Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

JSContana: Malicious JavaScript detection using adaptable context analysis and key feature extraction

Yunhua Huang[a], *Tao Li*[a,*], *Lijia Zhang*[b], *Beibei Li*[a], *Xiaojie Liu*[a]

[a] *School of Cyber Science and Engineering, Sichuan University, Chengdu, China*
[b] *School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China*

ARTICLE INFO

Article history:
Received 26 October 2020
Revised 30 December 2020
Accepted 2 February 2021
Available online 8 February 2021

Keywords:
Malicious JavaScript detection
Context analysis
Dynamic word embeddings
Key feature extraction
TextCNN

ABSTRACT

JavaScript has played a crucial role in web development, making it a primary tool for hackers to launch assaults. Although malicious JavaScript detection methods are becoming increasingly effective, the existing methods based on feature matching or static word embeddings are difficult to detect different versions and obfuscation of JavaScript code. To solve this problem, we present JSContana, a novel detection method that consists of adaptable context analysis and efficient key feature extraction. The key to our approach is context analysis based on dynamic word embeddings. We convert JavaScript code to syntax unit sequences with detailed information and get the real contextual representation of code by dynamic word embeddings. Furthermore, as a classification module in the method, TextCNN can effectively extract key features. To demonstrate the performance of the method, we have conducted extensive comparison experiments under five-fold cross-validation. Numerical results show that the method achieves 0.990 in AUC-score, and outperforms the state-of-the-art method by up to 3.5%.

1. Introduction

Most websites use JavaScript to create dynamic interaction, which brings great convenience to people's daily life, but also brings many security problems. According to the "2019 Internet Security Threat Report" (Symantec, 2019), the hackers used malicious JavaScript code to steal information trended upwards. Symantec data shows that 4,818 unique websites were compromised with malicious JavaScript code every month in 2018. Malicious JavaScript codes are challenging to detect because they can transcode malicious code by obfuscation methods, such as logical operation, string segmentation and string compression. Furthermore, JavaScript version iteration also makes detection more difficult.

The existing methods for malicious JavaScript detection mainly rely on feature matching or static word embeddings. However, feature learning based on static word embeddings does not represent the real context information of code, which means the problem of polysemy in the code is not resolved, making undesirable detection effects. Besides, the analysis methods based on feature matching can only detect known malicious samples.

Aiming at the problem of undesirable detection effects caused by insufficient use of code information in existing methods, we present JSContana, a novel detection method using adaptable context analysis and efficient key feature ex-

* Corresponding author.
E-mail address: litao@scu.edu.cn (T. Li).
https://doi.org/10.1016/j.cose.2021.102218
0167-4048/

traction. To achieve this, Word2Vec and two Bidirectional Long Short-Term Memory (Bi-LSTM) layers are introduced into feature learning of the model. It is feature learning based on dynamic word embeddings that makes full use of code information. In TextCNN, the object of convolution operation is the word vector, and convolution kernel width is consistent with the dimension of word vectors, which can extract features effectively. In summary, the main contributions of this paper are as follows:

- Adaptable context analysis: Dynamic word vectors is introduced to represent real contextual information of code, making features of code learned sufficiently.
- Key feature extraction: With TextCNN model for feature classification, key features are extracted effectively to improve detection accuracy.
- Sufficient utilization of code information: Source code is parsed into syntax unit sequences with detailed information. Unlike the existing methods that only extract node information, sufficient utilization of code information in this paper can obtain better detection effects.

2. Related work

We divide related work into two categories: static analysis methods based on code content and structure; dynamic analysis methods based on dynamic execution results.

2.1. *Static analysis method*

The traditional static analysis methods are to extract malicious code features to form a feature database, and then match the detected code with the feature database. With the rapid development of machine learning and deep learning, malicious JavaScript detection methods based on machine learning has attracted the attention of researchers. It typically converts code into appropriate eigenvectors and models these vectors to estimate whether the sample is a malicious JavaScript code.

Curtsinger et al. (2011) propose a detection method with low cost and high speed, which identifies highly predictive syntax elements based on the hierarchical features of the abstract syntax tree of JavaScript. EvilSeed starts from a known malicious page to search for similar malicious pages (Invernizzi et al., 2012). (Andreasen and Møller, 2014) analyzes JavaScript data streams immediately and certainly through sensitive path and propagation persistence. However, traditional detection methods mainly rely on digital signature and artificial feature extraction, which is difficult to detect the confused malicious JavaScript files. In recent years, more and more machine learning methods are introduced to detect malicious JavaScript. In order to analyze the performance of machine learning methods to distinguish between obfuscated and no-obfuscated JavaScript files, Tellenbach et al. (2016) conducted extensive experiments to compare the performance of nine different classifiers. Kar et al. (2016) proposed a Support Vector Machine-based SQL injection attack detection method to train and test the Support Vector Machine by creating token diagrams. Fass et al. (2019) extracted code features based on abstract code and semantic information, classified the features with random forest. Ndichu et al. (2019) extracted code features from two dimensions, and obtained fixed-length feature representation with Doc2Vec. Fang et al. (2020) detected malicious JavaScript code based on semantic analysis. This method used FastText to learn the characteristics of JavaScript code, which paid more attention to BiLSTM-Attention detection model rather than feature representation. In summary, the existing static detection methods mainly use static word embeddings to learn the features representation of code, which means that static word vectors cannot represent the real context information of the code, making the polysemy of code cannot be solved. It is inevitable that insufficient learning of code information leads to undesired detection effects.

