International Journal of Security and Its Applications
Vol. 9, No. 6 (2015), pp. 251-262
http://dx.doi.org/10.14257/ijsia.2015.9.6.24

A Half-Dynamic Classification Method on Obfuscated Malicious JavaScript Detection

Zhaolin Fang[1,2], Renhuan Zhu[3], Weihui Zhang[2] and Bo Chen[1]

[1]*College of Computer Sci. & Tech., Zhejiang University of Technology, Hangzhou, 310023, China*
[2]*Network Information Center, Zhejiang University of Technology, Hangzhou, 310014, China*
[3]*China United Telecommunications Co. Ltd., Zhejiang Branch, Hangzhou, 310000, China*
fzl@zjut.edu.cn

Abstract

The traditional static detection method for malicious JavaScript detection has high efficiency without the need of code executing, but it cannot detect new malicious script. While the dynamic method usually needs to execute code and extract features, which lead to low efficiency and highly difficulty. In this paper, we propose a half-dynamic detection method for classification, which can solve the problem of obfuscated malicious JavaScript. The proposed method starts with obtaining the intermediate-state machine code using the JavaScript interpreter to compile the JavaScript. After extracting the function calling sequence of machine code, the feature model of the sequence is built using N-gram. Then we use k-NN classifier for training and detecting the malicious script. N-gram can directly be used to statically analyze the sequence of the obfuscated JavaScript, but not available to recognize the maliciousness. Then N-gram on the call function sequence of the compiled machine code is proposed as an efficient half-dynamic malicious script detection method. Finally, the efficiency and effectiveness of the proposed method is demonstrated through the experiments.

Keywords: *Malicious JavaScript, Code Obfuscation, N-gram, k-NN*

1. Introduction

Malicious code detection has been an important part of information security, and people have made a lot of achievements. Malicious code detection can be divided to static detection and dynamic detection according to the objects. The static detection analyses text feature while the dynamic detection analyses the behavior of code execution.

The typical method of the static detection is the signature detection, which creates a database of for the unique signature of all the known malicious code based on the thought of pattern matching [1]. These signatures are extracted by experts after analysing virus samples, and each signature can only note one malicious code. The method based on signature detection can be achieved as follows: (1) Collecting samples of malicious code; (2) Extracting signature of malicious code from these samples; (3) Bring signature into database; (4) Detecting files. If the file contains the signature in the database, it must be malicious code or infected by malicious code.

The typical method of the dynamic detection is the behavior detection, which detects virus according to their peculiar behavior by dynamic code execution or virtual code execution [2]. After studying the virus for many years, one can find the common behavior of malicious code, which is so particular that cannot be found in normal code. Generally,

ISSN: 1738-9976 IJSIA
Copyright ⓒ 2015 SERSC

dynamic detection needs dynamic code execution, which is very inefficient and pretty hard because of the fact that we have to extract the code feature dynamically.

The static detection method cannot detect new malicious script, but has high efficiency. While the dynamic method can detect it with low efficiency and highly difficult to extract features. So the researcher focus on how to automatically detect new malicious code with efficiency, and the auto-classifier has been a central point in the domain of anti-virus. Schultz first utilize the technology of the data mining to detect the malicious code, and get good performance [3]. Nowadays more and more researchers has centralized on the data mining and machine learning, and done many meaningful attempts. Schultz first utilize the technology of the data mining to detect the malicious code, which get good performance. They use various classification algorithm according to the character of different kinds of virus, including Signature Methods [4], RIPPER [5], Naïve Bayes and Multi-Naïve Bayes [6], then detect features like the call of win32 dll, ASCII, bytes yards sequence [7]. Huang (2002) analyzed the machine code of the virus sample, and use active bayesclassifier to detect the malicious executable code [8]. JauHwan [9] investigated the automating the detection of malicious javascript using the algorithm of Decision Tree [10] and Naïve Bayes. Abou-Assaleh extracted the CNG(Common N-gram) [11] from the byte sequence as features, and use KNN(K-nearest neighborhood)[12] as classifier to detect the unknown virus effectively [13]. Kolter extracted the N-gram from byte sequence of malicious code as a feature set, and then select the most relevant features using feature gain. Classification methods like Instance-based Learner[14],Naive Bayes, Support Vector Machine [15], Decision Tree, Boosting[16] are used to detect the malicious execute code [17].Michael investigated the malicious script detection with machine learning. They treated the byte sequence of the benign script as the main one-class sample and the malicious script as the outlier samples, and used the one-class SVM to classification[18].Reddy used variable N-gram instead of fixed-length N-gram as classification feature。The result showed that variable N-gram can obtain better detection accuracy and lower false positive rate [19].Wang proposed a virus detection based on the API series analysis and SVM, using the Win API called by the PE as features, applying the SVM to classify the unknown virus, which is of good recognition performance [20]. Konrad investigated the behavior pattern of malicious script in sandbox, and extracted the corpus of malicious behavior to recognize new malicious software [21]. Moskovitch [22] discussed the detection performance of malicious code by extracting the command code and N-gram feature and using classification algorithm like Artificial Neural Networks [23],Decision Trees, Naïve Bayes. Liu designed a malicious code detection method based comprehensive behavior characteristics of code, describing the relationship between relevant subjects in malicious code execution step by modified attack tree model. In this way the vicious weight can reflect the impact on system during code execution more precisely [24]. Likarish argued that combining the key words of JavaScript with script readability and the amount of space as the feature set for classification can acquire good performance [25]. Zhang used the feature selection based weighted information gain to get high detection rate and accuracy, which utilized the N-gram and variable-step N-gram of binary code to select feature and multiple classifier[26].Choi applied N-gram, entropy, character size as testing index, and a implement had been made to detect the credibility of web script [27]. Santos analyzed operation code sequence for constructing a vector for a short code to distinguish malicious and benign behavior, which can recognize unknown malicious software and its transformation [28].

