MALICIOUS JAVASCRIPT DETECTION BASED ON CLUSTERING TECHNIQUES

Nguyen Hong Son[1] and Ha Thanh Dung[2]

[1]Faculty of Information Technology
Posts and Telecommunications Institute of Technology, Vietnam
[2]Falculty of Information Technology Saigon University, Vietnam

ABSTRACT

Malicious JavaScript code is still a problem for website and web users. The complication and equivocation of this code make the detection which is based on signatures of antivirus programs becomes ineffective. So far, the alternative methods using machine learning have achieved encouraging results, and have detected malicious JavaScript code with high accuracy. However, according to the supervised learning method, the models, which are introduced, depend on the number of labeled symbols and require significant computational resources to activate. The rapid growth of malicious JavaScript is a real challenge to the solutions based on supervised learning due to the lacking of experience in detecting new forms of malicious JavaScript code. In this paper, we deal with the challenge by the method of detecting malicious JavaScript based on clustering techniques. The known symbols that will be analyzed, the characteristics which are extracted, and a detection processing technique applied on output clusters are included in the model. This method is not computationally complicated, as well as the typical case experiments gave positive results; specifically, it has detected new forms of malicious JavaScript code.

KEYWORDS

Malicious JavaScript, Detection model, K-means clustering algorithm, Prediction support parameter.

1. INTRODUCTION

Dynamic web developers usually use JavaScript because of its efficiency. The flexibility in applications of the form of web programming language has been exploited by cyber attackers for carrying out target attacks. The reason that cyber attackers often use malicious JavaScript to attack websites maybe because they find it convenient to evade. It is difficult for ordinary antivirus programs to detect malicious codes. Until now, most of the malicious JavaScript code detection methods mainly focus on improving the accuracy as much as possible. There have been many studies applying machine learning and deep learning to develop high-precision models for detecting malicious JavaScript. Those models are all based on supervised learning. Nevertheless, the accuracy of the methods mostly depends on the labeled dataset. In reality, new malicious scripts constantly evolve, which requires the frequent upgrade of the dataset. The detection methods have to be upgraded as the upgrade of the new dataset so that they can detect new malicious codes. Besides, the complexity and heavy computational load in the detection models are also a negative side of the detection methods because of their increased resource cost and slow speed in response, especially when operating in a multitasking environment. Therefore, the detection methods encounter the challenge that is not only to improve the precision but also to maintain it. Moreover, the methods should also remain cost-effective. The main goal of the paper is to suggest an effective detection method that has low cost as well as high accuracy; simultaneously, it can detect new malicious codes which have not been found in the dataset without upgrading the model. The method supports the implementation in practice with different defense scales.

Using an unsupervised learning algorithm to get rid of the dependency on the labeled dataset, based on which the unidentified new malicious codes are detected. Particularly, the detection model could be developed based on a clustering algorithm. Firstly, the JavaScript codes will be analyzed and the typical features will be selected for extraction. The clusters are then processed against a learned threshold value to determine the existence of malicious JavaScript codes in web content. The clustering technique has also been used in many studies as surveyed in [1]. Among the unsupervised machine learning algorithms, the Isolation Forest algorithm [3] is also commonly used in anomaly detection and we also apply it to the malicious codes detection model as another option to compare with the proposed method.

The rest of the paper is presented in the following sections: Section 2 will introduce some recent typical studies in attempts to thoroughly detect malicious JavaScript codes, which shows that machine learning has become the first choice for this task. Section 3 will represent in detail the proposed method which includes detection application, developing the model, and handling clusters for malicious codes detection. Next, section 4 will describe the experimental process and evaluation of the proposed method with different cases and corresponding results. The paper ends with the conclusions in section 5.

