.

In our study, there are only two classes malicious and benign for classification of the range of j is from 1 to 2 only.

3.3.2. Support Vector Machines. Support Vector Machines (SVM) were developed by Boser et al. in 1992 [19]. However, the original optimal hyperplane algorithm was introduced by Vapnik and Lerner in 1963 [20]. SVM is considered as one of the most effective models for binary classification of high-dimensional data. An SVM performs classification by constructing an N-dimensional hyperplane that optimally separates the data into two categories to maximize the distance of the hyperplane. SVM can also be extended to the data that is not linearly separable by applying kernel to map the data into a higher-dimensional space and to separate the data on the mapped dimension.

3.3.3. K-Nearest Neighbour. The K-Nearest Neighbour Algorithm (KNN) [21] is the simplest machine learning algorithm. To determine the category of the test data, K-NN performs a test to check the degree of similarity between documents and k training data to store a certain amount of classified data. Since k-NN classifies instances, in our research, it will be malicious and benign code instances nearest to the training space. The classification of unknown instances is performed by measuring the distance between the training instance and unknown instance. Since instances are classified based upon the majority vote of neighbour, the most common neighbour is measured by a distance function. If $k = 1$, then the instance is assigned for the class of its nearest neighbour. In n-dimensional space distance between two points x and y is achieved by using any distance function as follows:

Euclidean distance function:

$$\sqrt{\sum_{i=1}^{k} \left(x_i - y_i\right)^2}, \tag{2}$$

Manhattan distance function:

$$\sum_{i=1}^{k} \left|x_i - y_i\right|, \tag{3}$$

Minkowski distance function:

$$\left(\sum_{i=1}^{k}\left(\left|x_i - y_i\right|\right)^q\right)^{1/q}. \quad (4)$$

3.3.4. Decision Trees. Decision tree learners are a non-parametric supervised method used for classification and regression. In a decision tree, classifier is represented as a tree whose internal nodes represent the condition of the variable and final nodes or leaves are the final decision of the algorithm. In the process of classification, a well-formed decision tree can efficiently classify a document by running a query from the root not until it reaches a certain node. The main advantage of using decision tree is that it is simple and easy to understand and interpret for naïve users. The risk associated with decision tree is overfitting which occurs when a tree is fully grown, and it may lose some generalization capabilities. Some common reasons of overfitting are the presence of noise, lack of representation instance, and multiple comparison procedures. Overfitting can be avoided by severing approaches such as prepruning and postpruning.

3.4. Performance Evaluation. Performance evaluation acts as a multipurpose tool which is used to measure the actual values within the system against expected values. Our purpose of evaluation is to study and measure the performance of a classifier in detecting malicious code. In order to achieve high results from the proposed approach, we are deeply concerned with the accuracy which is defined as

$$\text{Accuracy} = \frac{\text{No. of Classified Benign Scripts}}{\text{Total Benign Samples}} \times 100. \quad (5)$$

A false positive scenario occurs when the attack detection approach mistakenly treats a normal code as a malicious code. In a given approach, a false negative occurs when a malicious code is not detected despite its illegal behaviour. Detection rate is measured by using confusion matrix or error matrix for the assessment of false positives, and false negatives false positive and false negative detection rate is calculated by

$$\text{FPR} = \frac{\text{FP}}{N} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \quad (6)$$

And the false negative rate is calculated by

$$\text{FNR} = \frac{\text{FN}}{R} = \frac{\text{FN}}{\text{FN} + \text{TP}}, \quad (7)$$

where FPR is false positive rate, FNR is false negative rate, FN is false negative, TN is true negative, and TP is true positive.

True negative shows a number of negative samples correctly identified, false negative implies a number of malicious samples identified as negative, false positive indicates the number of negative samples identified as malicious, and true positive shows a number of malicious samples correctly identified [22]. The performance of the proposed detection approach will be the rate at which the malicious scripts are processed. The performance will be calculated by latency time which is taken in presence and absence of interceptor to display a page and another by calculating the system resource consumption in both scenarios. Achieving real-time detection is not possible where the detection system is poor. Receiver operating characteristic (ROC) had also been used for the evaluation process. ROC is a graphical plot generated by using a fraction TPR against the fraction of FPR for a binary classifier at various thresholds.

