International Journal of Information Security (2022) 21:279–291
https://doi.org/10.1007/s10207-021-00553-8

REGULAR CONTRIBUTION

Applying NLP techniques to malware detection in a practical environment

Mamoru Mimura[1] · Ryo Ito[2]

Published online: 6 June 2021

Abstract
Executable files still remain popular to compromise the endpoint computers. These executable files are often obfuscated to avoid anti-virus programs. To examine all suspicious files from the Internet, dynamic analysis requires too much time. Therefore, a fast filtering method is required. With the recent development of natural language processing (NLP) techniques, printable strings became more effective to detect malware. The combination of the printable strings and NLP techniques can be used as a filtering method. In this paper, we apply NLP techniques to malware detection. This paper reveals that printable strings with NLP techniques are effective for detecting malware in a practical environment. Our dataset consists of more than 500,000 samples obtained from multiple sources. Our experimental results demonstrate that our method is effective to not only subspecies of the existing malware, but also new malware. Our method is effective against packed malware and anti-debugging techniques.

Keywords Malware · Machine learning · Natural language processing

1 Introduction

Targeted attacks are one of the serious threats through the Internet. The standard payload such as a traditional executable file has still been remained popular. According to a report, executable file is the second of the top malicious email attachment types [49]. Attackers often use obfuscated malware to evade anti-virus programs. Pattern matching-based detection methods such as anti-virus programs barely detect new malware [28]. To detect new malware, automated dynamic analysis systems and sandboxes are effective. The idea is to force any suspicious binary to execute in sandboxes, and if their behaviors are malicious, then the file is classified as malware. While dynamic analysis is a powerful method, it requires too much time to examine all suspicious files from the Internet. Furthermore, it requires high-performance servers and their license including commercial OS and applications. Therefore, a fast filtering method is required.

In order to achieve this, static detection methods with machine learning techniques can be applicable. These methods extract features from the malware binary and Portable Executable (PE) header. While the printable strings are often analyzed, they were not a decisive element for detection. With the recent development of natural language processing (NLP) techniques, the printable strings became more effective to detect malware [8]. Therefore, the combination of the printable strings and NLP techniques can be used as a filtering method.

In this paper, we apply NLP techniques to malware detection. This paper reveals that printable strings with NLP techniques are effective for detecting malware in a practical environment. Our time series dataset consists of more than 500,000 samples obtained from multiple sources. Our experimental result demonstrates that our method can detect new malware. Furthermore, our method is effective against packed malware and anti-debugging techniques. This paper produces the following contributions.

- Printable strings with NLP techniques are effective for detecting malware in a practical environment.
- Our method is effective to not only subspecies of the existing malware, but also new malware.

✉ Mamoru Mimura
mim@nda.ac.jp

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

[2] Japan Ground Self-Defense Force 5-1 Honmura-cho, Ichigaya, Shinjuku-ku, Tokyo, Japan

– Our method is effective against packed malware and anti-debugging techniques.

The structure of this paper is as follows. Section 2 describes related studies. Section 3 provides the natural language processing techniques related to this study. Section 4 describes our NLP-based detection model. Section 5 evaluates our model with the dataset. Section 6 discusses the performance and research ethics. Finally, Section 7 concludes this study.

2 Related work

2.1 Static analysis

Malware detection methods are categorized into dynamic analysis and static analysis. Our detection model is categorized into static analysis. Hence, this section focuses on the features used in static analysis.

One of the most common features is byte n-gram features extracted from malware binary [47]. Abou-Assaleh et al. used the frequent n-grams to generate signatures from malicious and benign samples [1]. Kolter et al. used information gain to extract 500 n-grams features [12,13]. Zhang et al. also used information gain to select important n-gram features [55]. Henchiri et al. conducted an exhaustive feature search on a set of malware samples and strived to obviate overfitting [6]. Jacob et al. used bigram distributions to detect similar malware [9]. Raff et al. applied neural networks to raw bytes without explicit feature construction [41]. Similar approaches are extracting features from the opcode n-grams [3,10,35,57] or program disassembly [7,14,18,44,50].