It has achieved a lot research result that applying the machine learning to malicious detection domain, while most research are focused on the executable file in Windows system and the web JavaScript is lack of deep research. The code obfuscation techniques analysis need to be utilized in the malicious detection, as its extensive use in JavaScript. So there are at least two points worth research in malicious script detection based machine

 Copyright ⓒ 2015 SERSC

learning. One is to utilize the machine learning to script obfuscation recognition by distinguishing the feature of obfuscation and non-obfuscation script with multiple classification method. The other is to combine the static and dynamic detection method to design quick and efficient malicious script detection method.

2. Static JavaScript Obfuscation Classification Using N-gram and K-NN

2.1. N-gram Method

N-gram collects a series of overlapping sub-string with a sliding window of length N, which slides a unit length every time. For example, the binary string (101101110110110110110110), where every four bits represent a character, can be divided into 2-gram as (10110111),(01110110),(011011011),(11011011),(10110110).

Let fixed-length N sliding window slide linear bitstream on the target dataset, and counts the occurrences of every gram to get the frequency distribution of N-gram. Then, the consistency of different data streams can be divided based on the distribution characteristics of the N-gram. And the computational complexity of N-gram is increased with the length of the sliding window exponentially. For a JavaScript of length L, without considering the punctuation characters and other features, its N-gram information item is $(L \times (L + 1) / 2)$. Thus, script contains rich N-gram information item.

In these information items, not all are usefull for classification. An N-gram information item can be measured with three aspects: frequency, degree of dispersion and concentration.

Frequency: In document d, the N-gram frequency tf can be represented by its appears in d.

Dispersity: In document c, the dispersity df of N-gram information items can be represented by the number of documents. The items of information become more scattered in c while df increases; on the contrary, become less dispersed.

Concentration: In the document set D, the number of documents cf presents the concentration cf of N-gram information items. And cf smaller, the items of information in D become more concentrated; on the contrary, become less concentrated.

When N-gram extract information items, the higher the frequency, the greater the degree of dispersion, and the greater the concentration, makes greater impact on the classification results, which can obtain optimum classification results.

N-gram is good at handling small documents, it just need to linear scan the document to complete the N-gram information items statistics. But to large training set, it takes larger memory space. To this problem, an extraction step by step method is used. The basic idea is: at first, it extracts 1-gram information item of the document; then constructs candidate 2-gram information items from 1-gram, excluding the ones which does not meet the conditions to get 2-gram information items really needs. So obtained 3-gram, 4-gram, 5-gram.

2.2. *K*-NN Classification Method

K- nearest neighbor (K-nearest neighborhood, KNN), a theoretically mature method, which is one of the most simple machine learning algorithm, is widely used in the text categorization. The basic idea of the algorithm is: Given the new text, it chooses K text articles which are the nearest (most similar) to the new text in the training set, according to the category the K article text belongs to predicate the category of new text.

Copyright ⓒ 2015 SERSC

In KNN document classification method, all documents are represented by the vector space model. One vector space vector presents one document, this vector is also called document vectors. Each dimension of a vector is corresponds to each item of information in a document, which is the document properties. For a specific document, the value of each dimension is correspond to the frequency of this word in the document.