4. Experimental Results and Analysis

In our experiments, we used a dataset of total 1924 instances with 1515 instances of benign and 409 malicious instances. The aim of performance evaluation is to study and analyze the performance of classifiers in correctly classifying the instance by using the evaluation metrics such as accuracy, true positive rate, and false positive rate. In this study, we have conducted three experiments. In Experiment I, we used 100% labels as training. In Experiment II, the entire sample set is split into two parts: the training 80% and testing 20%. In Experiment III, each classifier was evaluated using 10-fold cross-validation. K-fold cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model and a test set to evaluate it, where k ranges from 1 to $k - 1$. In 10-fold cross-validation the data is portioned into ten subsets for each portioned subset the remaining nine subsets are used to train the classifier and the final subset is considered as a test set. In all the three conducted experiments, four classifiers such as Naïve Bayes, J48, SVM, and K-NN were used. For SVM, RBF kernel was used. In order to determine the parameter regularization, we used grid-search to determine C and gamma using cross-validation. Various pairs of (C, γ) values were tested and the one with the best cross-validation accuracy was picked; RBF kernel $C = 250007$ and $\gamma = 0.01$ were found optimal. For KNN, the Euclidean distance function was used as given in (2). The best value for K was chosen automatically by WEKA, which uses cross-validation and $K = 10$ was found optimal. The features used for the classification process were chosen by using wrapper method. The feature set obtained from wrapper method with respect to individual classifiers is given in Table 1. The results obtained from three experiments are listed in Tables 2, 3, and 4, respectively.

As shown in Tables 2, 3, and 4, the overall performances of all the four classifiers are good, but certain classifier performs better in each of the experiments. SVM in all the three experiments achieved low accuracy, as low as 94.55% in Experiment II. KNN in Experiment I outperformed with 100% accuracy rate. However, for KNN, a decrease in performance was observed within Experiments II and III, respectively. J48 achieved an accuracy of 97.09% in Experiment I, and it increased in Experiment II to 99.22%. The ROC curve obtained show that KNN achieved ROC = 1 in Experiment I followed by the J48 classifier in Experiment III with ROC = 0.983. Comparing the results obtained in all the

Table 2: Results obtained from four classifiers by using 100% training.

Classifier	Accuracy	TP rate	FP rate	Precision	Recall	F-measure	ROC	Class
Naïve Bayes	97.25%	0.875	0.001	0.994	0.875	0.931	0.943	Malicious
		0.999	0.125	0.967	0.999	0.983	0.943	Nonmalicious
J48	97.09%	0.966	0.003	0.99	0.966	0.978	0.982	Malicious
		0.997	0.034	0.991	0.997	0.994	0.982	Nonmalicious
SVM	96.51%	0.912	0.02	0.923	0.912	0.918	0.946	Malicious
		0.98	0.088	0.976	0.98	0.978	0.946	Nonmalicious
KNN	100.00%	1	0	1	1	1	1	Malicious
		1	0	1	1	1	1	Nonmalicious

Table 3: Results obtained from four classifiers by performing 10-fold cross-validation.

Classifier	Accuracy	TP rate	FP rate	Precision	Recall	F-measure	ROC	Class
Naïve Bayes	97.99%	0.868	0.001	0.994	0.868	0.927	0.963	Malicious
		0.999	0.132	0.966	0.999	0.982	0.936	Nonmalicious
J48	98.64%	0.951	0.004	0.985	0.951	0.968	0.971	Malicious
		0.996	0.049	0.987	0.996	0.991	0.971	Nonmalicious
SVM	95.42%	0.848	0.017	0.93	0.848	0.887	0.916	Malicious
		0.983	0.152	0.96	0.983	0.971	0.916	Nonmalicious
KNN	96.41%	0.927	0.026	0.907	0.927	0.917	0.956	Malicious
		0.974	0.073	0.98	0.974	0.977	0.956	Nonmalicious

Table 4: Results obtained from four classifiers for 80% training and 20% testing.

