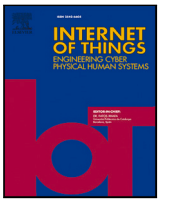
Internet of Things 19 (2022) 100521

Contents lists available at ScienceDirect

Internet of Things

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



Evaluation of printable character-based malicious PE file-detection method

Mamoru Mimura

National Defense Academy 1-10-20 Hashirimizu, Yokosuka, Kanagawa, Japan

ARTICLE INFO

Keywords:
Malware
Machine learning
Natural language processing

ABSTRACT

Printable characters extracted from portable executable (PE) files are a common surface analysis feature. String extraction is a supplemental feature for malware analysis. Recent developments in natural language processing techniques have enabled the rapid detection of malicious PE files. Previously, we proposed a method for detecting malicious PE files using printable characters using two language models for feature extraction and machine-learning. In this study, we evaluated the method using the latest FFRI dataset consisting of 400,000 benign and 400,000 malicious samples between 2019 and 2021. To the best of our knowledge, this is the first study to consider the time series of both malicious and benign samples. According to the results, specific tokens in the printable characters were effective in detecting the latest malicious PE files. The most practical combination was of the Doc2vec and multilayer perceptron, which achieved an F1 score of 0.981. Each run time showed an almost linear increase with increasing dataset size.

1. Introduction

Printable characters extracted from portable executable (PE) files are a common surface analysis feature. A string is a sequence of characters that can be a constant or a variable. A string contains useful information such as IP address, domains, functions, data, or any other information that has not been removed. String extraction is a supplemental feature for malware analysis. Recent developments in natural language processing (NLP) techniques have enabled the rapid detection of malicious PE files. Previously, a method for detecting malicious PE files using printable characters was proposed [1]. This method uses two language models for feature extraction and machine-learning models. The study showed that printable characters are effective in detecting malicious PE files.

However, whether this method can detect the latest malware in a real-world environment remains to be seen. Several studies simply split samples into training and test data without considering the time series of samples. Other studies have focused on the time series of malicious PE files. Few studies have considered the time series of both benign and malicious samples. While several papers have reported detection accuracies, others have not revealed the time complexity. For practical purposes, models need to be trained with sufficient samples in a reasonable amount of time, for which actual samples may be too large. Nonetheless, few studies have analyzed the time complexity in this context. The time complexity should be analyzed to determine the most practical model.

In this study, we evaluated our proposed method [1] by using the latest FFRI dataset consisting of 400,000 benign and 400,000 malicious samples between 2019 and 2021. We considered the time series of both malicious and benign samples, which to the best of our knowledge, has not been reported before. Our method was effective for the latest FFRI dataset. Our time-complexity analysis revealed the most practical model in terms of feasibility. The key findings of our study are as follows.

E-mail address: mim@nda.ac.jp.

https://doi.org/10.1016/j.iot.2022.100521
Received 30 January 2022; Received in revised form 6 March 2022; Accepted 7 March 2022
Available online 19 March 2022

Table 1
Comparison of related studies using string extraction.

	Malicious samples size	Benign sample size	Time series	Time complexity
[3]	3,265	1,001	–	–
[4]	31,518	8,320	–	–
[6]	37,262	44,602	–	–
[1]	37,375	250,000	malicious	–
This study	400,000	400,000	both	yes

- Specific tokens in printable characters were effective at detecting the latest malicious PE files.
- The effect of the time series appears to be limited in our method, and updating the training samples may not improve detection accuracy.
- Efficient training samples may build a robust classification model with the passage of time.
- The most practical combination was of the Doc2vec and multilayer perceptron, which achieved an F1 score of 0.981. Each run time showed an almost linear increase with increasing dataset size.

2. Related work

2.1. String extraction

Our detection model is a combination of a natural language model and a machine-learning model. The natural language model is built with a string extracted from the PE files. A string is any sequence of four or more printable characters that ends with a new-line or a null character and can provide information about program functionality and indicators associated with a suspect binary. It can be extracted from PE files using surface analysis and does not require dynamic analysis. Therefore, it is used as a feature for machine-learning-based malware detection [2]. For instance, Schultz et al. presented a data-mining framework that detects previously unseen malicious PE files [3]. This method uses the DLL, function calls, or strings extracted from PE files using the naive Bayes algorithm. Ye et al. developed a malware detection system based on an interpretable string analysis with machine learning techniques [4]. This method uses support vector machine (SVM) ensembles with bootstrap aggregation to classify the file samples and predict the exact types of malware. Kolosnjaji et al. developed a neural network consisting of convolutional and feedforward neural constructs. This method combines the convolution of n-grams of instructions with plain vectorization of features derived from the headers of the PE files [5]. Aghakhani et al. examined how machine learning-based static analysis features operate on packed samples [6]. They extracted the printable character sequences from PE files and obtained binary features by removing rare strings, from which more than 99.99% were seen in less than 0.4% of the samples. Thus, string extraction is a useful feature for malware analysis and machine learning-based malware detection. Although a string does not provide a clear picture of the purpose and capability of a file, it can provide a hint about what malware can do. Recent developments in NLP techniques have enabled the detection of malicious PE files. To detect malicious PE files, a method using NLP techniques has been proposed [1]. This method extracts printable characters and builds two language models called latent semantic indexing (LSI) and Doc2vec for feature extraction. LSI, also known as latent semantic analysis, is a method for analyzing a set of documents to discover statistical co-occurrences of words that appear together. Doc2vec, also known as the paragraph vector, is an unsupervised method for learning distributed representations of text. However, whether this method can detect the latest malware in a real-world environment remains to be seen. Additionally, the time complexity was not reported for this study. For practical purposes, this model should be trained with sufficient samples in a reasonable amount of time. Table 1 shows a comparison of the related studies using string extraction.