For document library D, assuming corresponding document properties set are *V*, $V = \{W_i\}, (i = 1, 2, ..., n)$. An existing document d, the vector model is expressed as:

$$\vec{d} = (w_1, w_2, ..., w_n) \quad (1)$$

Above, $w_i (i = 1 \sim n)$ presents corresponding weight value. TFIDF estimation method is usually used.

$$w_i = \frac{tf_i \times \log(N / n_i)}{\sqrt{\sum_{i=1}^{n} (tf_i)^2 [\log(N / n_i)]^2}} \quad (2)$$

N is the total number of training documents; tf_i represents the frequency w_i in the training document d; n_i represents the number of document contains w_i. Document property is calculated with equation (2), which makes the document vector equation (1) a unit vector. Thus, the similarity between the document d_i and d_j can be presented in cosine formula as:

$$Sim(d_i, d_j) = \vec{d_i} \cdot \vec{d_j} \quad (3)$$

KNN method for document classification process is as follows: To a given test document d, finds the k most similar training documents by based on similarity. On this basis, score each document class, this score presents the sum of similarity between these documents belongs to the class and test document. That is, in these K documents, there are multiple documents belong to a class, the class score is the sum of the similarity between these documents and test documents.

Sort the scores after above process. A threshold should be chosen, and only these whose score exceeds the threshold should be considered. Test documents belongs to the class whose score exceed the threshold. Formalized as:

$$score(\vec{d}, c_i) = \sum_{\vec{d} \in kN} Sim(\vec{d}, \vec{d_j}) y(\vec{d_j}, c_i) - b_i \quad (4)$$

$y(\vec{d_j}, c_i) = \{^{1 \quad \vec{d_j} \in c_i}_{0 \quad \vec{d_j} \in \overline{c_i}}$; b_i is the threshold; $score(\vec{d}, c_i)$ is the score of test document d belongs to class c_i.

To some specific class, bi should be optimal selected, and can be adjusted by a validation documentation set. Validation documentation set is part of the training documentation set. According to the results of the formula (4), the category of the test document can be determined. Obviously, for each test document, it must calculate the similarity with all other documents in a document library. Therefore, the time complexity of the KNN method is O (M, N) (M and N, respectively, the total number of documents for training and testing documents).

2.3. Direct Detection on JavaScript Code Obfuscation

2.3.1. JavaScript Obfuscation: With the development of WEB technology, HTML pages can nolonger satisfy the needs of page interaction. Therefore JavaScript is increasingly applied to web design. JavaScript implements a real-time, dynamic, interactive

 Copyright ⓒ 2015 SERSC

relationship between pages and users. But JavaScript also makes it easier for hackers to write and run dangerous code, and some malicious JavaScript use obfuscation techniques to hide their features, in order to avoid the detection of rule-based regular and expression-based anti-virus software.

Malicious code obfuscation technology refers to some kind of program code conversion, by morphing technology, while ensuring the execution of the script function unchanged, achieve to evade recognition detection. Script obfuscated technologies usually used are compression, alternative, restructuring, redundancy interference and encryption. With fuzzy conversion technology, a malicious code can be transformed into a number of different variants script through different confusing process. So signature-based scan detection tool can not effectively identify them.

2.3.2. Classification Methodology: This project collected a lot of obfuscated JavaScript scripts, analyzed its obfuscation principle and obfuscated scripts features, with N-gram method to extract the feature vectors of the script and the KNN classification method, to build an intelligent recognition system to detect obfuscated JavaScript. System design is shown in Figure 1, 2.