2. RELATED WORKS

Cyber attackers have used malicious JavaScript code as a handy tool to attack both websites and web users. In reality, malicious JavaScript is commonly used in types of attack such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), etc. Therefore, malicious JavaScript has become a concerning issue about information security. To go against these malicious attacks, it is crucial to detect malicious JavaScript as they appear on the web. Malicious JavaScript detection is not a leisurely task because of the diversity and constant volatility of this malicious code, especially in the context of high traffic usage. Currently, many studies have been attracted to find out the best solution to the challenge. In general, each proposed solution has its innovative detection technique, applying different technologies, such as the detection method based on static code analysis has proposed in [10]. Accordingly, the authors have introduced a detection framework called AMA (Amrita Malware Analyzer) that is capable of combating malicious JavaScript's evasion strategies. The main technique in this framework is to use the probable plaintext attack to deobfuscate the web malware. Recently, many studies have also appeared to exploit the advantages of machine learning technology to propose malicious JavaScript detection models. The proposal in [2] is a typical case that follows the direction. The authors in [2] also used static code analysis to analyze the scripts, add new features to the dataset, and apply various supervised algorithms to develop the classifiers. The classifiers achieved a high detection rate, which is about 97% to 99%. Other studies used machine learning proposed in [8,11,12]. The study in [11,12] uses Abstract Syntax Tree (AST) to represent the code structure and a machine learning method to conduct learning features called Doc2vec to detect malicious JavaScript. Doc2vec is a neural network that can learn the context of text with variable lengths. Through the representation of AST, the authors have built a new dataset to help train this neural network to have the ability to detect with high precision. The effectiveness of the methods of building malicious JavaScript detection models using machine learning requires having a numerous amount of labeled samples in the dataset, and this requirement is one of the difficulties when building a detection model. To overcome this problem, the authors in [4] suggested using Generative Adversarial Networks (GAN) and using the output from the GAN to train the classifiers. As a result, their method was able to achieve high precision with only a limited set of labeled samples. Besides the traditional methods of using shallow learning models to detect malicious JavaScript, there are also many studies using deep learning models to increase the detection ability of the model. For an instant, in the method proposed in [14], the authors built a deep learning network from many denoising auto-encoders called Stacked denoising auto-encoders (SdA) with the expectation of being able to extract more abstract features of JavaScript code, which results in high-precision detection. Being applied SdA together with logistic regression, their experiments gave precision as high as 95% and a false-positive rate less than 4.2%. Another method of using deep learning is proposed in [16]. The authors in [16] build a detection model based on the Bidirectional Long Short-Term Memory (BLSTM) neural network combined with the Program Dependency Graph (PDG) technique to preserve the rich semantic information. This method also gives very positive experimental results, with an accuracy of 97%. Similarly, the authors in [5] also analyzed the characteristics of JavaScript code, analyzed the extracted feature components, and used multilayer perceptrons to build a malicious JavaScript code detection model with an accuracy of 98.8% and a false positive rate of about 3%.

3. PROPOSED METHOD

3.1. Design of malicious JavaScript detection model

The main point of the detection application is an anomaly detection model based on an unsupervised machine learning algorithm. The application process is shown in Figure 1. First of all, the website data to be tested is collected and processed to retain only the features which are suitable to the model design. The output of the processing stage is a set of many data points, which may or may not contain malicious vectors. The processed data set will be fed into an unsupervised machine learning algorithm for detection. Unlike the classification model with supervised machine learning algorithms, the prediction results are not immediately available at the output of the model, but an extra computational step is needed to predict the anomaly. Depending on the type of unsupervised machine learning algorithm is used, there will be different prediction support parameters and how that parameter is calculated. Thus, in our proposal, there will be two important parts: building a model with an unsupervised learning algorithm and determining the prediction support parameter on the output of the model according to the selected unsupervised learning algorithm.

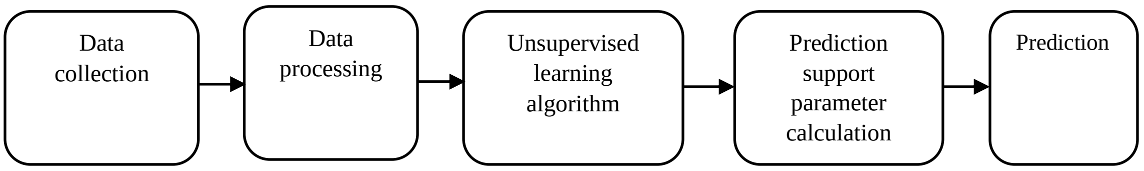

Data collection
Data processing
Unsupervised learning algorithm
Prediction support parameter calculation
Prediction

Figure 1. Process of detection application

The model is built through the following steps:

- Build the dataset
- Preprocessing and feature extraction
- Choose an unsupervised learning algorithm
- Check the discriminant ability of the model

The dataset is built using the same method as [2], which is collecting URLs containing malicious JavaScript and benign JavaScript, getting a list of malicious website addresses from the Phistank website [17], and getting a list of non-malicious website addresses from the Moz website [18]. After that, we will proceed to visit the website addresses that have been saved in the list to extract the JavaScript code. When the extraction is finished, there will be raw JavaScript data. The raw data contains the website name, JavaScript code and is labeled malicious or benign. In the raw data, we take 44 JavaScript features out of 77 features in [2]. The features are described in Table 1.

Table 1. JavaScript features used in our dataset

Sr.No.	Features (charater or function)	Sr.No.	Features (character or function)
1	#	23	eval
2	%	24	function
3	(	25	getElementById
4	)	26	indexOf
5	+	27	location
6	/	28	log
7	[	29	onerror
8	]	30	onload
9	{	31	parseInt
10	|	32	random
11	}	33	replace
12	addEventListener	34	return
13	attachEvent	35	search
14	charAt	36	setAttribute
15	Classid	37	setTimeout
16	Concat	38	split
17	Console	39	substring
18	Cookie	40	toString
19	createElement	41	unescape
20	Decode	42	var
21	Document	43	window
22	Escape	44	write

In the next step, we will build a dataset from the raw dataset with 44 features mentioned above by preprocessing using the TF-IDF method [13]. TF-IDF is an important technique used in information retrieval to estimate the importance of a word or phrase in a given text. In which the importance of words is quantified through the TF-IDF index calculation formula (1), the calculation process is as follows:

$$\mathrm{tf}(t,d) = \frac{f(t,d)}{\max\{f(w,d): w \in d\}} \qquad (1)$$

Where:

tf(t, d): frequency of occurrence of word t in the text d
f(t, d): Number of occurrences of the word t in the text d
max({f(w, d): w ∈ d}): Number of occurrences of the word with the most number of occurrences in the text d.