Early studies with printable characters did not use sufficient samples for evaluation [3,4]. Previous studies either did not consider the time series [6] or only considered the time series of malicious samples [1]. In this study, we used the latest FFRI dataset consisting of 400,000 benign and 400,000 malicious samples between 2019 and 2021. Furthermore, we considered the time series of both malicious and benign samples and analyzed the time complexity.

2.2. Static features

Our method uses printable characters extracted from PE files. Other static features are used for malware detection.

Byte n-grams obtained from PE files are one of the most common features [7,8]. Abou-Assaleh et al. applied a common n-gram analysis for malware detection [9]. They used byte n-grams to generate signatures from malicious and benign samples. Kolter et al. used byte n-grams with naive Bayes, decision trees, SVMs, and boosting [10,11]. Zhang et al. used byte n-grams with multiple classifiers [12]. This method selects features based on information gain. Jacob et al. presented a similarity measure based on the distribution of byte bigrams [13]. They also presented a packer detection technique that can distinguish between different levels of protection, such as unpacked, compressed, encrypted, and multi-layer encrypted codes. Opcode n-grams [14–17] and program disassembly [18–22] were also used as features.

Other popular features of static malware detection are PE headers [23–26]. Shafiq et al. presented a framework for detecting malicious PE files [27]. This method leverages the structural information of PE files. Saxe et al. proposed a deep neural network

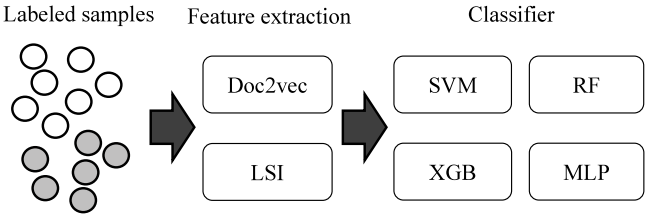


Fig. 1. Structure of the proposed detection model.

malware classifier with aggregated features [28]. They extracted the byte entropy histogram, import address table, and PE headers. Webster et al. focused on the hidden PE32 field known as the rich header [29]. They demonstrated how the information contained in the rich header can be leveraged to perform triage of samples, including packed and obfuscated binaries. Raff et al. explored the feasibility of applying neural networks to malware detection and feature learning [30]. They showed that neural networks can learn from raw bytes without explicit feature extraction.

2.3. Natural language processing

Our method uses NLP techniques for feature selection. NLP techniques are used to detect other types of malicious content or traffic [31,32]. Concerning malicious content, Doc2vec is used for detecting macro-malware [33,34], malicious Powershell scripts [35,36], or malicious JavaScript [37,38]. LSI is used for detecting macro-malware [39,40] and malicious Powershell scripts [35,36]. For detecting malicious traffic, Doc2vec is applied to proxy logs [41–43] or packet dump [44,45].

3. Detection model

3.1. Detection model

Recent developments in NLP techniques have enabled the rapid detection of malicious PE files. Previously, a method for detecting malicious PE files using printable characters has been proposed [1]. Fig. 1 shows the structure of the proposed detection model.

This method uses two language models, Doc2vec and LSI, for feature extraction and machine learning models, SVM, random forest (RF), XGBoost (XGB), and multilayer perceptron (MLP). These classifiers are popular in the various fields, and have each characteristic. Printable characters were obtained from the PE files and converted into tokens. These tokens consist of sequences of printable characters and are used to build these language models. While a previous study merely removed rare tokens [6], our method selected frequent tokens from both malicious and benign samples separately. These language models convert PE files into feature vectors. Feature vectors were used to train each classifier.

3.2. Baseline

While the previous study provided practical performance, it did not compare the method with other techniques. Owing to issues regarding the dataset or implementation, it is difficult to provide a fair comparison. To provide a fair comparison, we developed a simple detection method using string extraction. In this baseline method, the most frequent tokens were selected from the training samples. The selected tokens were converted into one-hot vectors. Other tokens in the training data or unseen tokens in the test data were ignored. Frequent token size was unified to the dimensions of these language models.

3.3. Implementation

We implemented the detection models using Python version 3.6.8. We used Gensim 3.6.0[1] to implement the language models. To implement the classifiers, we used sckit-learn 0.22.1.[2] and xgboost 0.90[3] MLP was implemented using TensorFlow 1.14.0.[4]

Based on our previous study [1], a grid search was used to optimize the parameters. Grid search is a search technique that has been widely used in many machine-learning studies. The optimized parameters are listed in Tables 2 and 3.

The other parameters were set to their default values.

[1] https://radimrehurek.com/gensim/
[2] https://scikit-learn.org/stable/
[3] https://xgboost.readthedocs.io/
[4] https://www.tensorflow.org/

Table 2
Optimized parameters for each language model.