2.2. *Dynamic analysis method*

The dynamic analysis methods can capture the behaviors of JavaScript code at run time. Researchers execute JavaScript files in test environments such as sandboxes and honeypots, and then extract behavioral features during code executioner for classification.

Kim et al. (2012) proposed a dynamic detection method based on internal hook function, which monitors the running process of the program and analyzes its behaviors. Kapravelos et al. (2013) detects malicious code according to the differences of codes, and learns feature representation by identifying the similarities of massive JavaScript files. Xue et al. (2015) uses Deterministic Finite Automaton to learn the behavior features of malicious code in the dynamic execution process and summarizes the features, so that detailed information of the attack behaviors is obtained. Wang et al. (2015) classified the attack behavior by combining feature representations and execution traces of the codes. The feature representations and execution traces are obtained by machine learning and dynamic execution process respectively. Kim et al. (2017) builts a forced JavaScript execution engine, which detects malicious behaviors by constantly exploring executable paths and parameters of functions. Starov et al. (2019) executs all the JavaScript files, and marks special behavior of codes. Finally, attack data is obtained by monitoring enormous URLs.

The static detection method is adopted in this paper for the following reasons. Malicious JavaScript code can detect whether the runtime environment is a real browser in a specific way. For example, malicious code will not be executed when ActiveX plug-ins exists in the environment. Besides, compared with static analysis, dynamic analysis methods often require more resources and execution time. Aiming at the shortcomings of existing detection methods, this paper proposes JSContana, a novel and static detection method based on context analysis and key feature extraction. This method is described detailed in the next section.

3. Proposed method

As shown in Fig. 1, the framework of JSContana consists of data preprocessing, feature learning and classification. In data preprocessing, for each JavaScript file, we parse it into an ab-

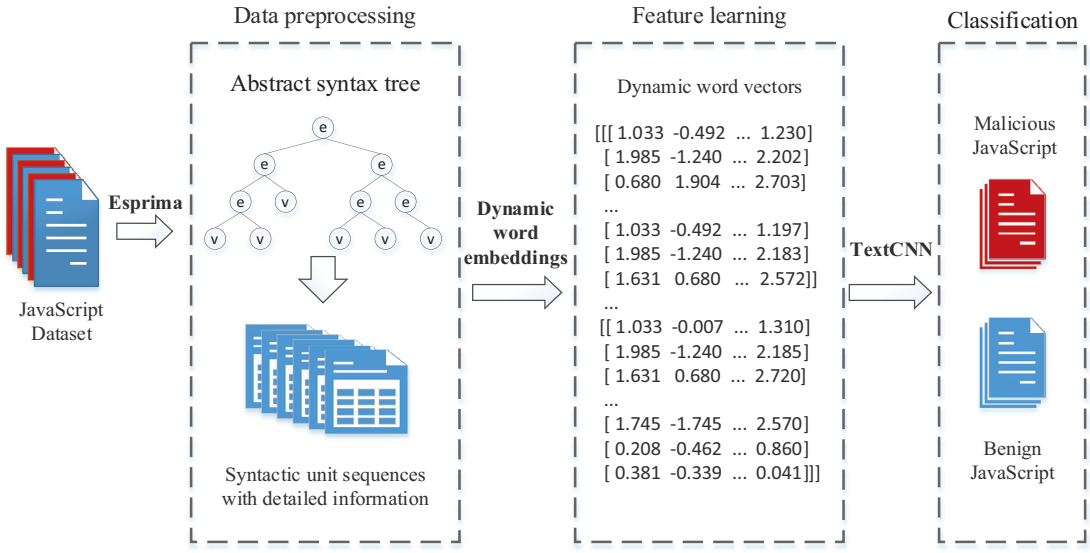


Fig. 1 – The Framework of JSContana.

stract syntax tree, each line of which is regarded as a syntax unit. Then an abstract syntax tree is transformed into syntax unit sequences with detailed information. For feature learing, the dynamic word embeddings model is introduced based on Word2Vec and two Bi-LSTM layers network, which obtains dynamic word vectors by context analysis of syntax unit sequences. The same syntax unit in different context is represented by different word vectors, which means that the polysemy problem of code is solved. Finally, dynamic word vectors are fed into the TextCNN model to classify JavaScript files by key feature extraction.

3.1. *Sufficient utilization of code information*

Abstract syntax tree is an abstract representation of the syntax structure of source code (Wikipedia, 2019), and its tree-like form represents the syntax structure of a programming language. In addition, it does not represent every detail that occurs in real syntax. For example, nested parentheses are implicit in the structure of the tree and are not presented as nodes. More and more malicious JavaScript code employs obfuscation to hide their maliciousness and evade detection. The abstract syntax tree can accurately represent this confusing JavaScript code to help detect malicious code.