IDF (Inverse Document Frequency): helps to evaluate the importance of a word. When calculating TF, all words are considered to be of equal importance. However, some words such as "is", "of", and "that" often appear many times but their importance is not high. Thus, we need to reduce the importance of these words (2).

$$\mathrm{idf}(\mathrm{t}, \mathrm{D}) = \log \frac{|\mathrm{D}|}{|\{\mathrm{d} \in \mathrm{D}: \mathrm{t} \in \mathrm{d}\}|} \qquad (2)$$

Where:

idf(t, D): idf value of word t in the text set.
|D|: total number of documents in the set D.
|{d∈D:t∈d}|: represents the number of documents in set D containing the word t.

$$\mathrm{tfidf}(\mathrm{t}, \mathrm{d}, \mathrm{D}) = \mathrm{tf}(\mathrm{t}, \mathrm{d}) * \mathrm{idf}(\mathrm{t}, \mathrm{D}) \qquad (3)$$

After calculating the TF-IDF for all the features using (3), the dataset will contain the TF-IDF values corresponding to each feature of the data point, which can be called the TF-IDF dataset. The TF-IDF dataset is still full of features and needs to be reduced before being fed into the unsupervised machine learning algorithm. To reduce the number of features, we use the Sequential Feature Selector method, thereby identifying features with too small influence and eliminating them. The final dataset has only a few features that contribute significantly to the model's prediction results.

The unsupervised machine learning algorithm is chosen as the clustering algorithm. In this study we use the K-means algorithm [7], the K-means clustering algorithm is used quite commonly because of its efficiency in many applications. With our original design goal of low computational cost and fast response, K-means is considered a suitable choice.

The model will be evaluated according to its ability to distinguish the data objects to be predicted, which is also the ability to correctly cluster on the input data. The model will be trained by re-labeling after clustering, checking the accuracy of the clustering, and fine-tuning in the steps of building dataset and model parameters until the highest possible accuracy is achieved.

3.2. Predictive method and parameter calculation

The goal of the application is to detect if a website has malicious codes or not. However, with the above model, the output is only clusters of data points isolated by the K-means algorithm and no other information. Therefore, there is no basis to know whether or not malicious code is present on the website. To solve this problem, we propose the following predictive support method for the model:

First, let d(Cx,Cy) be the difference between two clusters Cx and Cy, this difference is quantified by the distance between the two closest points between Cx and Cy in the feature space, a point in Cx and a point on Cy. Calling dp a data point belonging to a cluster, we have:

$$d(C_x, C_y) = \min\left(distance(dp_x, dp_y)\right) \qquad \forall dp_x \in C_x, \forall dp_y \in C_y \qquad (4)$$

When the clustering model has high precision, this measure accurately reflects the difference between the clusters. Based on experimental data on the dataset, we determine the smallest difference between the two clusters containing benign data and containing malicious codes. This difference was taken as the threshold value to support predictive analysis.

$$Thr = min\left(d\left(C_{benign}, C_{mal}\right)\right)\ \forall C_{benign}, C_{mal} \in D \qquad (5)$$

The Thr in (5) is the threshold and D is the set of output clusters of the K-means algorithm.

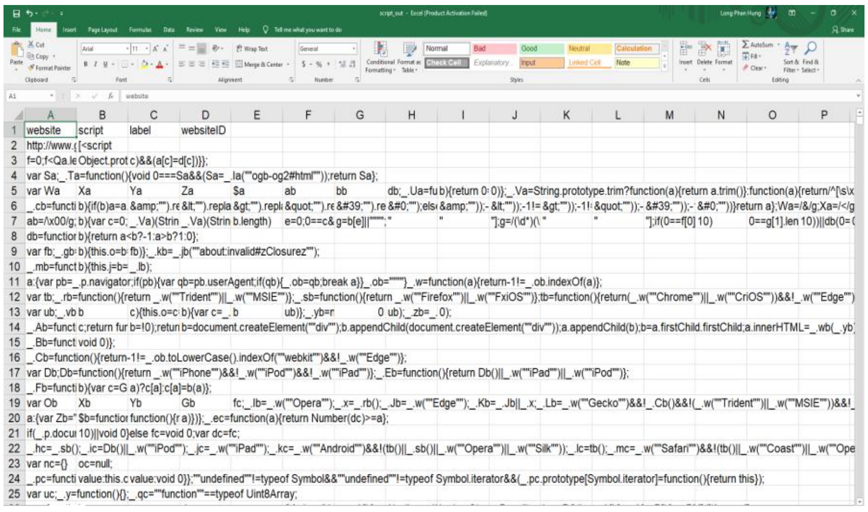
When the difference between the benign data cluster and any cluster is less than the threshold, the prediction result shows that there is no malicious code and vice versa, there is malicious code.