Language model	Parameter	Optimum value
Doc2vec	dimension	400
	alpha	0.025
	min_count	2
	window	15
	epoch	30
LSI	dimension	400

Table 3
Optimized parameters for each classifier.

Classifier	Parameter	Optimum value
SVM	kernel	linear
	C	0.5
RF	probability	True
	criterion	gini
	max_features	10
	min_samples_split	3
	min_samples_leaf	1
	n estimators	200
MLP	optimizer	Adam
	epoch	30

Table 4
Outline of the FFRI dataset.

Class	Period	Size	Packed	Unique words
Benign	2019	250,000	30,749	71,472,934
	2020	75,000	7,842	163,972,568
	2021	75,000	7,699	171,405,803
Malicious	2019	250,000	74,698	42,897,523
	2020	75,000	32,247	18,172,121
	2021	75,000	35,430	16,765,218

4. Evaluation

4.1. Dataset

In this experiment, we used the FFRI dataset. The FFRI dataset is part of anti-malware engineering workshop (MWS) datasets [46]. The latest dataset was created by surface analysis[5] and consists of JSON files. This dataset contains strings extracted from both malicious and benign samples. Table 4 presents an outline of the FFRI dataset.

The samples were obtained between 2019 and 2021. Malicious samples were provided by the FFRI Security.[6] FFRI security is a security vendor and provides endpoint security solutions. These malicious samples are analyzed using their algorithms. Benign samples were obtained from the AV-TEST.[7] AV-TEST crawls over 100 download portals for new or updated software and then downloads, installs, and analyzes these products. According to the result of PEiD,[8] both the benign and malicious samples contained many packed samples. Note that signature-based static packer detection methods have false positives and negatives. We extracted strings from all samples and calculated the number of unique tokens. Thus, this dataset contained many samples and is well distributed.

4.2. Methodology

To evaluate the basic performance of our method, we used samples from 2019 for training and samples from 2020 for testing. The training samples were randomly divided into five parts. We repeated the experiment five times for each set and calculated the average score. To examine the effect of the time series, the test samples were updated to the samples in 2021. The training samples were then updated to samples in 2020. To examine the time complexity of our method, we increased both benign and malicious sample sizes. The sample size ranged from 1,000 to 100,000.

[5] https://github.com/ffri/ffridataset-scripts
[6] https://www.ffri.jp/en/
[7] https://www.av-test.org/en/
[8] https://www.aldeid.com/wiki/PEiD

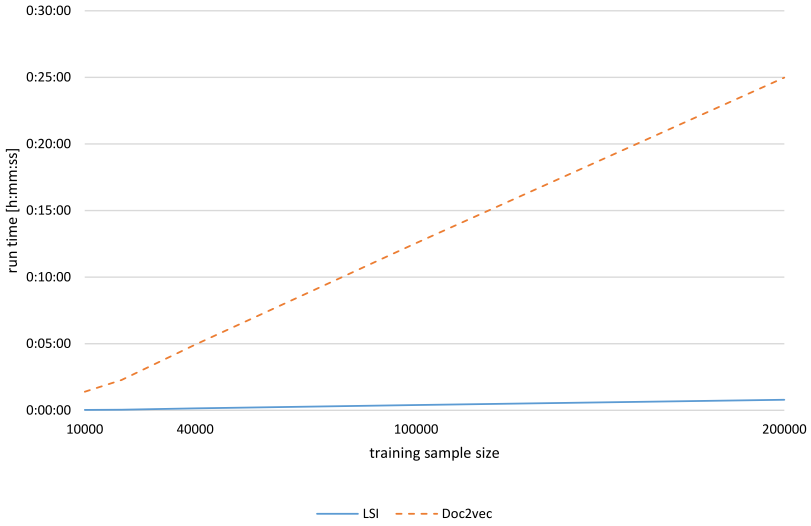


Fig. 2. Time complexity analysis of the LSI and Doc2vec models.

To evaluate the classification accuracy, we used basic metrics as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2Recall \times Precision}{Recall + Precision} \tag{4}$$

This experiment was conducted on a computer with Windows 10, Core i9- 7900X 3.3 GHz CPU, 128 GB DDR4 memory, and GeForce GTX 1080 Ti.

4.3. Time complexity

To analyze the time complexity of our method, the training sample size was increased. Fig. 2 shows the time variation of the LSI and Doc2vec models.

The horizontal and vertical axes indicate the training sample size and run time, respectively, for building each language model. Each run time for the building gradually increases with the training sample size. Doc2vec requires more time than the LSI for buildings. Figs. 3–5 show the time variation of each classifier with a one-hot vector, LSI, and Doc2vec in training.

The horizontal and vertical axes indicate the training sample size and run time, respectively, for training each classifier with a one-hot vector, LSI, or Doc2vec. Each run time for building both language models showed a linear increase. Each run time for training all classifiers except SVM showed an almost linear increase. The run time with SVM was unstable and dramatically increased with the training sample size. Finally, the test sample size was increased. Fig. 6 shows the time variation of each classifier during the testing.

The horizontal axis indicates the test sample size, and the vertical axis indicates the run time for testing each classifier. The run time for testing gradually increased with the test sample size. SVM required more time than other classifiers. Each run time for testing all classifiers showed a linear increase.