Esprima is an efficient and powerful parser that performs reliably lexical and syntactic analysis of different versions and confusing JavaScript code to produce abstract syntax trees (Hidayat, 2018). A sample JavaScript file are as follows: var a=1. The result of the JS file parsed by Esprima is shown in Listing 1. JavaScript code is represented in JavaScript object notation. Each line of the syntax tree is a syntax unit, and special characters are removed from each syntax unit as input to the feature learning. Specifically, the first line in Listing.1 is the syntax unit '{', and the second line is the syntax unit 'type: Program'. Finally, the abstract syntax tree has 24 lines, which means 24 syntax units. Note that this paper treats each line of the abstract syntax tree as a syntax unit, rather than each line of the source code.

3.2. *Adaptable context analysis based on dynamic word embeddings*

After extracting the syntactic unit sequences with detailed information, we transform these sequences into dynamic vector representations. Unlike traditional static word embeddings, dynamic word vectors carry real context information (Peters et al., 2018), which is beneficial to downstream classification tasks based on key feature extraction.

For example, for "A, B, C, A, D" sequences, the word vectors of 'A' generated by the static word embeddings model are identical. However, the word vectors generated by the dynamic word embeddings model are different. At the end of the language model training, the word vectors of the same syntax unit in different contexts are completely consistent, but the dynamic word embeddings model makes each syntactic unit contain the context information of its location, so that the same syntactic unit producing different word vectors due to their different contexts.

As shown in Fig. 2, the dynamic word embeddings model is composed of Word2Vec (Mikolov et al., 2013) and two Bi-LSTM layers network (Peters et al., 2017). A sequence is converted into static word vectors by Word2Vec, which are input into the two Bi-LSTM layers network and transformed into dynamic word vectors with real context information.

Given a sequence of N syntactic units, $(u_1, u_2, \ldots, u_N)$, a forward language model (two-layers LSTM) computes the probability of syntactic unit u_s by given the previous syntactic units $(u_1, u_2, \ldots, u_{s-1})$ as formula (1), the probability of the sequence is calculated using the same method. We use the classical static word embedding model Word2Vec to generate its representation x_s then pass it through two layers of forward LSTMs. At each syntactic unit s, each LSTM layer output an interme-

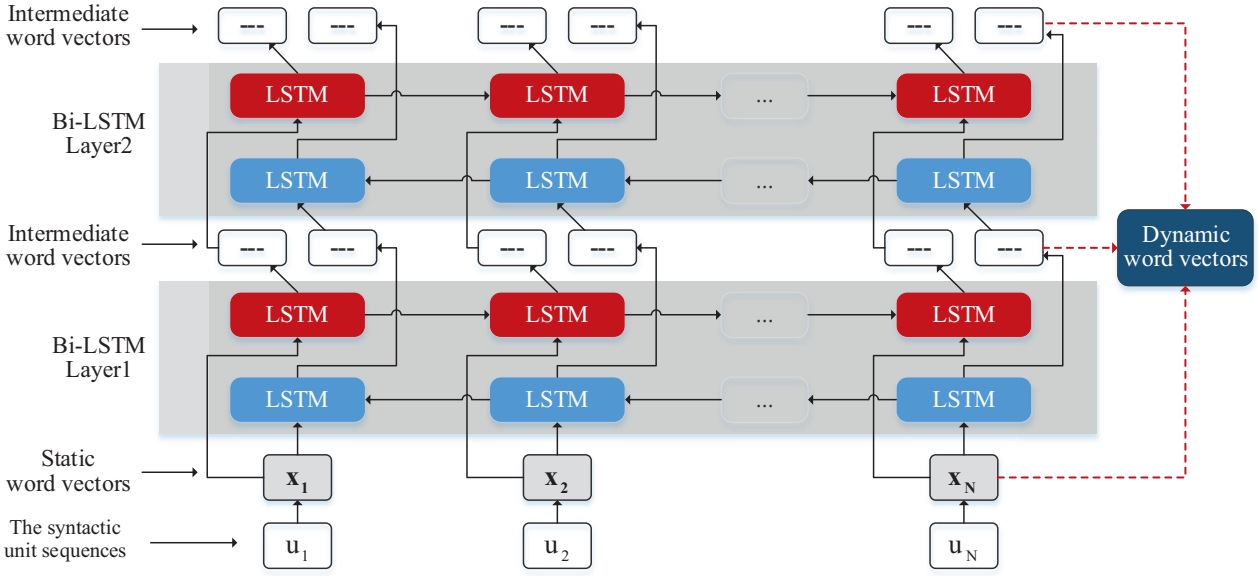


Fig. 2 – Dynamic word embedding model based on two Bi-LSTM layers.

diate word vector representation $\tilde{h}_s^j$ (j=1, 2) with context information.

$$p(u_1, u_2, \ldots, u_N) = \prod_{s=1}^{N} p(u_s|u_1, u_2, \ldots, u_{s-1}) \tag{1}$$

Similar to the forward language model, the backward language model computes the probability of the current syntactic unit by given following syntactic units. Similarly, at each syntactic unit s, each LSTM layer output an intermediate word vector representation $\overleftarrow{h}_s^j$ of u_s by given the future context $(u_{s+1}, u_{s+2}, \ldots, u_N)$ as formula (2).