$$\exists\, X, Y \in D:\ d(X,Y) > Thr\ \rightarrow malicious\ JavaScript\ is\ on \qquad (6)$$

If the type of malicious code is known in advance, it is possible to predict the type of malicious code appearing on the web, based on the unique difference calculation. Especially if the website contains a new type of unknown malicious code, this method can also detect it. Thus, the method of detecting malicious code based on an unsupervised learning algorithm model has a way of fully specifying normal cases in advance to eliminate (detection) anomalies when they occur. Therefore, the precision of the model will depend on the normal website data, the more normal website data there is to train the model, the higher the accuracy of the malicious JavaScript detection model.

4. Experiments and Results

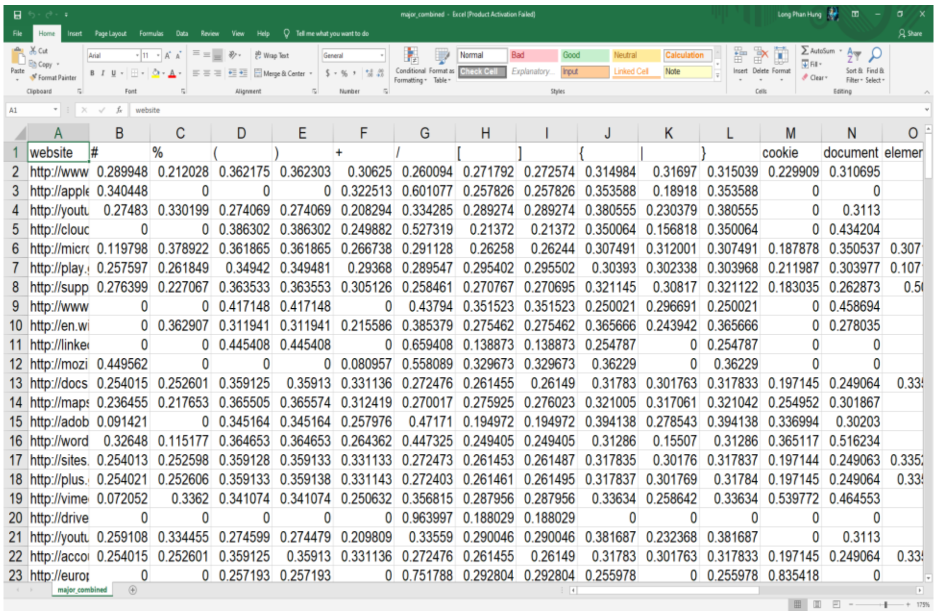
To build the dataset, we write a program to collect data from the website using Python language and Selenium library. The program will access the Phistank website [17] to get the addresses of malicious websites and visit the Moz website [18] to get the addresses of normal websites. The collected addresses are saved in two different files in preparation for Javascript code extraction. Websites with duplicate addresses will be removed. Next, the program will access the website addresses saved in the two files mentioned above to extract the Javascript code directly from these websites. The process of extracting malicious and non-malicious JavaScript is conducted separately and then saved together in a .csv file, named script.csv. The script.csv contains the website name, Javascript codes, malicious and non-malicious code labels, and the website's ID, as shown in Figure 2. The extracted Javascript codes include the types described in Table 1.



```
   A          B          C          D          E          F          G          H          I          J          K          L          M          N          O          P
1  website    script     label      websiteID
2  http://www.([<script
3  f=0;f<Qa.le Object.prot c)&&(a[c]=d[c])}};
4  var Sa;_.Ta=function(){void 0===Sa&&(Sa=_.Ia(""ogb-og2#html""));return Sa};
5  var Wa      Xa         Ya         Za         $a         ab         bb         db;_.Ua=fu b){return 0: 0)};_.Va=String.prototype.trim?function(a){return a.trim()}:function(a){return/^[\s\x
6  _.cb=functi b){if(b)a=a. &amp;"").re &lt;"").repla &gt;"").repl &quot;"").re &#39;"").re &#0;"");els &amp;""));- &lt;""));-1!= &gt;""));-1! &quot;""));- &#39;""));- &#0;""))}return a};Wa=/&/g;Xa=/</g
7  ab=/\x00/g; b){var c=0; _.Va)(Strin _.Va)(Strin b.length)  e=0;0==c& g=b[e]||""""; "     "          "];g=/(\d*)(\ "         "          "];if(0==f[0] 10)           0==g[1].len 10))||db(0=(
8  db=functior b){return a<b?-1:a>b?1:0};
9  var fb;_.gb b){this.o=b fb)};_.kb=_.jb(""about:invalid#zClosurez"");
10 _.mb=funct b){this.j=b=_.lb);
11 a:{var pb=_.p.navigator;if(pb){var qb=pb.userAgent;if(qb){_.ob=qb;break a}}_.ob=""""}_.w=function(a){return-1!=_.ob.indexOf(a)};
12 var tb;_.rb=function(){return _.w(""Trident"")||_.w(""MSIE"")};_.sb=function(){return _.w(""Firefox"")||_.w(""FxiOS"")};tb=function(){return(_.w(""Chrome"")||_.w(""CriOS""))&&!_.w(""Edge"")
13 var ub;_.vb b          c){this.o=c b){var c=_.b                ub)};_.yb=r                0 ub);_.zb=_.0);
14 _.Ab=funct c;return fur b=!0);retur b=document.createElement(""div"");b.appendChild(document.createElement(""div""));a.appendChild(b);b=a.firstChild.firstChild;a.innerHTML=_.wb(_.yb
15 _.Bb=funct void 0)};
16 _.Cb=function(){return-1!=_.ob.toLowerCase().indexOf(""webkit"")&&!_.w(""Edge"")};
17 var Db;Db=function(){return _.w(""iPhone"")&&!_.w(""iPod"")&&!_.w(""iPad"")};_.Eb=function(){return Db()||_.w(""iPad"")||_.w(""iPod"")};
18 _.Fb=functi b){var c=G a)?c[a]:c[a]=b(a)};
19 var Ob      Xb         Yb         Gb         fc;_.Ib=_.w(""Opera"");_.x=_.rb();_.Jb=_.w(""Edge"");_.Kb=_.Jb||_.x;_.Lb=_.w(""Gecko"")&&!_.Cb()&&!(_.w(""Trident"")||_.w(""MSIE""))&&!_
20 a:{var Zb=" $b=functior function(){r a)}}};_.ec=function(a){return Number(dc)>=a};
21 if(_.p.docu 10)||void 0}else fc=void 0;var dc=fc;
22 _.hc=_.sb();_.ic=Db()||_.w(""iPod"");_.jc=_.w(""iPad"");_.kc=_.w(""Android"")&&!(tb()||_.sb()||_.w(""Opera"")||_.w(""Silk""));_.lc=tb();_.mc=_.w(""Safari"")&&!(tb()||_.w(""Coast"")||_.w(""Ope
23 var nc={}   oc=null;
24 _.pc=functi value:this.c value:void 0}};""undefined""!=typeof Symbol&&""undefined""!=typeof Symbol.iterator&&(_.pc.prototype[Symbol.iterator]=function(){return this});
25 var uc;_.y=function(){};_.qc=""function""==typeof Uint8Array;
```