4.4. Comparison

To evaluate the basic performance of our method, samples from 2019 were used for training and samples from 2020 for testing. Owing to the drastic increase in training time, SVM could not provide the desired performance. Table 5 shows the performance of each combination for the 2019–2020 samples.

Each classifier provided better accuracy with one-hot vector and Doc2vec than without them. The combination of the Doc2vec and MLP achieved an F1 score of 0.981.

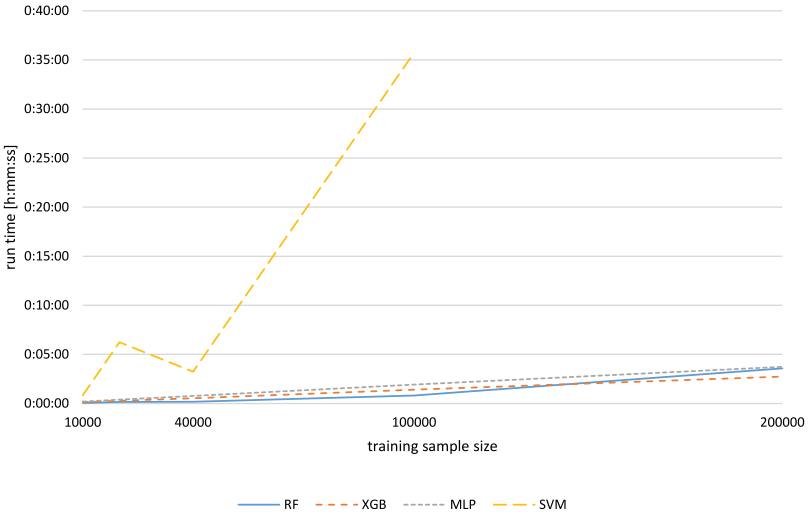


Fig. 3. Time complexity analysis of each classifier with one-hot vector in training.

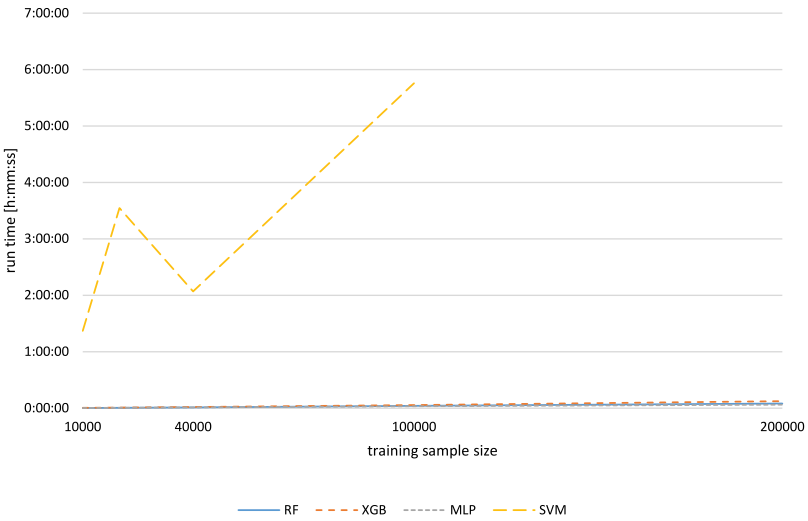


Fig. 4. Time complexity analysis of each classifier with LSI in training.

Table 5
Performance on 2019–2020 samples.

		Accuracy	Precision	Recall	F1
One-hot	RF	0.978	0.983	0.973	0.978
	XGB	0.973	0.978	0.968	0.973
	MLP	0.979	0.983	0.975	0.979
LSI	RF	0.931	0.966	0.893	0.928
	XGB	0.909	0.914	0.903	0.908
	MLP	0.948	0.961	0.939	0.947
Doc2vec	RF	0.977	0.984	0.970	0.977
	XGB	0.975	0.978	0.973	0.975
	MLP	0.980	0.982	0.979	**0.981**

4.5. Time series

The test samples were updated to samples in 2021. Table 6 shows the performance of each combination in the 2019–2021 samples.

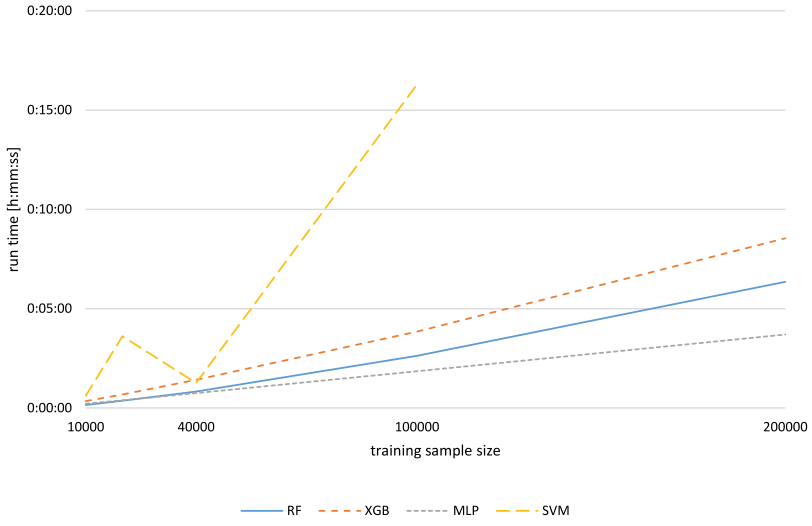


Fig. 5. Time complexity analysis of each classifier with Doc2vec in training.