$$p(u_1, u_2, \ldots, u_N) = \prod_{s=1}^{N} p(u_s|u_{s+1}, u_{s+2}, \ldots, u_N) \tag{2}$$

Dynamic word embedding model combines forward and backward language model. As formula (3), jointly maximizes the log likelihood function of bidirectional model. Where Θ_x is the parameter for syntactic unit representation layer and Θ_s is the parameter for Softmax layer, which shared in the forward and backward language models, except for the LSTM parameters (θ_{LSTM}). In fact, it is equivalent to training forward and backward language model respectively.

$$\sum_{k=1}^{N} \left(\log p(u_s|u_1, \ldots, u_{s-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log p(u_s|u_{s+1}, \ldots, u_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)\right) \tag{3}$$

For each syntactic unit u_s, a bidirectional language model computes a set of 2L+1 representation, where L represents the layer number of the bidirectional language model, h_s^0 is the static word embedding layer and $h_s^j = [\tilde{h}_s^j; \overleftarrow{h}_s^j]$ as formula (4).

$$R_s = \left\{x_s, \tilde{h}_s^j, \overleftarrow{h}_s^j \mid j = 1, 2\right\} = \left\{h_s^j \mid j = 0, 1, 2\right\} \tag{4}$$

Different layers of the model learn different information, so it is necessary to calculate the weighted combination of the three features. Considering the downstream classification task, we fold all the layers in R into a vector and use V to represent this dynamic word vector. We compute the task-specific weights for all the bidirectional language model layers as formula (5), s_j are softmax-normalized weights that indicates how much attention should be placed on each layer and the scalar parameter γ is a global scaling factor that allows the task model to scale the entire V vector.

$$V_s = E(R_s; \Theta) = \gamma \sum_{j=0}^{L} s_j h_s^j \tag{5}$$

3.3. *Key feature extraction*

Malicious code is only part of the JavaScript file, so classification model should make full use of this characteristic. TextCNN (Kim, 2014)is a text classification model suitable for variable length text, which core idea is to extract key features with varying window lengths.

The framework for TextCNN is shown in Fig. 3, $v_s \in \mathbb{R}^k$ represents the k-dimensional dynamic word vector representation corresponding to the s-th syntactic unit in the sequence. The syntactic unit sequence with length n is represented as formula (6), where $\oplus$ is the concatenation operator.

$$v_{1:n} = v_1 \oplus v_2 \oplus \ldots \oplus v_n \tag{6}$$

$filter \in \mathbb{R}^{hk}$ is a key part of convolution operation, producing a new feature in a window of *h* syntactic units. For example, it produces a new feature t_s in a window of $v_{s:s+h-1}$ words by formula (7), where $b \in \mathbb{R}$ is a bias term and *f* is a non-linear function.