Figure 2. The content in script.csv file

Next, according to the method presented in Section 3, we write a program to calculate the TF-IDF for the features of each data point in script.csv. Since TF-IDF only allows TF-IDF calculation of features in character or word form, the features will be divided into two types: characters will be calculated TF-IDF in TF-IDF_01.csv file, words will be calculated in the TF-IDF_02.csv file and will be merged into the combined.csv file based on the website ID from script.csv. Figure 3 is the TF-IDF result of the combined.csv file, which is also the final output of the dataset preparation phase. It includes more than 2000 benign and malicious data points. However, we will extract smaller datasets for experiments and verify the effectiveness of the proposed method with the small datasets. As mentioned above, we will conduct feature extraction by the Sequential Feature Selector method before clustering.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	website	#	%	(	)	+	/	[	]	{	|	}	cookie	document	elemer
2	http://www	0.289948	0.212028	0.362175	0.362303	0.30625	0.260094	0.271792	0.272574	0.314984	0.31697	0.315039	0.229909	0.310695	
3	http://apple	0.340448	0	0	0	0.322513	0.601077	0.257826	0.257826	0.353588	0.18918	0.353588	0	0	
4	http://youtu	0.27483	0.330199	0.274069	0.274069	0.208294	0.334285	0.289274	0.289274	0.380555	0.230379	0.380555	0	0.3113	
5	http://clouc	0	0	0.386302	0.386302	0.249882	0.527319	0.21372	0.21372	0.350064	0.156818	0.350064	0	0.434204	
6	http://micrc	0.119798	0.378922	0.361865	0.361865	0.266738	0.291128	0.26258	0.26244	0.307491	0.312001	0.307491	0.187878	0.350537	0.307
7	http://play.ç	0.257597	0.261849	0.34942	0.349481	0.29368	0.289547	0.295402	0.295502	0.30393	0.302338	0.303968	0.211987	0.303977	0.107
8	http://supp	0.276399	0.227067	0.363533	0.363553	0.305126	0.258461	0.270767	0.270695	0.321145	0.30817	0.321122	0.183035	0.262873	0.5
9	http://www	0	0	0.417148	0.417148	0	0.43794	0.351523	0.351523	0.250021	0.296691	0.250021	0	0.458694	
10	http://en.wi	0	0.362907	0.311941	0.311941	0.215586	0.385379	0.275462	0.275462	0.365666	0.243942	0.365666	0	0.278035	
11	http://linke	0	0	0.445408	0.445408	0	0.659408	0.138873	0.138873	0.254787	0	0.254787	0	0	
12	http://mozi	0.449562	0	0	0	0.080957	0.558089	0.329673	0.329673	0.36229	0	0.36229	0	0	
13	http://docs	0.254015	0.252601	0.359125	0.35913	0.331136	0.272476	0.261455	0.26149	0.31783	0.301763	0.317833	0.197145	0.249064	0.33
14	http://maps	0.236455	0.217653	0.365505	0.365574	0.312419	0.270017	0.275925	0.276023	0.321005	0.317061	0.321042	0.254952	0.301867	
15	http://adob	0.091421	0	0.345164	0.345164	0.257976	0.47171	0.194972	0.194972	0.394138	0.278543	0.394138	0.336994	0.30203	
16	http://word	0.32648	0.115177	0.364653	0.364653	0.264362	0.447325	0.249405	0.249405	0.31286	0.15507	0.31286	0.365117	0.516234	
17	http://sites.	0.254013	0.252598	0.359128	0.359133	0.331133	0.272473	0.261453	0.261487	0.317835	0.30176	0.317837	0.197144	0.249063	0.335
18	http://plus.	0.254021	0.252606	0.359133	0.359138	0.331143	0.272403	0.261461	0.261495	0.317837	0.301769	0.31784	0.197145	0.249064	0.33
19	http://vime	0.072052	0.3362	0.341074	0.341074	0.250632	0.356815	0.287956	0.287956	0.33634	0.258642	0.33634	0.539772	0.464553	
20	http://drive	0	0	0	0	0	0.963997	0.188029	0.188029	0	0	0	0	0	
21	http://youtu	0.259108	0.334455	0.274599	0.274479	0.209809	0.33559	0.290046	0.290046	0.381687	0.232368	0.381687	0	0.3113	
22	http://acco	0.254015	0.252601	0.359125	0.35913	0.331136	0.272476	0.261455	0.26149	0.31783	0.301763	0.317833	0.197145	0.249064	0.33
23	http://euro	0	0	0.257193	0.257193	0	0.751788	0.292804	0.292804	0.255978	0	0.255978	0.835418	0	