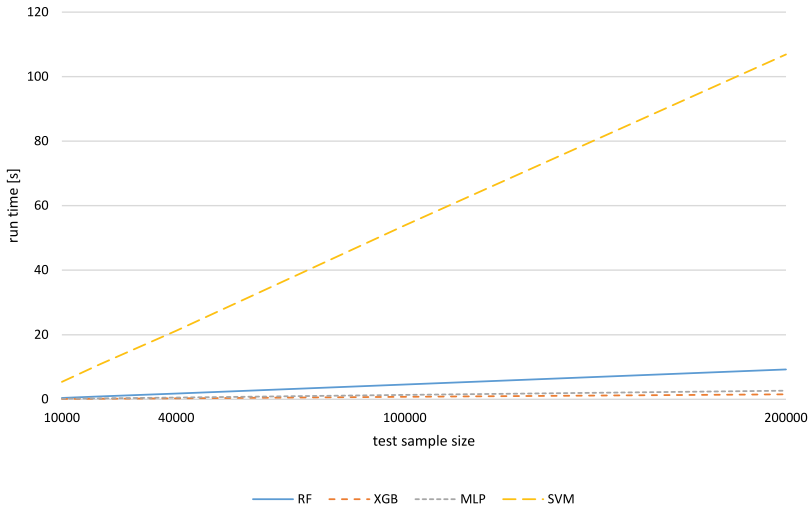


Fig. 6. Time complexity analysis of each classifier in testing.

Table 6
Performance on 2019–2021 samples.

		Accuracy	Precision	Recall	F1
One-hot	RF	0.995	0.999	0.989	**0.995**
	XGB	0.990	0.999	0.979	0.990
	MLP	0.994	0.999	0.989	0.994
LSI	RF	0.918	0.978	0.856	0.913
	XGB	0.893	0.926	0.855	0.889
	MLP	0.983	0.988	0.978	0.983
Doc2vec	RF	0.990	0.999	0.982	0.990
	XGB	0.983	0.992	0.975	0.983
	MLP	0.993	0.996	0.991	0.993

The performance of each combination remained constant over time. Similarly, each classifier provided better accuracy with a one-hot vector and Doc2vec than without them. The combination of the one-hot and RF achieved an F1 score of 0.994. The training samples were then updated to samples in 2020. Table 7 shows the performance of each combination in the 2020–2021 samples.

Contrary to our expectation, the performance of each combination was slightly decreased by updating. The combination of Doc2vec and MLP achieved an F1 score of 0.961.

Table 7
Performance on 2020–2021 samples.

		Accuracy	Precision	Recall	F1
One-hot	RF	0.955	0.922	0.994	0.957
	XGB	0.930	0.885	0.988	0.934
	MLP	0.956	0.922	0.996	0.957
LSI	RF	0.943	0.939	0.948	0.944
	XGB	0.939	0.910	0.974	0.941
	MLP	0.917	0.867	0.985	0.923
Doc2vec	RF	0.956	0.930	0.986	0.957
	XGB	0.947	0.912	0.989	0.949
	MLP	0.960	0.923	0.990	**0.961**

5. Discussion

5.1. Time complexity

Our time complexity analysis shows that each run time for building both language models increases linearly. Concerning training, each run time except SVM showed an almost linear increase. The runtime with SVM dramatically increased with the training sample size. While Doc2vec requires more time than LSI, the building model does not require real-time processing. Nevertheless, the run time with SVM is too considerable for a practical sample size.

The run time for testing all classifiers increased linearly. SVM requires more time than other classifiers. Although RF is generally a fast algorithm, run time slightly increased. This could be because both SVM and RF are implemented with scikit-learn. Thus, other classifiers appear to be sufficient for practical use.

5.2. Comparison

Our experimental results show that each classifier with a one-hot vector and Doc2vec provides a better accuracy. The combination of Doc2vec and MLP achieved an F1 score of 0.981. However, there is almost no difference between the performances of the one-hot vector and Doc2vec. One possible reason for this is that the training and test samples contained common tokens. Given that the dataset contains many distributed samples, these may be common features for classification. Each classifier with LSI provided a relatively worse accuracy. This may imply that the topics categorized by the LSI are not efficient for classification. These results are consistent with this observation. We analyzed the common features of training samples randomly divided into five parts and found 398 frequent tokens. Intriguingly, many frequent tokens (328 in 2020 and 326 in 2021) are contained in both benign and malicious samples in the test samples. Therefore, the specific tokens in malicious samples are not decisive features. Combinations of specific tokens appear to be effective for classification. Thus, specific tokens in strings can be used for classification.

In a previous study [1], the best F1 score was 0.934 using the FFRI 2019 dataset. In this study, the FFRI 2019 dataset was randomly divided into 10 groups. The same dataset was randomly divided into five groups for training in this experiment, and the FFRI 2020 and 2021 datasets were used for testing. Accordingly, the best F1 score was 0.981. Each score in the five groups was almost stable. Therefore, the training sample size appears to be efficient for classification.