$$t_i = f(w \cdot v_{s:s+h-1} + b) \tag{7}$$

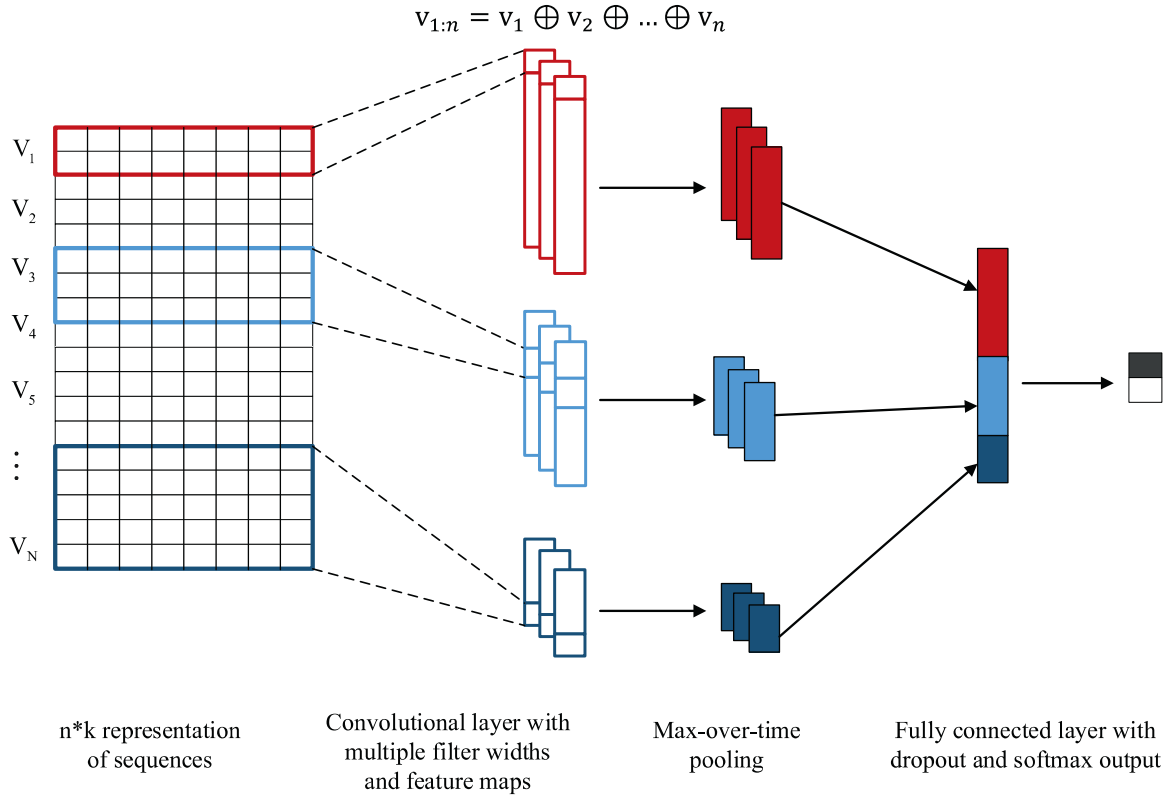


Fig. 3 – The Framework for TextCNN.

This filter slides down the matrix of the sequences $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$ to produce a feature map $t \in \mathbb{R}^{n-h+1}$ as formula (8). Further, perform max-over-time pooling operation (Collobert et al., 2011) on feature map and take the maximum value $\hat{t} = \max\{t\}$ as the feature representation corresponding to this filter.

$$t = [t_1, t_2, \ldots, t_{n-h+1}] \tag{8}$$

This is the process of extracting a feature from a filter. Different filters of TextCNN can obtain different features. Finally, a fully connected Softmax layer uses these features to compute the probability distribution over labels.

4. Experiments

4.1. *Experimental setup*

Data set: There are 8000 unique JavaScript samples in our data set. In particular, they are composed of 4000 benign and 4000 malicious JavaScript files. These JavaScript files were obtained separately, all benign samples come from Alexa (Cooper, 2020) top 100 by depth crawling, and the malicious samples were from Github (HynekPetrak, 2017), malicious platform VX Heavens (VX, 2020) and Curtsinger et al. (2011). In order to verify that the malicious data set is malicious, 200 samples are randomly selected for manual verification, and the results shown in Table 1 are all malicious. In addition, considering that the malicious samples may be larger than the benign samples, we calculated the file size in the dataset. As shown in Table 2, the size of benign and malicious samples is balanced. Note that in order to ensure that the size of the benign sample is similar to malicious sample, we screened the benign sample by size. Finally, we randomly split the data set into training set and test set at a ratio of 4:1.

Devices and environment: We built this model on Linux 3.10.0. An Nvidia Geforce RTX 2080 GPU and an Intel Xeon E5-2678 v3 CPU are used in our experiment. Besides, the software config-

Table 1 – Manual validation results of 200 malicious samples.

Type	Total
Drive-by-Download	48
Cross-site scripting	35
Cross-Site Request Forgery	26
Heap spraying attacks	14
Other malicious	77
Benign	0

Table 2 – Statistical results of sample size in data set.

Data type	<1k	1k-5k	5k-10k	>10k
Benign	1512	2273	148	67
Malicious	1480	2300	150	70

uration is as follows: Keras 2.2.5, Esprima 4.0.1, Bilm 0.1, and Tensorflow-GPU 1.2.

Parameters: The experimental procedure of the method proposed in this paper is as follows. For each JavaScript file, we first use Esprima to get an abstract syntax tree shown in Listing 1 and extract the sequences of syntax units with detailed information. Then, 187 different syntax units were obtained. Second, we train a Word2Vec model with all syntax unit sequences to represent syntax units as static word vectors. Word2Vec model parameters include size=50, window=8, and min count=1. The static word vectors are used as the input of two Bi-LSTM layers network model to obtain the dynamic word vectors representation. The training parameters of JSContana include negative samples batch=100, epochs=10, dim=100, and train tokens = 400000. Finally, TextCNN is used for classification. We found that four convolution kernels worked best, detailed parameters are as follows: kernel size=7, 15, 25, 35, activation=tanh, batch size=30, epochs=20.

Comparison experiment: To evaluate the effectiveness of the proposed method in this paper, we have designed two types of comparative experiments: longitudinal comparative experiment and horizontal comparative experiment. Longitudinal comparative experiments verify the validity of the method structure by changing the method structure. Horizontal comparison experiment verifies the effectiveness of this method by comparing with other methods. The longitudinal comparison experiments are as follows. Three variant models of JSContana called JSContana-P, JSContana-S, and JSContana-R, respectively. We don't extract the syntax unit sequences with detailed information in data preprocessing for JSContana-P, but only the syntax unit sequences at the node. For JSContana-S, we use static word vectors derived from Word2Vec instead of dynamic vectors in feature learning. For JSContana-R, we use TextRNN instead of TextCNN in classification. The horizontal comparison experiments are as follows. We compared JSContana with four other models. BiLSTM-Attention (Fang et al., 2020) is the latest malicious JavaScript code detection model. Its structure is similar to JSContana, but it focuses on the detection model and uses static word embedding. Besides, we also reproduce three machine learning models for basic comparison (Wang et al., 2014), namely, Support Vector Machine (SVM), and Decision Tree.