While many studies focus on the body of malware, several studies focus on the PE headers. Shafiq et al. proposed a framework that automatically extracts 189 features from PE headers [48]. Perdisci et al. used PE headers to distinguish between packed and non-packed executables [38]. Elovici et al. used byte n-grams and PE headers to detect malicious code [4]. Webster et al. demonstrated how the contents of the PE files can help to detect different versions of malware [53]. Saxe et al. used a histogram of byte entropy values, DLL imports, and numerical PE fields with neural networks [45]. Li et al. extracted top of features from PE headers and sections with a recurrent neural network (RNN) model [17]. Raff et al. used raw byte sequences obtained from PE headers with a Long Short-Term Memory (LSTM) network [40].

Other features are also used to detect malware. Schultz et al. used n-grams, printable strings, and DLL imports with machine learning techniques for malware detection [46]. Masud et al. used byte n-grams, assembly instructions, and DLL function calls [20]. Ye et al. used interpretable strings such as API execution calls and important semantic strings [54]. Lee et al. focused on the similarity between two files to identify and classify malware [16]. The similarity is calculated from the extracted printable strings. Mastjik et al. analyzed string matching methods to identify the same malware family [19]. Their method used 3 pattern matching algorithms, Jaro, Lowest Common Subsequence (LCS), and n-grams. Kolosnjaji et al. proposed a method to classify malware with neural network that consists of convolutional and feedforward neural constructs [11]. Their model extracts feature from the n-grams of instructions, and the headers of executable files. Aghakhani et al. studied how machine learning-based on static analysis features operates on packed samples [2]. They used a balanced dataset with 37,269 benign samples and 44,602 malicious samples.

Thus, several studies used the printable strings as features. However, the printable strings are not used as the main method for detection. The combination of the printable strings and NLP techniques are not evaluated in a practical environment. This paper pursues the possibility of the printable strings as a filtering method.

2.2 NLP-based detection

Our detection model uses some NLP techniques. This section focuses on the NLP-based detection methods.

Moskovitch et al. used some NLP techniques such as Term Frequency-Inverse Document Frequency (TF-IDF) to represent byte n-gram features [35]. Nagano et al. proposed a method to detect malware with Paragraph Vector [36]. Their method extracts the features from the DLL Import name, assembly code, and hex dump. A similar approach is classifying malware from API sequences with TF-IDF and Paragraph Vector [51]. This method requires dynamic analysis to extract API sequences. Thus, the printable strings are not used as the main method for detection.

NLP-based detection was applied to detect malicious traffic and other contents. Paragraph Vector was applied to extract the features of proxy logs [30,32]. This method was extended to analyze network packets [22,31]. To mitigate class imbalance problems, the lexical features are adjusted by extracting frequent words [23,33]. Our method uses this technique mitigate class imbalance problems. Some methods use a Doc2vec model to detect malicious JavaScript code [29,37,39]. Other methods use NLP techniques to detect memory corruptions [52] or malicious VBA macros [24–27,34].

3 NLP techniques

This section describes some NLP techniques related to this study. The main topic of NLP techniques is to enable computers to process and analyze large amounts of natural

language data. The documents written in natural language are separated into words to apply NLP techniques such as Bag-of-Words. The corpus of words is converted into vectors which can be processed by computers.

3.1 Bag-of-words

Bag-of-Words (BoW) [43] is a fundamental method of document classification where the frequency of each word is used as a feature for training a classifier. This model converts each word in a document into vectors based on the frequency of each word. Let d, w, and n be expressed as a document, word ($w_{i=1,2,3,...}$), and a frequency of w, respectively. The document d can be defined by Eq. (1). For this Eq. (1), next (2) locks the position of n on d, and omitted the word w. This $\hat{d}_j$ ($\hat{d}_{j=1,2,3,...}$) is shown as a vector (document-word matrix). In Eq. (3), let construct the other documents to record the term frequencies of all the distinct words (other documents ordered as in Eq. (2)).

$$d = [(w_1, n_{w_1}), (w_2, n_{w_2}), (w_3, n_{w_3}), ..., (w_i, n_{w_i})] \quad (1)$$

$$\hat{\mathbf{d}}_\mathbf{j} = (n_{w_1}, n_{w_2}, n_{w_3}, ..., n_{w_i}) \quad (2)$$

$$|D| = \begin{bmatrix} n_{w_{1,1}} \ldots n_{w_{1,i}} \\ \vdots \\ n_{w_{j,1}} \ldots n_{w_{j,i}} \end{bmatrix}. \quad (3)$$

Thus, this model enables to convert documents into vectors. Apparently, this model does not preserve the order of the words in the original documents. The dimension of converted vectors attains the number of distinct words in the original documents. To analyze large-scale data, the dimension should be reduced so that can be analyzed in a practical time.