5.3. Time series

To estimate the effect of the time series, the test samples were updated. Nevertheless, the performance remained steady with the passage of time. The training samples were updated. Contrary to our expectation, the performance slightly decreased with the update. From these results, the effect of the time series was limited in our method. Updating the training samples may not improve the detection accuracy. Thus, efficient training samples may build a robust classification model over time.

5.4. Practical use

Based on the experimental results, the most practical combination is Doc2vec and MLP, which achieved an F1 score of 0.981. While Doc2vec requires time to build a model, it does not require real-time processing. Each run time showed an almost linear increase. Thus, both models appear to be scalable for the actual sample size. The combination of one-hot and RF appears to be practical. Although RF is generally a fast algorithm, the run time slightly increased. This could be because the RF is implemented with scikit-learn. Other implementations may improve time complexity. Therefore, our method is suitable to implement in IoT devices of low computing resources. Our method can be applied to detecting IoT malware.

5.5. Limitations

Our study clearly has some limitations.

The first is attributed to the FFRI dataset. As previously described, 800,000 samples were used in this study. However, the actual samples may be more distributed. Therefore, the FFRI dataset may not appropriately represent the population. In fact, we cannot use all actual samples for evaluation. To the best of our knowledge, a possible solution is to use a large-scale dataset.

The second limitation is the lack of a comparison. In this study, we compared the proposed method with traditional techniques. However, we could not provide a fair comparison with other related studies due to implementation problems. Further experiments must provide a fair comparison with other studies.

6. Conclusion

In this study, we evaluated the method using the latest FFRI dataset consisting of 400,000 benign and 400,000 malicious samples between 2019 and 2021. We conducted several experiments considering the time series of both malicious and benign samples. Our experimental results show that specific tokens in printable characters are effective in detecting the latest malicious PE files. The effect of the time series seems to be limited in our method, and updating the training samples may not improve the detection accuracy. Efficient training samples may build a robust classification model over time. Finally, the most practical combination was Doc2vec and MLP, which achieved an F1 score of 0.981. Each run time showed an almost linear increase.

Our study had some limitations. Owing to the issues regarding the dataset or implementation, it was not feasible to provide a fair comparison with related studies. Further experiments are required to provide a fair comparison with related studies.

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.

Acknowledgment

This work was supported by JSPS, Japan KAKENHI, Japan Grant Number 21K11898.

References

[1] M. Mimura, R. Ito, Applying NLP techniques to malware detection in a practical environment, Int. J. Inf. Secur. (2021) http://dx.doi.org/10.1007/s10207-021-00553-8.

[2] J. Lee, C. Im, H. Jeong, A study of malware detection and classification by comparing extracted strings, in: Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2011, Seoul, Republic of Korea, February 21 - 23, 2011, 2011, p. 75, http://dx.doi.org/10.1145/1968613.1968704.

[3] M.G. Schultz, E. Eskin, E. Zadok, S.J. Stolfo, Data mining methods for detection of new malicious executables, in: 2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001, IEEE Computer Society, 2001, pp. 38–49, http://dx.doi.org/10.1109/SECPRI.2001.924286.

[4] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, M. Zhao, SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging, J. Comput. Virol. 5 (4) (2009) 283–293, http://dx.doi.org/10.1007/s11416-008-0108-y.

[5] B. Kolosnjaji, G. Eraisha, G.D. Webster, A. Zarras, C. Eckert, Empowering convolutional networks for malware classification and analysis, in: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017, IEEE, 2017, pp. 3838–3845, http://dx.doi.org/10.1109/IJCNN.2017.7966340.

[6] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, C. Kruegel, When malware is packin' heat; limits of machine learning classifiers based on static analysis features, in: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020, The Internet Society, 2020, URL https://www.ndss-symposium.org/ndss-paper/when-malware-is-packin-heat-limits-of-machine-learning-classifiers-based-on-static-analysis-features/.

[7] O. Henchiri, N. Japkowicz, A feature selection and evaluation scheme for computer virus detection, in: ICDM, IEEE Computer Society, 2006, pp. 891–895, URL http://www.computer.org/csdl/proceedings/icdm/2006/2701/00/index.html.

[8] A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer, Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey, Inf. Sec. Techn. Report 14 (1) (2009) 16–29.

[9] T. Abou-Assaleh, N. Cercone, V. Keselj, R. Sweidan, Detection of new malicious code using N-grams signatures, in: Second Annual Conference on Privacy, Security and Trust, October 13-15, 2004, Wu Centre, University of New Brunswick, Fredericton, New Brunswick, Canada, Proceedings, 2004, pp. 193–196, URL http://dev.hil.unb.ca/Texts/PST/pdf/assaleh.pdf.

[10] J.Z. Kolter, M.A. Maloof, Learning to detect and classify malicious executables in the wild, J. Mach. Learn. Res. 7 (2006) 2721–2744, URL http://jmlr.org/papers/v7/kolter06a.html.

[11] J.Z. Kolter, M.A. Maloof, Learning to detect malicious executables in the wild, in: W. Kim, R. Kohavi, J. Gehrke, W. DuMouchel (Eds.), Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004, ACM, 2004, pp. 470–478.