Figure 3. The content of combined.csv containing TF-IDF value of the features

When setting the goal of detecting malicious codes using unsupervised learning algorithms, we pay attention to Isolation Forest because it is an unsupervised learning algorithm specializing in anomaly detection that has received a lot of attention recently, as in [6,9,15]. So we will also install and test malicious JavaScript detection with Isolation Forest and compare it with the method proposed here.

As analyzed in section 3 when proposing the method, the proportion of malicious data in the dataset to be tested will influence the detection efficiency. Therefore, we will conduct experiments with different numbers of malicious code to test the effectiveness and difference between cases. For each case, a corresponding threshold will be determined from the experiment. The program will conduct a check by calculating the difference between groups, and malicious code groups will be detected when the difference is greater than the threshold.

First, we will experiment with the training set with more than 2000 data points containing 10% malicious code to determine the threshold. Then, we will perform the test process with many different shuffled test sets and each test set is tested many times, the test sets are of the same size as the training set. Simultaneously, we also applied the Isolation Forest detection method on the same data set for comparison. The results show that with the proposed method, the detection threshold is 0.2881 and the accuracy is 98%, while the accuracy of the Isolation Forest method is only 92%, as presented in table 2.

Table 2. Results of the experiment with 10% malicious code

Method	**Threshold**	**Accuracy**
Proposed method	0.2881	98%
Isolation Forest		92%

Next, we increase the percentage of malicious code in the training set to 15% and determine a threshold of 0.2961. Performing the test in the same way as above gives an accuracy of 97%. Also conducting the test according to the Isolation Forest method has much lower results, only 88%, as presented in table 3.

Table 3. Results of the experiment with 15% malicious code

Method	**Threshold**	**Accuracy**
Proposed method	0.2916	97%
Isolation Forest		88%

In the same way, we increase the percentage of malicious code in the training set to 20% and determine the threshold is 0.2933. The test results give an almost constant precision of 97% while the precision of the Isolation Forest method is also low at 86%, as presented in table 4, due to detecting more malicious code in the test data.

Table 4. Results of the experiment with 20% malicious code

Method	**Threshold**	**Accuracy**
Proposed method	0.2933	97%
Isolation Forest		86%

However, when continuing to increase the proportion of malicious code in the training set to higher than in the training set, the threshold value increases as shown in Figure 4, and the accuracy decreases. Especially, when the proportion is increased to 40%, the threshold value is 0.31 and the accuracy drops to 87%. The accuracy of the Isolation Forest method is also reduced by 72%. The change in accuracy with the proportion of malicious code in the training data set is shown in Figure 5.