3.2 Latent semantic indexing

Latent Semantic Indexing (LSI) analyzes the relevance between a document group and words included in a document. In the LSI model, the vectors with BoW are reduced by singular value decomposition. Each component of the vectors is weighted. The decomposed matrix shows the relevance between the document group and words included in the document. In weighting each component of the vector, Term Frequency-Inverse Document Frequency (TF-IDF) is usually used. $|D|$ is the total number of documents, $\{d : d \ni t_i\}$ is the total document including word i, $frequency_{i,j}$ is the appearance frequency of word i in document j. TF-IDF is defined by Eq. (4).

$$tf_{i,j} * idf_i = frequency_{i,j} * \log \frac{|D|}{\{d : d \ni t_i\}} \quad . \quad (4)$$

TF-IDF weights the vector to perform singular value decomposition. The components $x_(i, j)$ of the matrix X show the TF-IDF value in the document j of the word i. Let X be decomposed into orthogonal matrices U and V and diagonal matrix Σ, from the theory of linear algebra. In this singular value decomposition, U is a column orthogonal matrix and linearly independent with respect to the column vector. Therefore, U is the basis of the document vector space. The matrix X product giving the correlation between words and documents is expressed by the following determinant. Generally, this matrix U represents a latent meaning.

$$X = \begin{bmatrix} x_{1,1} \ldots x_{1,j} \\ \vdots \ddots \vdots \\ x_{i,1} \ldots x_{i,j} \end{bmatrix} = U \Sigma V^T$$

$$= \begin{bmatrix} u_{1,1} \ldots u_{1,r} \\ \vdots \ddots \vdots \\ u_{i,1} \ldots u_{i,r} \end{bmatrix} * \begin{bmatrix} \sigma_{1,1} \ldots & 0 \\ \vdots \ddots & \vdots \\ 0 & \ldots \sigma_{r,r} \end{bmatrix} * \begin{bmatrix} v_{1,1} \ldots v_{1,j} \\ \vdots \ddots \vdots \\ v_{r,1} \ldots v_{r,j} \end{bmatrix}.$$

In this model, the number of dimension can be determined arbitrarily. Thus, this model enables to reduce the dimension so that can be analyzed in a practical time.

3.3 Paragraph vector

To represent word meaning or context, Word2vec was created [21]. Word2vec is shallow neural networks which are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of documents and produces a vector space of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words which share common contexts in the corpus are located in close proximity to one another in the space. $queen = king - man + woman$ is an example of operation using each word vector generated by Word2vec. Word2vec is a method to represent a word with meaning or context. Paragraph Vector is the extension of Word2vec to represent a document [15]. Doc2vec is an implementation of the Paragraph Vector. The only change is replacing a word into a document ID. Words could have different meanings in different contexts. Hence, vectors of two documents which contain the same word in two distinct senses need to account for this distinction. Thus, this model represents a document with word meaning or context.

4 NLP-based detection model

This section produces our detection model based on the previous study [8]. The previous study used an SVM model to classify samples. The main revised point is adding several

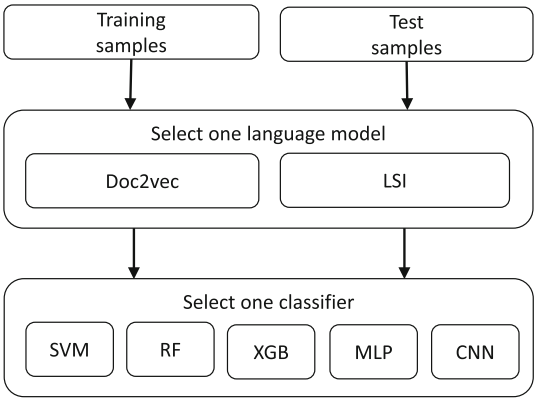

Fig. 1 Structure of the NLP-based detection model 4

classifiers. The structure of our detection model is shown in Fig. 1.