Evaluation standard: In order to make full use of the value of the dataset and objectively validate different models, we chose 5-fold cross-validation as the test method when comparing with other methods. The main evaluation metrics are accuracy, precision, recall, and F1-score, which defined as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$precision = \frac{TP}{TP + FP} \tag{10}$$

$$recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{12}$$

where TP as the number of malicious JavaScript files correctly classified as malicious, TN as the number of benign JavaScript files correctly classified as benign, FN as the number of malicious JavaScript files classified as benign, and FP as the number of benign JavaScript files classified as malicious. Besides, a receiver operating characteristic (ROC) graph was used for visual performances of models, and the area under curve (AUC) score was calculated to determine the model performance.

4.2. *Performance comparison*

The results of longitudinal comparison experiments are shown in Table 3. It can be seen from the table that compared with JSContana-P, JSContana-S, and JSContana-R, JSContana achieves the best performance in all metrics, which means that each part of the JSContana can improve its detection accuracy. The JSContana-S showed the lowest accuracy, recall, and F1-score, which indicated that the dynamic word vector introduced in the detection model contributed the most to improving model accuracy. All JSContana-R indicators are lower than those of JSContana, which means that TextCNN is more suitable for detecting malicious files than TextRNN because it is good at key feature extraction. The indexes of JSContana-P are in the middle level, indicating that the use of syntax unit sequences with detailed information in data preprocessing helps improve the accuracy of model detection.

The results of horizontal comparison experiments under five-fold cross-validation are shown in Table 4. The JSContana model is superior to the BiLSTM-Attention model and other traditional models in all aspects, demonstrating that JSContana can make full use of code information, effectively extract code features and improve the detection accuracy of different versions and obfuscation of JavaScript code.

To further explore the effect of the method proposed in this paper, we randomly shuffle the data set and conducted comparison experiment again. The ROC curve and AUC score of each model are shown in Fig. 4. It can be seen from the AUC score of the JSContana is also larger than that of the BiLSTM-Attention model, SVM, and Decision Tree. The experiment proves that JSContana based on context analysis and key feature extraction is useful, with highly accurate.

We recorded the processing time of the model on training samples and test samples, as shown in Table 5. JSContana requires more time for model training because the dynamic word embedding model converges slowly. In addition, the model loading time and detection time of JSContana and BiLSTM-Attention are relatively close, which proves that JSContana has an efficient detection effect after model training. Although traditional machine learning takes less time, detec-

Table 3 – Results compared with longitudinal comparison experiments.

Model	Accuracy	Precision	Recall	F1-score
JSContana	0.989	0.984	0.993	0.989
JSContana-P	0.967	0.978	0.955	0.967
JSContana-S	0.950	0.960	0.965	0.963
JSContana-R	0.968	0.971	0.964	0.967

Table 4 – Results compared with horizontal comparison experiments under five-fold cross-validation.

Model	Accuracy	Precision	Recall	F1-score
JSContana	0.988 (± 0.007)	0.988 (± 0.005)	0.989 (± 0.008)	0.988 (± 0.007)
BiLSTM-Attention	0.955 (± 0.014)	0.971 (± 0.012)	0.961 (± 0.011)	0.966 (± 0.011)
SVM	0.886 (± 0.022)	0.875 (± 0.028)	0.902 (± 0.015)	0.888 (± 0.015)
Decision Tree	0.843 (± 0.034)	0.853 (± 0.043)	0.833 (± 0.033)	0.842 (± 0.033)