Classifier	Accuracy	TP rate	FP rate	Precision	Recall	F-measure	ROC	Class
Naïve Bayes	95.06%	0.795	0.007	0.971	0.795	0.874	0.964	Malicious
		0.993	0.205	0.946	0.993	0.969	0.965	Nonmalicious
J48	99.22%	0.964	0	1	0.964	0.982	0.983	Malicious
		1	0.036	0.99	1	0.995	0.983	Nonmalicious
SVM	94.55%	0.88	0.036	0.869	0.88	0.874	0.922	Malicious
		0.964	0.12	0.967	0.964	0.965	0.922	Nonmalicious
KNN	97.14%	0.904	0.01	0.962	0.904	0.932	0.957	Malicious
		0.99	0.096	0.974	0.99	0.982	0.957	Nonmalicious

three experiments, it can be concluded that KNN performed very well and has achieved 100% accuracy with ROC = 1. All the experiments were run on a PC with Quad core 3.6 GHZ processor with 16 GB primary memory. In order to know the computation costs, the same PC was used with the Internet speed of 2.02 Mbps as download speed and 3.90 Mbps as upload speed. The time between the first requests made by the user till the display of first page was recorded with and without interceptor. It was found that the average time to display a web page without interceptor was 0.55 seconds, whereas the average time for displaying a single web page after classification was 0.97 seconds. WEKA was used to obtain the classification results. WEKA is a collection of machine learning algorithms for data mining tasks developed by the University of Waikato, New Zealand [23].

5. Related Work

Malicious JavaScript attacks have become an essential business proposition due to the seriousness of their threat and impact. Detection of malicious JavaScript attacks thus became a vital field of research. Despite a number of techniques for detection of JavaScript attack have been proposed, JavaScript attacks still remain a threat to users. Thus an efficient approach to mitigate XSS is demanded. Researchers took the services of machine learning to find the accurate detection approach for detection of previously unknown attacks. In 2005, Kam and Thi [24] were the first to explore the possibility of using machine learning for detection malicious web page. Their work limited up to the detection of a malicious web page based on URL by performing lexical

Table 5: Complete feature list.

(1)	Number_Of_Redirections
(2)	Difference_In_Redirections
(3)	Number_Of_Instantiated_Objects
(4)	Difference_In_Instantiated_Objects
(5)	Length_Of_Longest_Single_Evaluated_Code
(6)	Total_Bytes_Of_Evaluated_Code
(7)	Total_Bytes_Allocated
(8)	Number_Of_Executions
(9)	Length_Of_Longest_Unprintable_String
(10)	Number_Of_Unprintable_Strings
(11)	Ratio_Of_Definitions_To_Uses
(12)	Length_Of_Method_Call_Parameter
(13)	Total_Number_Of_Method_Calls
(14)	Memory_Overflow_Occured
(15)	Unprintable_String_Used_For_Instantiated_Object_Parameter
(16)	Unprintable_Strings_AND/OR_Redirects
(17)	Method_Cal
(18)	open
(19)	run
(20)	Savetofile
(21)	Send
(22)	Setattribute
(23)	Write
(24)	timespanformat
(25)	shellexecute
(26)	playerproperty
(27)	uploadlogs
(28)	hgs_startnotify
(29)	downloadandinstall
(30)	getvariable
(31)	linksbicons
(32)	openurl
(33)	getresponseheader
(34)	keyframe
(35)	setrequestheader
(36)	Setslice
(37)	buildpath
(38)	getspecialfolder
(39)	environment
(40)	rawparse
(41)	iestartnative
(42)	setformatlikesample
(43)	createobject
(44)	specialfolders
(45)	replace
(46)	createnewfoldername
(47)	addevent
(48)	isversionsupported
(49)	split
(50)	new
(51)	evaluate
(52)	msdatasourceobject
(53)	allowscriptaccess
(54)	concat
(55)	url
(56)	mode
(57)	type
(58)	Import
(59)	close
(60)	allowcontextmenu
(61)	cachefolder
(62)	compressedpath
(63)	console
(64)	printsnapshot
(65)	shownavigationbuttons
(66)	snapshotpath
(67)	wkspictureinterface
(68)	zoom
(69)	script_Size
(70)	intent