Our detection model consists of language models and classifiers. One model is selected from these models, respectively. In training phase, the selected language model is constructed with extracted words from malicious and benign samples. The constructed language model extracts the lexical features. The selected classifier is trained with the lexical features and labels. In testing phase, the constructed language model and trained classifier classify unknown executable files into malicious or benign ones.

4.1 Training

The training procedure is shown in Algorithm 1. Our method extracts all printable (ASCII) strings from malicious and benign samples and splits the strings into words, respectively. The frequent words are selected from the words, respectively. Thenceforth, the selected language model is constructed from the selected words. Our method uses a Doc2vec or LSI model to represent the lexical features. The Doc2vec model is constructed by the corpus of the words. The LSI model is constructed from the TF-IDF scores of the words. These words are converted into lexical features with the selected language model. Thus, the labeled feature vectors are derived. Thereafter, the selected classifier is trained with the both labeled feature vectors. The classifiers are Support Vector Machine (SVM), Random Forests (RF), XGBoost (XGB), Multi-Layer Perceptron (MLP), and Convolutional Neural Networks (CNN). These classifiers are popular in the various fields, and have each characteristic.

4.2 Test

The test procedure is shown in Algorithm 2. Our method extracts printable strings from unknown samples, and splits the strings into words. The extracted words are converted into lexical features with the selected language model, which was constructed in training phase. The trained classifier examines the feature vectors and predicts the labels.

Algorithm 1 training

```
/* Extract printable strings */
for all malware samples m do
    mw ← extract printable strings from m
end for
for all benign samples b do
    bw ← extract printable strings from b
end for
imw ← select frequent words from m
ibw ← select frequent words from b
if Doc2vec then
    /* Construct a Doc2vec model */
    construct a Doc2vec model from imw, ibw
else
    /* Construct a LSI model */
    construct a TF-IDF model from imw, ibw
    construct a LSI model from the TF-IDF
end if
/* Convert printable strings into vectors */
for all malware samples mw do
    if Doc2vec then
        mv ← Doc2vec(mw)
    else
        mv ← LSI(mw)
    end if
end for
for all benign samples bw do
    if Doc2vec then
        bv ← Doc2vec(bw)
    else
        bv ← LSI(bw)
    end if
end for
/* Train classifiers with the labeled vectors */
if SVM then
    Train SVM(mv, bv)
else if RF then
    Train RF(mv, bv)
else if XGB then
    Train XGB(mv, bv)
else if MLP then
    Train MLP(mv, bv)
else
    Train CNN(mv, bv)
end if
return
```

4.3 Implementation

Our detection model is implemented in Python 2.7. Gensim [42] provides the LSI and Doc2vec models. Scikit-learn [1] provides the SVM and RF classifiers. The XGB is provided as a Python package [2]. The MLP and CNN are implemented

[1] https://scikit-learn.org/.

[2] https://xgboost.readthedocs.io/.

Algorithm 2 test

```
/* Extract printable strings */
for all unknown samples u do
    uw ← extract printable strings from u
end for
/* Convert printable strings into vectors */
for all unknown samples uw do
    if Doc2vec then
        uv ← Doc2vec(uw)
    else
        uv ← LSI(uw)
    end if
end for
/* Predict labels with the trained classifiers */
for all unknown vectors uv1 do
    if SVM then
        label ← SVM(uv)
    else if RF then
        label ← RF(uv)
    else if XGB then
        label ← XGB(uv)
    else if MLP then
        label ← MLP(uv)
    else
        label ← CNN(uv)
    end if
end for
return
```

with chainer [3]. The parameters will be optimized in the next section.