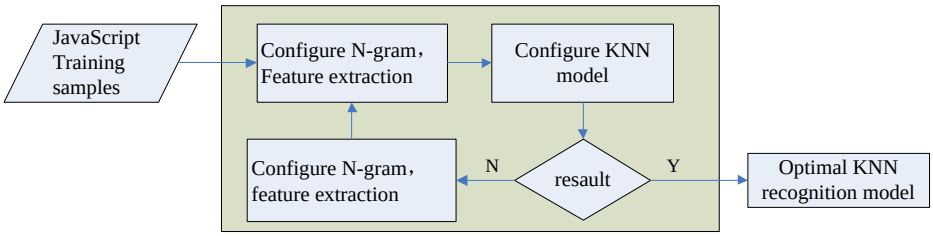

Figure 1. Flow Chart of Training the Obfuscated JavaScripts

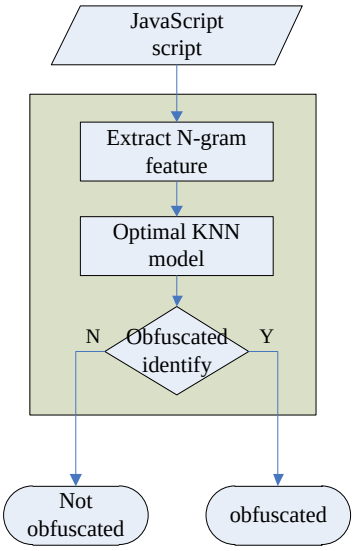

Figure 2. Flow Chart of the Obfuscated Identify Scripts

1) feature selection

To the obfuscated JavaScript, we propose to extract the eigenvectors of obfuscated and not obfuscated JavaScript with N-gram document classification method. The N-gram sliding window size is expressed as N, N ranges from 2 to 8. The dimension of extract features is too high, when N> 3, the characteristic dimension reach 10,000 or more, due to the N-gram features extracted increased exponentially with N, and a large part of frequency value is 0 or close to 0, so for the extracted feature, sorted according to the

Copyright ⓒ 2015 SERSC

frequency, the TopN frequency eigenvalues were taken experiment training, TopN ranges from 100 to 800.

2) Classification

For N-gram feature vectors extracted, compute the similarity between the script to classify script, means the category of a script that is determined by the distance between the samples. We use KNN algorithm to extract the N-gram feature vectors for classification. The distance is defined as follows:

$$dist = \sum_{k=0}^{TopN} ((f_k - f_{test\_k}) / (f_k + f_{test\_k}))^2 \qquad (5)$$

f_k represents the values of Top N gram frequency in the classified scripts, $f_{test\ k}$ represents the values of Top N gram frequency in the test scripts, $0< k < TopN$,

Feature Selection Process:
Input: training scripts *train_files*
Output: N-gram feature vector *ngram_files*
For *N*=2 to 8
For each *train_files*
While（scripts end）
read first N characters, add up frequency of its appearance
silde a character
End
End
End

KNN Classification：
Input： obfuscated N-gram profile *obsc_gram_file*
no obfuscated N-gram profile *no_obsc_gram_file*
test N-gram profile *test_gram_file*
Output： resault *res={obsc , no_obsc}*
For each *obsc_gram_file*
dist_new= $\sum_{k=0}^{TopN}((f_k - f_{test\_k})/(f_k + f_{test\_k}))^2$
dist1=MIN(*dist1*,*dist_new*)
End

For each *no_obsc_gram_file*
dist_new= $\sum_{k=0}^{TopN}((f_k - f_{test\_k})/(f_k + f_{test\_k}))^2$
dist2=MIN(*dist2*,*dist_new*)
End

If *dist1*<*dist2* Then
res=obsc
Else
res=no_obsc
End

2.3.3. Experimental Results: We collect 1500 benign not obfuscated JavaScript from Sourceforge, Github and Google code repository for experiment, then use obfuscation tool JsPacker, JsCompressor to process scripts, obtain 1500 obfuscated script. And extract N-gram feature extraction from 3000 samples, wherein the value of N is: N = 2,3 ... 8.

Table 1 shows top 20 frequency gram with different N after feature extraction.