[12] B. Zhang, J. Yin, J. Hao, D. Zhang, S. Wang, Malicious codes detection based on ensemble learning, in: B. Xiao, L.T. Yang, J. Ma, C.M. Aller-Schloer, Y. Hua (Eds.), Autonomic and Trusted Computing, 4th International Conference, ATC 2007, Hong Kong, China, July 11-13, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4610, Springer, 2007, pp. 468–477.

[13] G.A. Jacob, P.M. Comparetti, M. Neugschwandtner, C. Kruegel, G. Vigna, A static, packer-agnostic filter to detect similar malware samples, in: U. Flegel, E.P. Markatos, W.K. Robertson (Eds.), DIMVA, in: Lecture Notes in Computer Science, vol. 7591, Springer, 2012, pp. 102–122.

[14] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, Y. Elovici, Unknown malcode detection via text categorization and the imbalance problem, in: ISI, IEEE, 2008, pp. 156–161.

[15] M. Zolotukhin, T. Hamalainen, Detection of zero-day malware based on the analysis of opcode sequences, in: 2014 IEEE 11th Consumer Communications and Networking Conference, (CCNC), 2014, pp. 386–391.
[16] M.E. Karim, A. Walenstein, A. Lakhotia, L. Parida, Malware phylogeny generation using permutations of code, J. Comput. Virol. 1 (1–2) (2005) 13–23.
[17] D. Bilar, Opcodes as predictor for malware, IJESDF 1 (2) (2007) 156–168.
[18] L. Martignoni, M. Christodorescu, S. Jha, OmniUnpack: Fast, generic, and safe unpacking of malware, in: ACSAC, IEEE Computer Society, 2007, pp. 431–441, URL http://www.computer.org/csdl/proceedings/acsac/2007/3060/00/index.html.
[19] D. Kong, G. Yan, Discriminant malware distance learning on structural information for automated malware classification, in: I.S. Dhillon, Y. Koren, R. Ghani, T.E. Senator, P. Bradley, R. Parekh, J. He, R.L. Grossman, R. Uthurusamy (Eds.), The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013, ACM, 2013, pp. 1357–1365, URL http://dl.acm.org/citation.cfm?id=2487575.
[20] R. Tian, L.M. Batten, S. Versteeg, Function length as a tool for malware classification, in: MALWARE, IEEE Computer Society, 2008, pp. 69–76, URL http://doi.ieeecomputersociety.org/10.1109/MALWARE.2008.4690860.
[21] I. Ismail, M.N. Marsono, S.M. Nor, Detecting worms using data mining techniques: Learning in the presence of class noise, in: K.Y. Atongnon, A. Dipanda, R. Chbeir (Eds.), Sixth International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2010, Kuala Lumpur, Malaysia, December 15-18, 2010, IEEE Computer Society, 2010, pp. 187–194, URL http://www.computer.org/csdl/proceedings/sitis/2010/4319/00/index.html.
[22] V.S. Sathyanarayan, P. Kohli, B. Bruhadeshwar, Signature generation and detection of malware families, in: Y. Mu, W. Susilo, J. Seberry (Eds.), Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008, Proceedings, in: Lecture Notes in Computer Science, vol. 5107, Springer, 2008, pp. 336–349.
[23] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, C. Glezer, Applying machine learning techniques for detection of malicious code in network traffic, in: J. Hertzberg, M. Beetz, R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, OsnabrÜCk, Germany, September 10-13, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4667, Springer, 2007, pp. 44–50, http://dx.doi.org/10.1007/978-3-540-74565-5_5.
[24] R. Perdisci, A. Lanzi, W. Lee, Classification of packed executables for accurate computer virus detection, Pattern Recognit. Lett. 29 (14) (2008) 1941–1946, http://dx.doi.org/10.1016/j.patrec.2008.06.016.
[25] B. Li, K.A. Roundy, C.S. Gates, Y. Vorobeychik, Large-scale identification of malicious singleton files, in: G. Ahn, A. Pretschner, G. Ghinita (Eds.), Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017, ACM, 2017, pp. 227–238, http://dx.doi.org/10.1145/3029806.3029815.
[26] T. Rezaei, F. Manavi, A. Hamzeh, A PE header-based method for malware detection using clustering and deep embedding techniques, J. Inform. Secur. Appl. 60 (2021) 102876, http://dx.doi.org/10.1016/j.jisa.2021.102876, URL https://www.sciencedirect.com/science/article/pii/S2214212621001046.
[27] M.Z. Shafiq, S.M. Tabish, F. Mirza, M. Farooq, PE-miner: Mining structural information to detect malicious executables in realtime, in: E. Kirda, S. Jha, D. Balzarotti (Eds.), Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009, Proceedings, in: Lecture Notes in Computer Science, vol. 5758, Springer, 2009, pp. 121–141, http://dx.doi.org/10.1007/978-3-642-04342-0_7.
[28] J. Saxe, K. Berlin, Deep neural network based malware detection using two dimensional binary program features, in: 10th International Conference on Malicious and Unwanted Software, MALWARE 2015, Fajardo, PR, USA, October 20-22, 2015, IEEE Computer Society, 2015, pp. 11–20, http://dx.doi.org/10.1109/MALWARE.2015.7413680.