5 Evaluation

5.1 Dataset

To evaluate our detection model, hundred thousands of PE samples were obtained from multiple sources. One is the FFRI dataset, which is a part of MWS datasets [5]. This dataset contains logs collected from the dynamic malware analysis system Cuckoo sandbox [4] and a static analyzer. This dataset is written in JSON format, and categorized into 2013 to 2019 based on the obtained year. Each data except 2018 contains printable strings extracted from malware samples. These data can be used as malicious samples (FFRI 2013–2017, 2019). Note that this dataset does not contain malware samples themselves. Printable strings extracted from benign samples are contained in 2019's as Cleanware. These benign data do not contain the time stamps. Hence, we randomly categorized these data into 3 groups (Clean A, B, and C). Other samples are obtained from Hybrid Analysis (HA) [5], which is a popular malware distribution site. Almost ten thousand of samples are obtained with our web crawler. These samples are posted into the VirusTotal [6], and are identified by multiple anti-virus programs. Thereafter, the identified samples are categorized into 2016 to 2018 based on the first year defined by Microsoft defender. The printable strings are extracted from these malware samples (HA 2016–2018). To extract printable strings from these samples, we use the strings command on Linux. This command provides each string on one line. Our method uses these strings as word. Our extraction method is identical to the FFRI dataset [7]. Thus, our dataset is constructed with the multiple sources.

Table 1 The number of each data, unique words, and family name

Class	Dataset	File	Unique words	Family
	Clean A	10,000	39,390,887	–
benign	Clean B	40,000	122,217,004	–
(Cleanware)	Clean C	200,000	269,112,345	–
	FFRI 2013	2,637	1,317,912	621
	FFRI 2014	3,000	5,193,423	671
	FFRI 2015	3,000	3,224,583	474
	FFRI 2016	8,243	11,443,274	629
Malicious	FFRI 2017	6,251	1,534,580	394
	FFRI 2019	250,000	320,177,723	3237
	HA 2016	5,787	13,352,423	1,063
	HA 2017	2,834	8,944,685	181
	HA 2018	5,623	15,584,544	243

Table 1 shows the number of each data, unique words, and family name. Tables 2 and 3 show the top family names.

The unique words indicate the number of distinct words extracted from the whole dataset. In the NLP-based detection method, the number of unique words is important. Because the classification difficulty and complexity mainly depend on the number. The family indicates the number of distinct malware family defined by Microsoft defender. In benign samples, each dataset contains huge number of unique words. This means these benign samples are well distributed and not biased. In malicious samples, each dataset contains enough number of unique words and malware families. This suggests they contain not only subspecies of the existing malware, but also new malware. Hence, these samples are well distributed and not biased.

5.2 Metrics

Several metrics are used to evaluate our detection model. These metrics are based on the confusion matrix shown in Table 4.

[3] https://chainer.org/.

[4] https://cuckoosandbox.org/.

[5] https://www.hybrid-analysis.com/.

[6] https://www.virustotal.com/.

[7] https://github.com/FFRI/ffridataset-scripts.

Table 2 Top 5 family names in the FFRI dataset

Dataset	No.	Family name
FFRI 2013	1	Trojan:Win32/Bulta!rfn
	2	Worm:Win32/Vobfus.GZ
	3	Worm:Win32/Vobfus.CF
	4	Worm:Win32/Vobfus
	5	Trojan:Win32/Lethic.B
FFRI 2014	1	SoftwareBundler:Win32/Ogimant
	2	Trojan:Win32/Bulta!rfn
	3	Backdoor:Win32/Kelihos.F
	4	Trojan:Win32/Dacic.A!rfn
	5	Worm:MSIL/Mofin.A
FFRI 2015	1	Trojan:Win32/Dynamer!ac
	2	PWS:Win32/Zbot
	3	Trojan:Win32/Bulta!rfn
	4	Trojan:Win32/Bagsu!rfn
	5	Trojan:Win32/Miuref.F
FFRI 2016	1	Trojan:Win32/Dynamer!ac
	2	Trojan:Win32/Skeeyah.A!rfn
	3	TrojanSpy:Win32/Ursnif.HN
	4	VirTool:Win32/Injector.FQ
	5	Trojan:Win32/Bagsu!rfn
FFRI 2017	1	Ransom:Win32/Cerber
	2	Backdoor:Win32/Fynloski.A
	3	Ransom:Win32/Cerber!rfn
	4	Trojan:Win32/Dynamer!ac
	5	Worm:Win32/Pykspa!rfn
FFRI 2019	1	Trojan:Win32/Occamy.C
	2	Trojan:Win32/Tiggre!rfn
	3	Virus:Win32/Morefi.A
	4	Trojan:Win32/Skeeyah.A!rfn
	5	Trojan:Win32/Toga!rfn