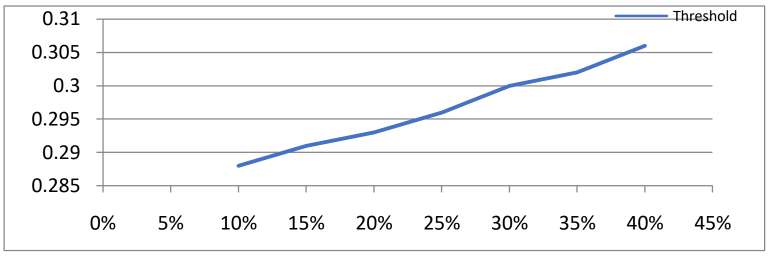


Figure 4. Threshold values increase as the increase in the proportion of malicious code

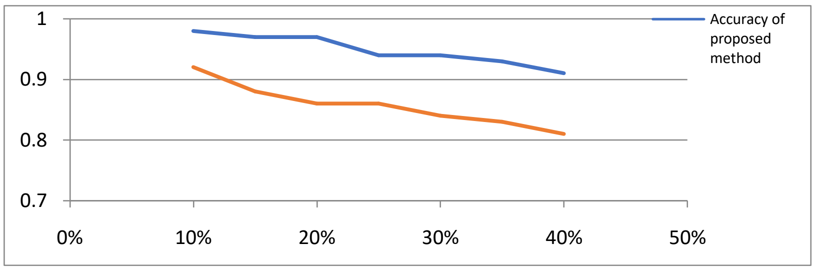


Figure 5. The accuracy decreases as the increase in the proportion of malicious code

The threshold, which is used to find malicious code, is a configuration parameter of the malicious code detection system. This parameter depends on the number of malicious codes in the training data set. As the number of malicious increases, this parameter also tends to increase and the accuracy of the model decreases. To test the effect of the configuration parameter, we reduce its value below the learned threshold value, especially in the case of a large proportion of malicious code in the dataset, as depicted in Figure 6. The results show that it is possible to improve the accuracy of the model as depicted in Figure 7. Therefore, it is recommendable to set the threshold value in the program lower than the value learned in the case of a dataset with a high proportion of malicious code.

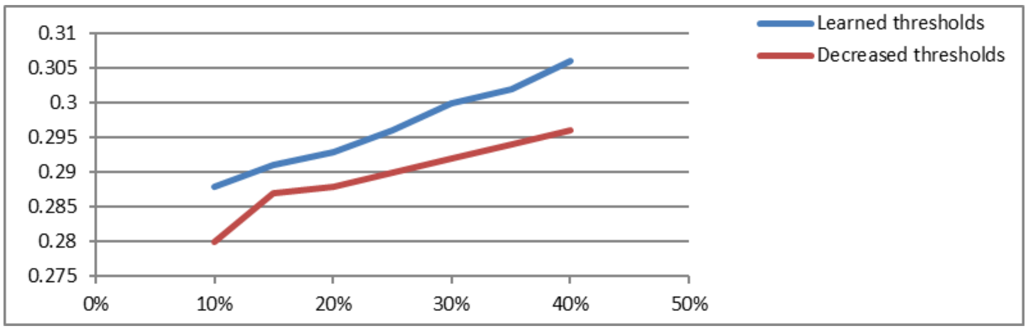


Figure 6. The threshold value is adjusted lower than the learned value

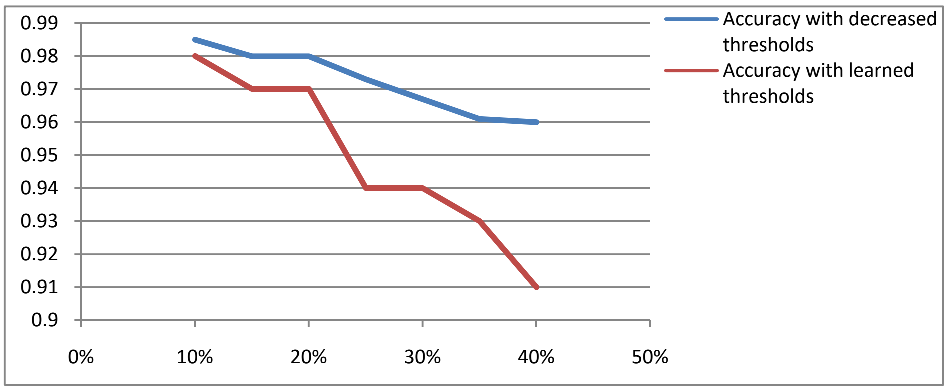


Figure 7. The accuracy is improved due to reducing the threshold value.

The results from the above experiments show that the proposed method achieves higher accuracy than the method using Isolation Forest. Especially, when the percentage of malicious code in the dataset is at 10% to 20%, the accuracy is from 97% to 98%. As the rate of malicious code increases, the accuracy tends to decrease. However, the experiment also shows that in the case of a high malware rate, we can reduce the threshold value to improve accuracy.