 Copyright ⓒ 2015 SERSC

Table 1. Top 20 Frequency Gram with Different N After Feature Extraction

N=2	N=3	N=4	N=5	N=6	N=7	N=8
ON 286	HIS 180	THIS 180	THIS. 178	****** 126	******* 124	******** 122
TH 251	THI 180	HIS. 178	***** 128	CTION(76	FUNCTIO 70	FUNCTION 70
AR 243	IS. 178	**** 130	CTION 105	FUNCTI 70	NCTION(70	UNCTION(70
S. 240	*** 132	TION 110	MONGO 78	NCTION 70	UNCTION 70	=FUNCTIO 61
IS 206	ION 118	CTIO 105	TION(76	UNCTIO 70	=FUNCTI 61	.PROTOTY 42
ST 203	TIO 110	ION(78	FUNCT 70	=FUNCT 61	RETURN_ 50	OTOTYPE. 42
HI 181	CTI 105	MONG 78	NCTIO 70	RETURN 54	.PROTOT 42	PROTOTYP 42
TE 165	ON(94	ONGO 78	UNCTI 70	ETURN_ 50	OTOTYPE 42	ROTOTYPE 42
); 142	MON 78	VAR_ 72	=FUNC 61	MASTER 45	PROTOTY 42	HIS._CON 30
** 134	NGO 78	FUNC 70	ETURN 54	(THIS. 42	ROTOTYP 42	THIS._CO 30
ER 134	ONG 78	NCTI 70	RETUR 54	.PROTO 42	TOTYPE. 42	CONNECTI 26
RE 130	UNC 75	UNCT 70	TURN_ 50	;THIS. 42	HIS._CO 30	EST.PROT 26
NG 125	AR_ 72	=FUN 61	ASTER 45	OTOTYP 42	IS._CON 30	NNECTION 26
ET 121	VAR 72	NAME 60	MASTE 45	OTYPE. 42	THIS._C 30	ONNECTIO 26
CT 120	TER 71	ETUR 54	(THIS 42	PROTOT 42	CONNECT 29	ST.PROTO 26
){ 118	FUN 70	RETU 54	.PROT 42	ROTOTY 42	EST.PRO 26	T.PROTOT 26
IO 118	NCT 70	TURN 54	;THIS 42	TOTYPE 42	NECTION 26	TEST.PRO 26
TI 118	RET 68	PORT 52	OTOTY 42	.PUSH(39	NNECTIO 26	ARTMONGO 25
;} 107	ARG 64	URN_ 50	OTYPE 42	THIS._ 39	ONNECTI 26	STARTMON 25
.P 105	IF(62	ARGS 47	PROTO 42	ECTION 35	ST.PROT 26	TARTMONG 25

We use 3-fold cross validation method to experiment.1500 samples were divided into three groups, denoted as A1, A2, A3,1500 a obfuscated script also were divided into three groups, denoted as B1, B2, B3.Take one obfuscated group and another normal group as classification group, and the remaining as test samples, as shown in Table 2.

Table 2. Experimental Training Sample Grouping Results

Normal samples	Obfuscated samples	Test samples
A1	B2	A3、B3
A1	B3	A2、B2
A2	B1	A3、B3
A2	B3	A1、B1
A3	B1	A2、B2
A3	B2	A1、B1

In the experiment, we found that in the calculation of the shortest distance, the shortest distance always appears between several certain scripts, means there is a typical classification sample, is most vulnerable to become the comparison object as shortest distance. Based on these characteristics, we screened classified script, only retaining the scripts that has been identified as the shortest distance greater than a certain number of times. Thus, KNN algorithm's efficiency of finding the shortest distance can be greatly enhanced, the efficiency of the script from the original 1000 script every need relatively reduced to just compare 100 scripts, while little affects on the accuracy. False positive rate and false negative rate as shown in Figure 3, 4.

Copyright ⓒ 2015 SERSC

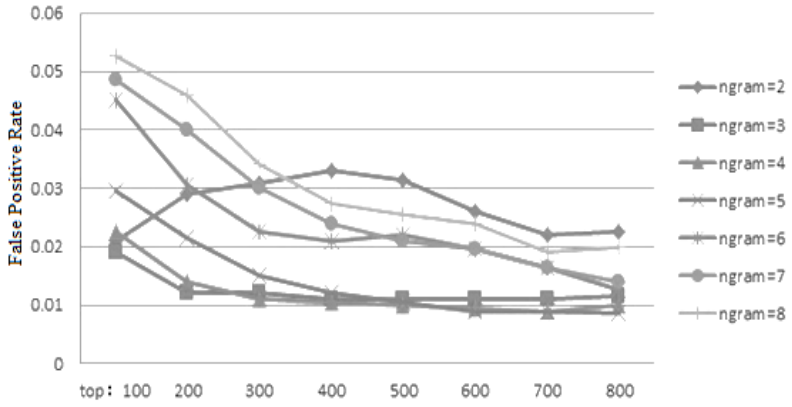

Figure 3. False Positive Rate of the Obfuscated Scripts

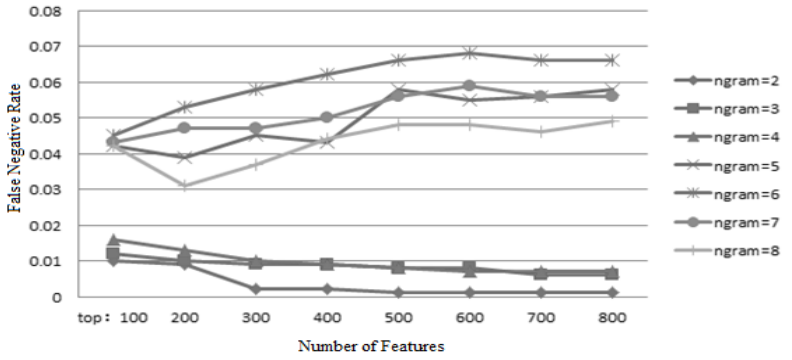

Figure 4. False Negative Rate of the Obfuscated Scripts

Through the analysis of experimental data, when N = 3, the number of features TopN = 500, the false positive rate and false negative rate of less than 1%, it can effectively identify obfuscated script. When N = 3, the number of features TopN = 500, the confusion matrix (Confusion Matrix) of obfuscated scripts as shown in Table 3.