Table 3 Top 10 family names in the HA dataset

Dataset	No.	Family name
HA 2016	1	PUA:Win32/InstallCore
	2	Backdoor:MSIL/Bladabindi.B
	3	Backdoor:MSIL/Bladabindi
	4	Trojan:Win32/Skeeyah.A!bit
	5	PUA:Win32/Keygen
	6	Trojan:Win32/Skeeyah.A!rfn
	7	Backdoor:MSIL/Noancooe.A
	8	Backdoor:MSIL/Bladabindi.AJ
	9	SoftwareBundler:Win32/Prepscram
	10	Trojan:Win32/Fuery.B!cl
HA 2017	1	Trojan:Win32/Tiggre!plock
	2	Trojan:Win32/Tiggre!rfn
	3	Trojan:Win32/Fuerboos.C!cl
	4	Trojan:Win32/Dynamer!rfn
	5	Trojan:Win32/Bluteal!rfn
	6	Trojan:Win32/Fuerboos.E!cl
	7	Trojan:Win32/Bitrep.A
	8	Trojan:Win32/Fuerboos.A!cl
	9	Ransom:Win32/WannaCrypt.A!rsm
	10	Trojan:Win32/MereTam.A
HA 2018	1	Trojan:Win32/Occamy.C
	2	PUA:Win32/Presenoker
	3	Ransom:Win32/CVE-2017-0147.A
	4	Trojan:Win32/Zpevdo.B
	5	Trojan:Win32/Zpevdo.A
	6	Trojan:Win32/Sonbokli.A!cl
	7	Trojan:Win32/CryptInject
	8	Trojan:Win32/Casdet!rfn
	9	Trojan:Win32/Meterpreter.O
	10	Trojan:Win32/Emotet.AC!bit

In this experiment, Accuracy (A), Precision (P), Recall (R), and F1 score (F) are mainly used. These metrics are defined as Eqs. (5)–(8).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2Recall * Precision}{Recall + Precision}. \tag{8}$$

In this experiment, TP indicates predicting malicious samples correctly. Since our detection model performs binary classification, Receiver Operating Characteristics (ROC) curve and Area under the ROC Curve (AUC) are used. An

Table 4 Confusion matrix

		Actual Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

ROC curve is a graph showing the performance of a classification model at all classification thresholds. AUC measures the entire two-dimensional area underneath the entire ROC curve.

5.3 Parameter optimization

To optimize the parameters of our detection model, the Clean A–B and FFRI 2013–2016 are used. The Clean A and FFRI

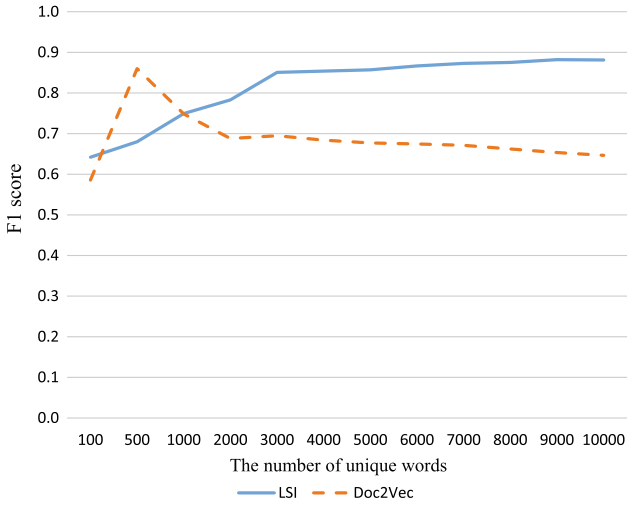

Fig. 2 The F1 score for each model

Table 5 The optimized parameters in each language model

Language model	Parameter	Optimum value
	Dimension	400
	alpha	0.075
Doc2vec	min_count	2
	window	1
	epoch	20
LSI	Dimension	800