5. CONCLUSIONS

The malicious JavaScript detection method based on an unsupervised machine learning model has been presented. The key part of the method is the model with data preprocessing components, feature extraction, clustering by K-means algorithm, and cluster processing. The model is designed to be installed as a malicious JavaScript detection program running on the user's computers. The method is not only simple and low cost but also detects unknown malicious JavaScript codes. Experimental results show that the proposed method achieves higher accuracy than the method using the Isolation Forest algorithm. Especially, when the training dataset has a low proportion of malicious codes, the accuracy is higher. When the percentage of benign data is high, the ability to distinguish malicious codes is increased, and the value of the learned threshold parameter is correspondingly small. The threshold parameter also serves as a configuration parameter and can be downregulated in some cases to increase the accuracy of the model.

REFERENCES

[1] Binita Bohara, Jay Bhuyan, Fan Wu, and Junhua Ding, A Survey on The Use of Data Clustering for Intrusion Detection System in Cybersecurity, International Journal of Network Security & Its Applications (IJNSA) Vol. 12, No.1, January 2020.

[2] Dharmaraj R. Patil and J. B. Patil, (2017) Detection of Malicious JavaScript Code in Web Pages, Indian Journal of Science and Technology, Vol 10(19), DOI: 10.17485/ijst/2017/v10i19/114828, May 2017

[3] F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.

[4] Guo J., Cao Q., Zhao R., Li Z. (2020) Improving Detection Accuracy for Malicious JavaScript Using GAN. In: Bielikova M., Mikkonen T., Pautasso C. (eds) Web Engineering. ICWE 2020. Lecture Notes in Computer Science, vol 12128. Springer, Cham. https://doi.org/10.1007/978-3-030-50578-3_12

[5] Guo Z. (2021) Malicious JavaScript Detection Method Based on Multilayer Perceptron. In: MacIntyre J., Zhao J., Ma X. (eds) The 2020 International Conference on Machine Learning and Big Data Analytics for IoT Security and Privacy. SPIOT 2020. Advances in Intelligent Systems and Computing, vol 1282. Springer, Cham. https://doi.org/10.1007/978-3-030-62743-0_74

[6] Heigl, M.; Anand, K.A.; Urmann, A.; Fiala, D.; Schramm, M.; Hable, R. On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data. Electronics 2021, 10, 1534. https://doi.org/10.3390/electronics10131534

[7] Lloyd, S. P. (1957). Least squares quantization in PCM. Technical Report RR-5497, Bell Lab, September 1957.

[8] Michal Kedziora, Paulina Gawin, Michal Szczepanik and Ireneusz Jozwiak, Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code, International Journal of Network Security & Its Applications (IJNSA) Vol. 11, No.1, January 2019.

[9] Ounacer, S., Bour, H.A., Oubrahim, Y., Ghoumari, M.Y., & Azzouazi, M. (2018). Using isolation forest in anomaly detection: The case of credit card transactions. Periodicals of Engineering and Natural Sciences (PEN), 6, 394-400.

[10] Prabhu Seshagiri, Anu Vazhayil, Padmamala Sriram, AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts, Procedia Computer Science,Volume 93,2016,Pages 768-773,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2016.07.291. (https://www.sciencedirect.com/science/article/pii/S187705091631537X)

[11] Samuel Ndichu, S. Ozawa, T. Misu and K. Okada, "A Machine Learning Approach to Malicious JavaScript Detection using Fixed Length Vector Representation," 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489414.

[12] Samuel Ndichu, Sangwook Kim, Seiichi Ozawa, Takeshi Misu, Kazuo Makishima, A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors,

Applied Soft Computing, Volume 84, 2019, 105721, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2019.105721. (https://www.sciencedirect.com/science/article/pii/S1568494619305022)
[13] TF-IDF. In: Sammut C., Webb G.I. (eds) (2011) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_832
[14] Wang, Y., Cai, W., and Wei, P. (2016) A deep learning approach for detecting malicious JavaScript code. Security Comm. Networks, 9: 1520- 1534. doi: 10.1002/sec.1441.
[15] Waspada, I., Bahtiar, N., Wirawan, P.W., & Awan, B.D. (2020). Performance Analysis of Isolation Forest Algorithm in Fraud Detection of Credit Card Transactions.
[16] Xuyan Song , Chen Chen, Baojiang Cui, and Junsong Fu, (2020) Malicious JavaScript Detection Based on Bidirectional LSTM Model, Appl. Sci. 2020, 10, 3440; doi:10.3390/app10103440.
[17] https://www.phishtank.com/
[18] https://moz.com/top500

AUTHORS

Nguyen Hong Son received his B.Sc. in Computer Engineering from The University of Technology in HCM city, his M.Sc. and Ph.D. in Communication Engineering from the Post and Telecommunication Institute of Technology Hanoi. His current research interests include communication engineering, machine learning, data science, network security, and cloud computing.

Ha Thanh Dung, received his B.Sc in Information Technology from VNU Hanoi-University of Science, and his M.Sc in Data Transmission and Computer Networks from Post and Telecommunication Institute of Technology in 2012. His research areas are communication engineering, information systems, machine learning, and network security.