Table 3. The Confusion Matrix of the Obfuscated Scripts

	Predicted as obfuscated	Predicted as benign	Total
Obfuscated	*TP*=992	*FN*=8	1000
Benign	*FP*=11	*TN*=989	1000
Total	1003	997	

3. Half-Dynamic Detection on Obfuscated Malicious JavaScript

3.1. JavaScript Machine code Generation

In order to extract effective features from the obfuscated scripts, this paper use the JavaScript engine to pretreat obfuscated scripts. The current mainstream JavaScript engines are Spider Monkey, Carakan, Chakra, Nitro, V8, etc., in which chrome browser using V8 JavaScript engine. V8 JavaScript compiled scripts into machine code before execution, the efficiency is very high, and the V8 is an open source project, so this paper uses V8 engine to deal with the obfuscated scripts, and extract the machine code.

Three key parts of V8 engine to quickly parse are fast property access, dynamic machine code generation and efficient garbage collection, this article focuses on the dynamic machine code generation. JavaScript is weakly typed language, dynamically

 Copyright ⓒ 2015 SERSC

identify data's type are needed to deal with variables. V8 uses dictionary-like structure to store properties of objects, and creates hidden class, then it can query from hidden class to identify the type of variables. V8 usually compiled JavaScript to native machine code directly at the first execution of JavaScript, rather than using the intermediate byte code, there is no need for interpreter, and the property access is accomplished by inline cache code, these codes are usually transformed by V8 to appropriate machine instructions while running.

The corresponding machine code generated after compiled by V8 is as follows

ebx = the point object

cmp [ebx, <hidden class offset>], <cached hidden class>

jne <inline cache miss>

mov eax, [ebx, <cached x offset>]

V8 engine can effectively improve the execution speed of JavaScript scripts, and its main application in this paper is dynamic machine code generation. When the traditional text feature analysis fails in obfuscated script, with the V8 engine, we will compile the obfuscated scripts into machine code, and then process N-gram feature extraction. The following is a fragment of V8 machine code generated. After Analyzed the machine code, we found the word appeared relatively stable, so we extract N-gram features with the word for the unit. Further analysis of machine code's text feature, found that each statement not only contain the operating code, but also includes a register address, memory address. And this code have little to do with the machine code excution, but make different N-gram features because of storage addresses various, and interferes with the classification results. So we extracted opcodes separately from the machine code, as shown in Figure 5.

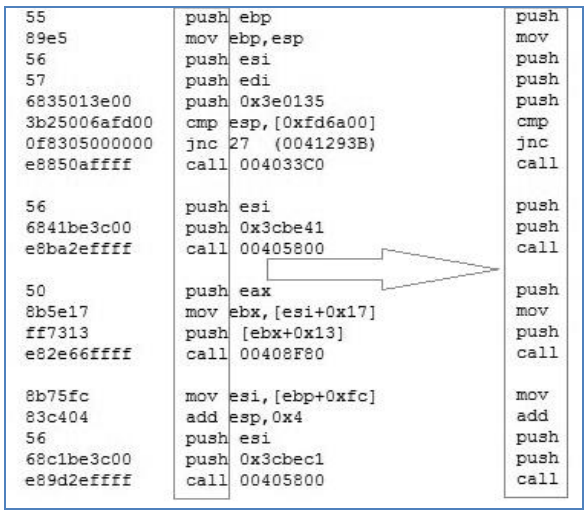

Figure 5. The Opcode Illustration in the Machine Code

3.2. Detection Methodology and Results

In this section, we collect a certain amount of obfuscated malicious script, use V8 engine to compile scripts into machine code, and then extract the feature set of the script with the N-gram method. Use KNN classification method to establish intelligent detection methods JavaScript code confusion script.

Copyright ⓒ 2015 SERSC

To the newly extracted operation code, we use N-gram feature extraction, classification and prediction with KNN. Experimental result false positive rate and false negative rate is as shown in Figure 6, 7.