Table 6 The optimized parameters in each classifier

Classifier	Parameter	Optimum value	
		Doc2vec	LSI
	kernel	rbf	rbf
SVM	C	100	100
	gamma	0.01	0.1
RF	n_estimators	393	442
	n_jobs	14	32
	n estimators	998	911
	max_depth	11	3
XGB	min_child_weight	11	5
	subsample	0.5	0.8
	colsample_bytree	0.7	0.6
MLP	optimizer	Adam	Adam
	epoch	40	40
CNN	optimizer	Adam	Adam
	epoch	40	40

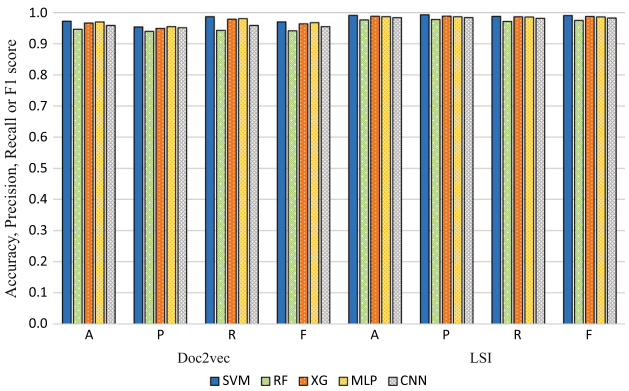

Fig. 3 The result of tenfold cross-validation

2013–2015 are used as the training samples. The remainder Clean B and FFRI 2016 are used as the test samples.

First, the number of unique words is optimized. To construct a language model, our detection model selects frequent words from each class. The same number of frequent words from each class are selected. This process adjusts the lexical features and enables to mitigate class imbalance problems [23]. The F1 score for each model is shown in Fig. 2.

In this figure, the vertical axis represents the F1 score, and the horizontal axis represents the total number of unique words. In the Doc2vec model, the most optimum number of the unique words is 500. In the LSI model, the F1 score gradually rises and achieves the maximum value at 9000.

Thereafter, the other parameters are optimized by grid search. Grid search is a search technique that has been widely used in many machine learning researches. The optimized parameters are shown in Tables 5 and 6.

Thus, our detection model uses these parameters in the remaining experiments.

5.4 Cross-validation

To evaluate the generalization performance, tenfold cross-validation is performed on the Clean A and FFRI 2013–2015. Figure 3 shows the result.

The vertical axis represents the Accuracy (A), Precision (P), Recall (R), or F1 score (F). Overall, each metric performed good accuracy. The LSI model was more effective than the Doc2vec model. Thus, the generalization performance of our detection model was almost perfect.

5.5 Time series analysis

To evaluate the detection rate (recall) for new malware, the time series is important. The purpose of our method is detecting unknown malware. In practical use, the test samples should not contain the earlier samples. To address this problem, we consider the time series of samples. In this