analysis. This study does not check the code on the web page for any malicious code. In 2006, Kolter and Maloof [25] used 4-gram sequence for feature extraction of features from two 500 n-grams by using a measure of information gain. Classifiers such as Naïve Bayes, SVM, and decision trees were used for classification. Results show that only J48 classifier could achieve an AUC of 0.996. In 2007 Garera et al. [26] used regression analysis by using about 18 selected features for detection of a phishing website. An accuracy of 97.3% was achieved by using a small data set of about 2500 URLs. In 2008, McGrath and Gupta [27] try to detect phishing URLs by performing a comparative analysis of phishing and nonphishing URLS. Their study does not use any classifier. In 2008, Polychronakis and Provos [28] carried out a study for drive-by exploit of URLs by using machine learning classifiers prefilters. The limitation to this approach is that it is time-consuming as it employs a heavyweight classifier for classification. Based on dynamic analysis approach Ahmed et al. in 2009 [29] proposed an approach for malware detection by extracting features in combined from spatial and temporal information from API at runtime. In 2012, Abbasi et al. [30] used a very small dataset for classification of fake medical websites. This study is restricted only up to the detection of the fake medical website and cannot detect other types of fake or malicious websites. Huda et al. in 2016 [31] proposed an approach for malware detection based on API call statistics as a feature for the SVM classifier. Soska et al. proposed an approach for detection of malicious websites based on features extracted from network traffic statistics, file system, and contents on the web pages using C4.5 decision tree based algorithm. Another study conducted by Alazab in 2015 [32] proposed an approach for detection of malware based on API call sequence quite similar to [31], by using K-NN classifier. Al-Taharwa et al. in 2015 [33] proposed an approach for detection of deobfuscation of JavaScripts for detection of malicious JavaScripts. Our work is based on extracting features based on their quality, which is precisely analyzed to be used during the classification process. These features act as pivotal in initiating a JavaScript attack.

6. Conclusion and Future Work

In this paper, we proposed an interceptor for detection of malicious JavaScripts and conducted experiments. The proposed approach achieved an accuracy of 100% in detection for previously unknown malicious JavaScript's attacks based upon the learning. Experimental results show that ROC = 1 was achieved by KNN classifier with no false positive. The wrapper method played an important role in feature selection, which leads to high accuracy compared to other studied static approaches.

Appendix

See Table 5.

Competing Interests

The authors declare that there is no conflict of interests between the publication of this paper and the funding agency, Universiti Malaysia Sarawak.

Acknowledgments

The research conducted is part of the DPP (Ph.D. Research Grant) under Universiti Malaysia Sarawak (Research Grant no. F08(DPP28)/1217/2015(03)).

References

[1] "Internet Security Threat Report," Symantec, Vol. 21, April 2016.

[2] M. Van Gundy and H. Chen, "Noncespaces: using randomization to enforce information ow tracking and thwart cross-site scripting attacks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '09)*, pp. 1–18, San Diego, Califo, USA, February 2009.

[3] 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 *Proceedings of the 17th IEEE Symposium on Computers and Communication (ISCC '12)*, pp. 702–707, IEEE, Cappadocia, Turkey, July 2012.

[4] K. Schütt, M. Kloft, A. Bikadorov, and K. Rieck, "Early detection of malicious behavior in javascript code," in *Proceedings of the 5th ACM Workshop on Artificial Intelligence and Security (AISec '12)*, pp. 15–24, Raleigh, NC, USA, October 2012.

[5] K. Rieck, T. Krueger, and A. Dewald, "Cujo: efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*, pp. 31–39, Austin, Tex, USA, December 2010.

[6] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "ZOZZLE: fast and precise in-browser JavaScript malware detection," in *Proceedings of the 20th USENIX Conference on Security (SEC '11)*, San Francisco, Calif, USA, 2011.

[7] M. Fraiwan, R. Al-Salman, N. Khasawneh, and S. Conrad, "Analysis and identification of malicious JavaScript code," *Information Security Journal*, vol. 21, no. 1, pp. 1–11, 2012.