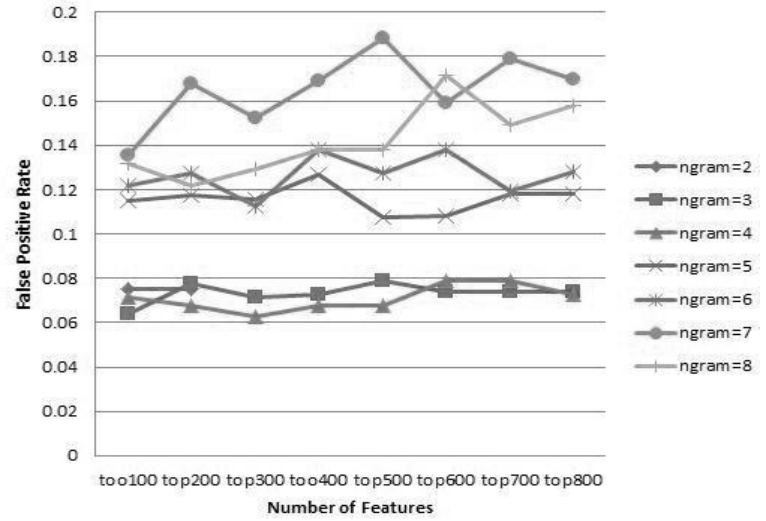

Figure 6. Experimental Results on False Positive Rate

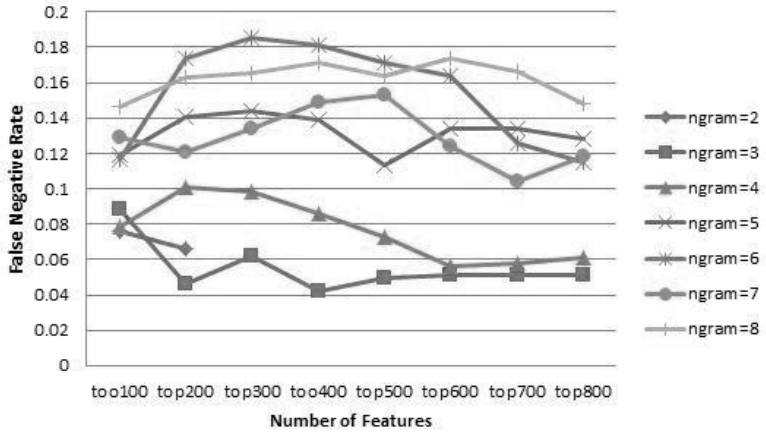

Figure 7. Experimental Results on False Negative Rate

Through the analysis of experimental data, extracts opcode classification performance is better than the whole machine code. When N = 3, TopN = 500, the false positive rate and false negative rate is 8% and 5%, respectively.

The confusion matrix of malicious scripts experiments results is shown in Table 4, the method proposed can effectively detect and identify malicious obfuscated scripts.

Table 4. Experiments Results on the Confusion Matrix of Malicious Scripts

	Predicted as malicious	Predicted as benign	Total
Malicious	TP=933	FN=67	1000
Benign	FP=23	TN=977	1000
Total	956	1044	2000

 Copyright ⓒ 2015 SERSC

4. Conclusions

In this paper, we propose a so called half-dynamic detection method for malicious JavaScript detection, supporting the obfuscation code. The proposed method uses the intermediate-state machine code by compiling the JavaScript using the script interpreter V8. After extracting the function calling sequence from V8 code of JavaScript, the feature model of these call sequences is built using N-gram, and k-NN classifier is then employed for training and detecting the malicious script. N-gram can handle the sequence of the obfuscated JavaScript, but cannot identify the maliciousness. Therefore, N-gram on the call function sequence of the compiled machine code is proposed as an efficient half-dynamic malicious script detection method. Experiments show that our half-dynamic detection method can achieve good results of false positive and false negative rate on detecting the maliciousness of obfuscated JavaScript.

Acknowledgements

This work is partially supported by Zhejiang National Science Foundation with grant No. LY12F02039, Y15F020101.