[29] G.D. Webster, B. Kolosnjaji, C. von Pentz, J. Kirsch, Z.D. Hanif, A. Zarras, C. Eckert, Finding the needle: A study of the PE32 rich header and respective malware triage, in: M. Polychronakis, M. Meier (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment - 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings, in: Lecture Notes in Computer Science, vol. 10327, Springer, 2017, pp. 119–138, http://dx.doi.org/10.1007/978-3-319-60876-1_6.
[30] E. Raff, J. Sylvester, C. Nicholas, Learning the PE header, malware detection with minimal domain knowledge, in: B.M. Thuraisingham, B. Biggio, D.M. Freeman, B. Miller, A. Sinha (Eds.), Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017, ACM, 2017, pp. 121–132, http://dx.doi.org/10.1145/3128572.3140442.
[31] M. Mimura, H. Miura, Detecting unseen malicious VBA macros with NLP techniques, J. Inf. Process. 27 (2019) 555–563, http://dx.doi.org/10.2197/ipsjjip.27.555.
[32] M. Mimura, An improved method of detecting macro malware on an imbalanced dataset, IEEE Access 8 (2020) 204709–204717, http://dx.doi.org/10.1109/ACCESS.2020.3037330.
[33] H. Miura, M. Mimura, H. Tanaka, Macros finder: Do you remember LOVELETTER? in: C. Su, H. Kikuchi (Eds.), Information Security Practice and Experience - 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings, in: Lecture Notes in Computer Science, vol. 11125, Springer, 2018, pp. 3–18, http://dx.doi.org/10.1007/978-3-319-99807-7_1.
[34] M. Mimura, Using fake text vectors to improve the sensitivity of minority class for macro malware detection, J. Inf. Secur. Appl. 54 (2020) 102600, http://dx.doi.org/10.1016/j.jisa.2020.102600.
[35] Y. Tajiri, M. Mimura, Detection of malicious PowerShell using word-level language models, in: K. Aoki, A. Kanaoka (Eds.), Advances in Information and Computer Security - 15th International Workshop on Security, IWSEC 2020, Fukui, Japan, September 2-4, 2020, Proceedings, in: Lecture Notes in Computer Science, vol. 12231, Springer, 2020, pp. 39–56, http://dx.doi.org/10.1007/978-3-030-58208-1_3.
[36] M. Mimura, Y. Tajiri, Static detection of malicious PowerShell based on word embeddings, Internet Things 15 (2021) 100404, http://dx.doi.org/10.1016/j.iot.2021.100404, URL https://www.sciencedirect.com/science/article/pii/S2542660521000482.
[37] S. Ndichu, S. Kim, S. Ozawa, T. Misu, K. Makishima, A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors, Appl. Soft Comput. 84 (2019) http://dx.doi.org/10.1016/j.asoc.2019.105721.
[38] P.M. Ngoc, M. Mimura, Detection of malicious javascript on an imbalanced dataset, Internet Things 13 (2021) 100357, http://dx.doi.org/10.1016/j.iot.2021.100357.
[39] M. Mimura, T. Ohminami, Towards efficient detection of malicious VBA macros with LSI, in: N. Attrapadung, T. Yagi (Eds.), Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings, in: Lecture Notes in Computer Science, vol. 11689, Springer, 2019, pp. 168–185, http://dx.doi.org/10.1007/978-3-030-26834-3_10.
[40] M. Mimura, T. Ohminami, Using LSI to detect unknown malicious VBA macros, J. Inf. Process. 28 (2020) 493–501, http://dx.doi.org/10.2197/ipsjjip.28.493.
[41] M. Mimura, H. Tanaka, Heavy log reader: Learning the context of cyber attacks automatically with paragraph vector, in: R.K. Shyamasundar, V. Singh, J. Vaidya (Eds.), Information Systems Security - 13th International Conference, ICISS 2017, Mumbai, India, December 16-20, 2017, Proceedings, in: Lecture Notes in Computer Science, vol. 10717, Springer, 2017, pp. 146–163, http://dx.doi.org/10.1007/978-3-319-72598-7_9.
[42] M. Mimura, H. Tanaka, Leaving all proxy server logs to paragraph vector, J. Inf. Process. 26 (2018) 804–812, http://dx.doi.org/10.2197/ipsjjip.26.804.
[43] M. Mimura, Adjusting lexical features of actual proxy logs for intrusion detection, J. Inf. Secur. Appl. 50 (2020) http://dx.doi.org/10.1016/j.jisa.2019.102408.
[44] M. Mimura, H. Tanaka, Reading network packets as a natural language for intrusion detection, in: H. Kim, D. Kim (Eds.), Information Security and Cryptology - ICISC 2017 - 20th International Conference, Seoul, South Korea, November 29 - December 1, 2017, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 10779, Springer, 2017, pp. 339–350, http://dx.doi.org/10.1007/978-3-319-78556-1_19.
[45] M. Mimura, An attempt to read network traffic with doc2vec, J. Inf. Process. 27 (2019) 711–719, http://dx.doi.org/10.2197/ipsjjip.27.711.
[46] M. Hatada, M. Akiyama, T. Matsuki, T. Kasama, Empowering anti-malware research in Japan by sharing the MWS datasets, J. Inf. Process. 23 (5) (2015) 579–588, http://dx.doi.org/10.2197/ipsjjip.23.579.