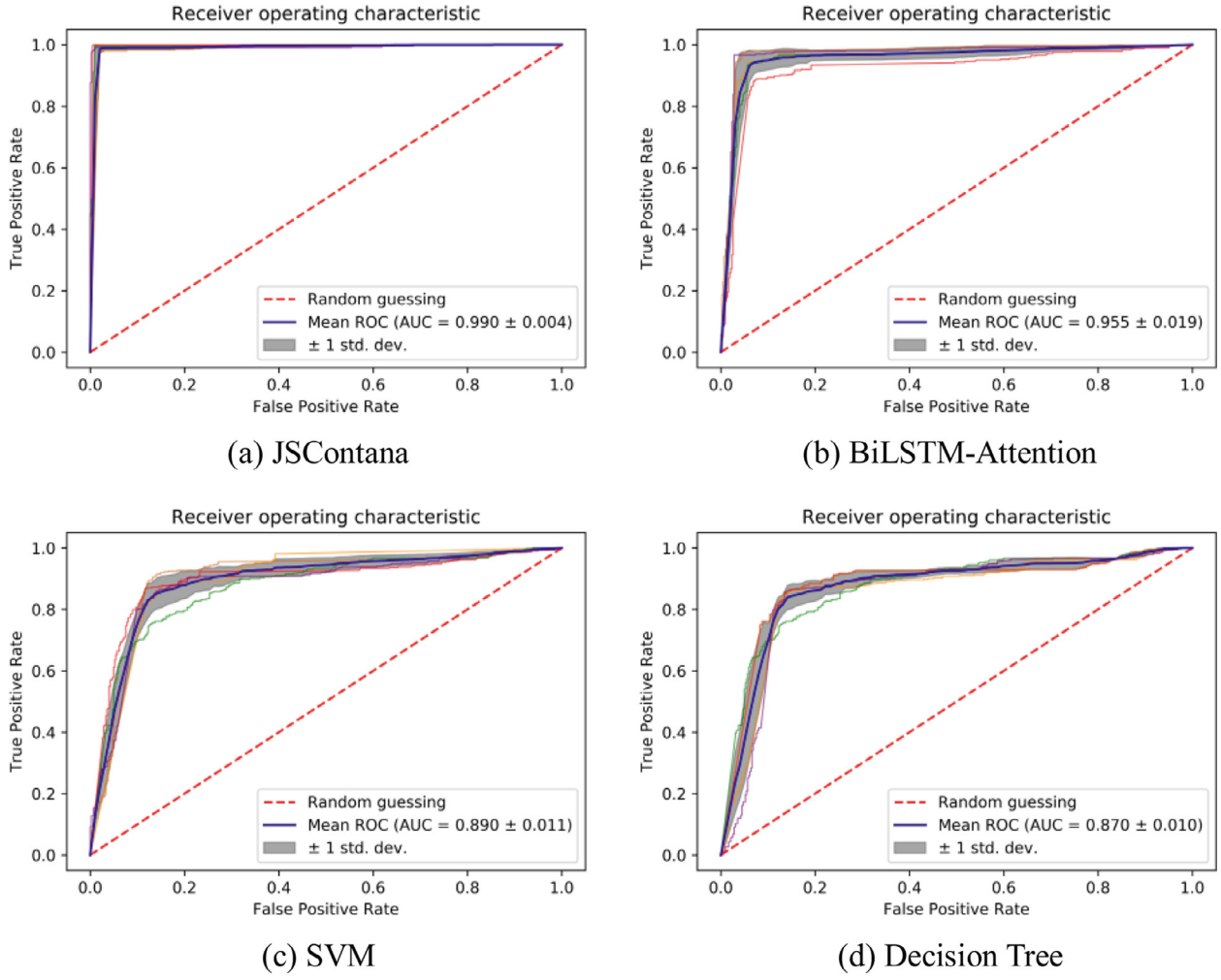


Fig. 4 – ROC curves and AUC scores of different detection models under five-fold cross-validation.

Table 5 – Average processing time for each sample by different models.

Model	Training time	Model load time	Detection time
JSContana	47.848s	6.584ms	2.792ms
BiLSTM-Attention	1.156s	3.359ms	4.206ms
SVM	0.011s	0.118ms	0.121ms
Decision Tree	0.003s	0.107ms	0.094ms

tion methods based on deep learning have better detection results.

4.3. *Discussion*

Experiments show that JSContana can effectively detect confused malicious samples. Context analysis based on dynamic word embedding can effectively solve the problem of code ambiguity in static analysis, and the detection accuracy is improved by sufficient utilization of code information. However, during the experiment we found the following limitations of JSContana:

- Longer training time: Dynamic word embedding is composed of a multi-layer network, which results in a longer time for model convergence.

```
{
  "type": "Program",
  "body": [
        "type": "VariableDeclaration",
        "declarations": [
          {
            "type": "VariableDeclarator",
            "id": {
              "type": "Identifier",
              "name": "a"
            },
            "init": {
              "type": "Literal",
              "value": 1,
              "raw": "1"
              …
  "sourceType": "script"
}
```

Listing 1 – The Result of JavaScript code processed by Esprima.

- Binary classification: JSContana can detect whether the detected samples are malicious, but cannot classify the malicious samples.
- Samples length limit: TextCNN can detect the variable-length samples, but a maximum is required for the model implementation. If the sample length exceeds the maximum, it may result in unsatisfactory test results.

5. Conclusion

In this paper, we have proposed JSContana, a novel and static malicious JavaScript code detection method. In order to improve the detection effect of different versions and obfuscation JavaScript code, JSContana extracts the syntax unit sequences with detailed information in the data preprocessing part. Feature learning based on dynamic word embeddings makes word vectors carry real context information of code, which is the key of JSContana. In addition, TextCNN as a classification module in the method can effectively key feature extraction. Extensive comparison experiments show that the accuracy of JSContana is up to 0.988, and the AUC score reaches 0.990 under 5-fold cross-validation, and outperforms the state-of-the-art method by up to 3.5%. This means that JSContana can effectively detect different versions and obfuscation of malicious JavaScript code.

However, JSContana still exists limitations. JSContana consists of a multi-layer network, which means that longer training time and more training samples are required. The classification model needs to set the maximum value of input data when it is built, which leads to the possibility of long malicious samples escaping detection. In addition, the detection model cannot classify the malicious samples into multiple categories. In future, we will improve this so that the method can be better applied to malicious JavaScript detection.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (No. 2020YFB1805400); in part by the National Natural Science Foundation of China (No. U1736212, No. U19A2068, No. 62002248, and No. 62032002); in part by the China Postdoctoral Science Foundation (No. 2019TQ0217 and No. 2020M673277); in part by the Provincial Key Research and Development Program of Sichuan (No. 20ZDYF3145); in part by the Fundamental Research Funds for the Central Universities (No. YJ201933).