References

[1] Y. K. Penya Santos, J. Devesa and P. G. Bringas, "N-grams-based file signatures for malware detection", Proceedings of the 11th International Conference on Enterprise Information Systems, **(2009)**, pp. 317-320.
[2] J. Aycock, "Computer viruses and malware", Advances in Information Security, vol. 22, **(2006)**, pp. 253-259.
[3] M. G. Schultz, E. Eskin, E. Zadok and S. J. Stolfo, "Data mining methods for detection of new malicious executables", Proceedings of 2001 IEEE Symposium on Security and Privacy, **(2001)**, pp. 38.
[4] Y. T. Hou, Y. Chang and T. Chen, "Malicious web content detection by machine learning", Expert Systems with Applications, vol. 37, **(2010)**, pp. 55-60.
[5] W. W. Cohen, "Learning rules that classify e-mail", Papers from the AAAI Spring Symposium on Machine Learning in Information Access, **(1996)**, pp. 25.
[6] A. McCallum and K. A. Nigam, "Comparison of event models for naive bayes text classification", Learning for Text Categorization, **(1998)**, pp. 41-48.
[7] M. G. Schultz, E. Eskin, E. Zadok and S. J. Stolfo, "Data mining methods for detection of new malicious executables", Proceedings of 2001 IEEE Symposium on Security and Privacy, **(2001)**, pp. 38.
[8] G. Richards, S. Lebresne, B. Burg and J. Vitek, "An analysis of the dynamic behavior of JavaScript programs", Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, **(2010)**, pp. 1-12.
[9] J. H. Wang, P. S. Deng and Y. S. Fan, "Virus detection using data mining techniques", Proceedings of the 37th IEEE Annual International Carnahan Conference on Security Technology, **(2003)**, pp. 71-76.
[10] P. Horton and K. Nakai, "Better prediction of protein cellular localization sites with the k nearest neighbors classifier", Proceedings of International Conference on Intelligent Systems for Molecular Biology, vol. 5, **(1997)**, pp. 147-152.
[11] W. B. Cavnar, andM. Trenkle, "N-gram-based text categorization", Ann Arbor MI, vol. 48113, **(1994)**, pp. 4001.
[12] G. Parthasarathy and N. Chatterji, "A class of new KNN methods for low sample problems", IEEE Transactions on Systems, Man and Cybernetics, vol. 20, no. 3, **(1990)**, pp. 715-718.
[13] T. A. Assaleh, N. Cercone, V. Keselj and R. Sweidan, "Detection of new malicious code using n-grams signatures", Proceedings of the International Conference on Intelligent Information Systems, **(2004)**, pp. 193–196.
[14] J. G. Cleary and L. E. Trigg, "An instance-based learner using an entropic distance measure", Machine Learning-International Workshop, **(1995)**, pp. 108–114.
[15] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers", Neural processing letters. Kluwer Academic Publishers, vol. 9, no. 3, **(1999)**, pp. 293-300.
[16] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm", Proceedings of the Thirteenth International Conference on Machine Learning, **(1996)**, pp. 148-156.
[17] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild", The Journal of Machine Learning Research, vol. 7, **(2006)**, pp. 2721–2744.
[18] D. M. Cai, J. Theiler and M. Gokhale, "Detecting a malicious executable without prior knowledge of its patterns", Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005, vol. 5812, **(2005)**, pp. 1-12.

Copyright ⓒ 2015 SERSC

[19] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection", Journal in Computer Virology, vol. 2, no. 3, **(2006)**, pp. 231-239.
[20] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", International Joint Conference on Artificial Intelligence, vol. 14, **(1995)**, pp. 1137-1145.
[21] K. Rieck, T. Holz and C. Willems, "Learning and classification of malware behavior", Fifth conference on Detection of Intrusions and Malware, and Vulnerability Assessment, **(2008)**, pp. 108-125.
[22] R. Moskovitch, C. Feher and N. Tzachar, "Unknown malcode detection using opcode representation", Intelligence and Security Informatics, **(2008)**, pp. 204-215.
[23] C. M. Bishop, "Neural networks for pattern recognition", Oxford University Press, **(1995)**.
[24] P. F. Brown, P. V. Della, P. V. Desouza and R. L. Mercer, "Class-based n-gram models of natural language", Computational Linguistics, vol. 18, no. 4, **(1992)**, pp. 467-479.
[25] P. Likarish, E. Jung and I. Jo, "Obfuscated malicious JavaScript detection using classification techniques", The 4th International Conference on Malicious and Unwanted Software, **(2009)**, pp. 47-54.
[26] I. Santos, Y. Penya, J. Devesa and P. G. Bringas, "N-Grams-based file signatures for malware detection", Proceedings of the 11th International Conference on Enterprise Information Systems, **(2009)**, pp. 317–320
[27] Y. H. Choi, T. G. Kim and S. J. Choi, "Automatic detection for JavaScript obfuscation attacks in web pages through string pattern analysis", Future Generation Information Technology, **(**2009**)**, pp. 160-172.
[28] I. Santos, F. Brezo and J. Nieves, "Idea: opcode-sequence-based malware detection", Engineering Secure Software and Systems, **(2010)**, pp. 35-43.

 Copyright ⓒ 2015 SERSC