[8] S. Guarnieri and V. B. Livshits, "GATEKEEPER: mostly static enforcement of security and reliability policies for javascript code," in *Proceedings of the 18th Conference on USENIX Security Symposium (SSYM '09)*, vol. 10, Montreal, Canada, August 2009.

[9] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay, "Safe active content in sanitized JavaScript," Tech. Rep., Google, 2008.

[10] N. Khan, J. Abdullah, and A. S. Khan, "Towards vulnerability prevention model for web browser using interceptor approach," in *Proceedings of the 9th International Conference on IT in Asia (CITA '15)*, pp. 1–5, Seoul, Korea, August 2015.

[11] T. Johnson and P. Seeling, "Desktop and mobile web page comparison: Characteristics, trends, and implications," *IEEE Communications Magazine*, vol. 52, no. 9, pp. 144–151, 2014.

[12] S. C. Y. Johnson, *Yet Another Compiler-Compiler*, vol. 32, Bell Laboratories, Murray Hill, NJ, USA, 1975.

[13] DeRemer, Franklin L. Practical translators for LR (k) languages, Diss. MIT, 1969.

[14] leakiEst, *School of Computer Science*, University of Birmingham, Birmingham, UK.

[15] R. Kohavi and G. H. John, "The wrapper approach," in *Feature Extraction, Construction and Selection*, pp. 33–50, Springer, New York, NY, USA, 1998.

[16] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, ACM, July 1992.

[17] J. Zurada, "Does feature reduction help improve the classification accuracy rates? a credit scoring case using a german data set," *Review of Business Information Systems (RBIS)*, vol. 14, no. 2, 2010.

[18] D. Lewis David, "Naive (Bayes) at forty: the independence assumption in information retrieval," in *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings*, vol. 1398 of *Lecture Notes in Computer Science*, pp. 4–15, Springer, Berlin, Germany, 1998.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, Pittsburgh, Pa, USA, July 1992.

[20] V. Vapnik and A. Lerner, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, no. 6, pp. 774–780, 1963.

[21] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[22] 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 *Proceedings of the 17th IEEE Symposium on Computers and Communication (ISCC '12)*, pp. 000702–000707, Cappadocia, Turkey, July 2012.

[23] Weka, University of Waikato, New Zealand.

[24] M.-Y. Kan and H. O. N. Thi, "Fast webpage classification using URL features," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM '05)*, pp. 325–326, Bremen, Germany, November 2005.

[25] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[26] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the ACM Workshop on Recurring Malcode (WORM '07)*, pp. 1–8, ACM, Alexandria, Va, USA, November 2007.

[27] D. K. McGrath and M. Gupta, "Behind phishing: an examination of phisher modi operandi," *LEET*, vol. 8, p. 4, 2008.

[28] M. Polychronakis and N. Provos, "Ghost turns zombie: exploring the life cycle of web-based malware," *LEET*, vol. 2, pp. 1–8, 2008.

[29] F. Ahmed, H. Hameed, M. Zubair Shafiq, and M. Farooq, "Using spatio-temporal information in API calls with machine learning algorithms for malware detection," in *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence (AISec '09)*, Chicago, Ill, USA, November 2009.

[30] A. Abbasi, F. M. Zahedi, and S. Kaza, "Detecting fake medical web sites using recursive trust labeling," *ACM Transactions on Information Systems*, vol. 30, no. 4, article 22, 2012.

[31] S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, "Hybrids of support vector machine wrapper and filter based framework for malware detection," *Future Generation Computer Systems*, vol. 55, pp. 376–390, 2016.

[32] M. Alazab, "Profiling and classifying the behavior of malicious codes," *Journal of Systems and Software*, vol. 100, pp. 91–102, 2015.

[33] I. A. Al-Taharwa, H.-M. Lee, A. B. Jeng, K.-P. Wu, C.-S. Ho, and S.-M. Chen, "JSOD: JavaScript obfuscation detector," *Security and Communication Networks*, vol. 8, no. 6, pp. 1092–1107, 2015.