REFERENCES

Andreasen E, Møller A. Determinacy in static analysis for jQuery[C]. In: Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications; 2014. p. 17–31.

Collobert R, Weston J, Bottou L, et al. Natural language processing (almost) from scratch[J]. J. Mach. Learn. Res. 2011;12(ARTICLE):2493–537.

Cooper, K., 2020. Alexa. https://www.alexa.com/.

Curtsinger C, Livshits B, Zorn B G, et al. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection[C. In: USENIX security symposium; 2011. p. 33–48.

Fang Y, Huang C, Su Y, et al. Detecting malicious JavaScript code based on semantic analysis[J]. Computers & Security 2020;93.

Fass A, Backes M, Stock B. JStap: a static pre-filter for malicious JavaScript detection[C]. In: Proceedings of the 35th Annual Computer Security Applications Conference; 2019. p. 257–69.

Hidayat, A., 2018. Esprima master documentation.

HynekPetrak, 2017. Javascript malware collection. https://github.com/HynekPetrak/javascript-malware-collection.

Invernizzi L, Comparetti P M, Benvenuti S, et al. Evilseed: A guided approach to finding malicious web pages[C]. In: 2012 IEEE symposium on Security and Privacy. IEEE; 2012. p. 428–42.

Kapravelos A, Shoshitaishvili Y, Cova M, et al. Revolver: An automated approach to the detection of evasive web-based malware[C]. In: 22nd {USENIX} Security Symposium ({USENIX} Security 13); 2013. p. 637–52.

Kar D, Panigrahi S, SQLiGoT Sundararajan S. Detecting SQL injection attacks using graph of tokens and Support Vector Machine[J]. Computers & Security 2016;60:206–25.

Kim HC, Choi YH, Lee DH. JsSandbox: A Framework for Analyzing the Behavior of Malicious JavaScript Code using Internal Function Hooking[J]. KSII Transactions on Internet & Information Systems 2012;6(2).

Kim K, Kim I L, Kim C H, et al. J-force: Forced execution on javascript[C]. In: Proceedings of the 26th international conference on World Wide Web.; 2017. p. 897–906.

Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.

Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]. Advances in Neural Information Processing Systems 2013:3111–19.

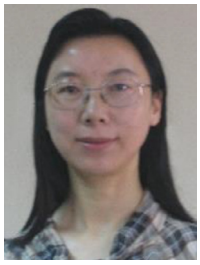
Ndichu S, Kim S, Ozawa S, et al. A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors[J]. Appl. Soft Comput. 2019;84.
Peters M E, Ammar W, Bhagavatula C, et al. Semi-supervised sequence tagging with bidirectional language models[J]. arXiv preprint arXiv:1705.00108, 2017.
Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations[J]. arXiv preprint arXiv:1802.05365, 2018.
Starov O, Zhou Y, Wang J. Detecting Malicious Campaigns in Obfuscated JavaScript with Scalable Behavioral Analysis[C]. In: 2019 IEEE Security and Privacy Workshops (SPW). IEEE; 2019. p. 218–23.
Symantec. 2019. https://docs.broadcom.com/doc/istr-24-2019-en.
Tellenbach B, Paganoni S, Rennhard M. Detecting obfuscated JavaScripts from known and unknown obfuscators using machine learning[J]. Int. J. Adv. Secur. 2016;9(3/4):196–206.
VX Valut, 2020, Blog URL, http://vxvault.net/ViriList.php.
Wang J, Xue Y, Liu Y, et al. Jsdc: A hybrid approach for javascript malware detection and classification[C]. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security; 2015. p. 109–20.
Wang W, Wang X, Feng D, et al. Exploring permission-induced risk in android applications for malicious application detection[J]. IEEE Trans. Inf. Forensics Secur. 2014;9(11):1869–82.
Wikipedia, 2019. Abstract syntax tree. https://en.wikipedia.org/wiki/Abstract_syntax_tree.
Xue Y, Wang J, Liu Y, et al. Detection and classification of malicious JavaScript via attack behavior modelling[C]. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis; 2015. p. 48–59.

Yunhua Huang received the B.Eng. degree in Hebei University of Technology, Tianjin, China, in 2019. He is currently pursuing the M.A. degree with the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include web security and artificial intelligence.

Tao Li received the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, in 1994. He is currently a Professor with the School of Cyber Science and Engineering, Sichuan University, China. His main research interests include network security, and cloud computing and cloud storage.

Lijia Zhang received the B.Eng. degree in Xiamen University, Xiamen, China, in 2017. She is currently pursuing the M.A. degree with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China. Her current research interests include communication and information system and artificial intelligence.

Beibei Li received the Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, in 2019. He is currently an associate professor with the School of Cyber Science and Engineering, Sichuan University, P.R. China. His current research interests include several areas in security and privacy issues on cyber-physical systems, with a focus on intrusion detection techniques, artificial intelligence, and applied cryptography.



Xiaojie Liu received the M.Eng. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1991. She is currently an professor with the School of Cyber Science and Engineering, Sichuan University, P.R. China. Her current research interests include network security and data protection.