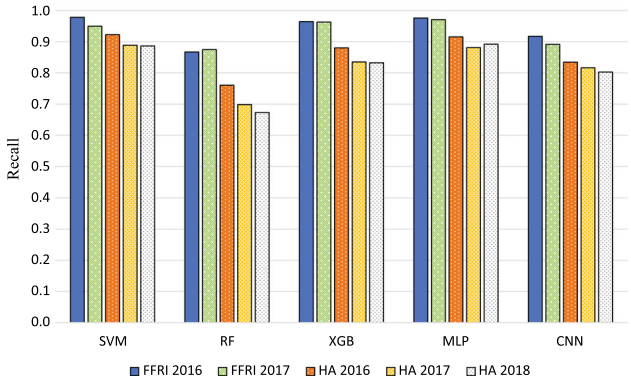

Fig. 4 The result of Doc2vec in the time series analysis

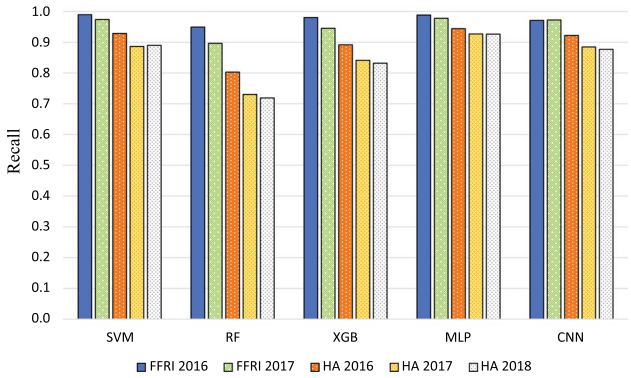

Fig. 5 The result of LSI in the time series analysis

experiment, the Clean A and FFRI 2013–2015 are used as the training samples. The Clean B, FFRI 2016–2017, and HA 2016–2018 are used as the test samples. As described in Table 1, the training samples are selected from the earlier ones. Moreover, the benign samples account for the majority of the test samples. This means the test samples are imbalanced, which represent more practical environment. Thus, the combination of each dataset is more challenging than cross-validation. The results of the time series analysis are shown in Figs. 4 and 5.

The vertical axis represents the recall. Overall, the recall gradually decreases as time proceeds. The detection rates in the FFRI dataset are better than the ones in the HA dataset. This seems to be because the training samples were obtained from the same source. Nonetheless, the recall in HA maintains almost 0.9. Note that these samples were identified by VirusTotal and categorized based on the first defined year. The LSI model was more effective than the Doc2vec model. In regard to classifiers, the SVM and MLP performed good accuracy. Thus, our detection model is effective against new malware. Furthermore, the same technique [23] mitigates class imbalance problems in executable files.

To visualize the relationship between sensitivity and specificity, the ROC curves of each model with FFRI 2016 are depicted in Figs. 6 and 7.

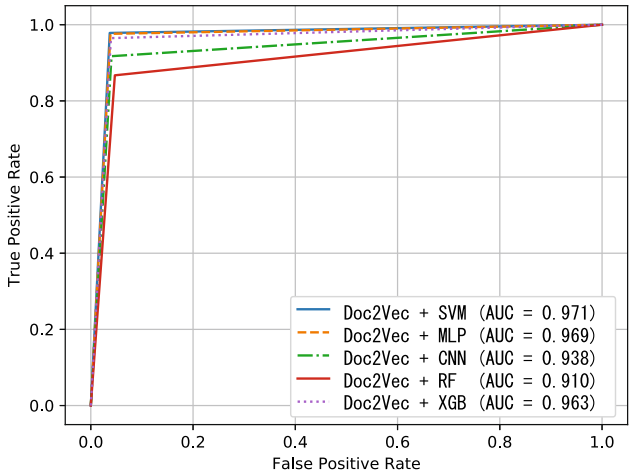

Fig. 6 The ROC curve of Doc2vec in the time series analysis

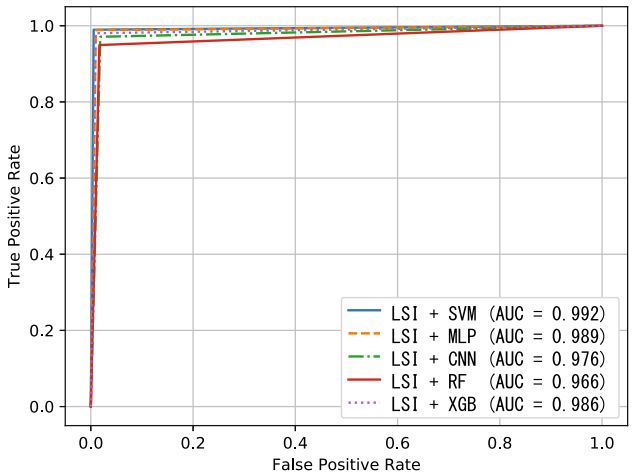

Fig. 7 The ROC curve of LSI in the time series analysis

The vertical axis represents the true positive rate (recall), and the horizontal axis represents the false positive rate. Our detection model maintains the practical performances with a low false positive rate. As we expected, the LSI model was more effective than the Doc2vec model. The best AUC score achieves 0.992 with the LSI and SVM model.

The required time for training and test of FFRI 2016 is shown in Table 7.

This experiment was conducted on the computer with Windows 10, Core i7-5820K 3.3GHz CPU, 32GB DDR4 memory, and Serial ATA 3 HDD. In regard to training time, it seems to depend on the classifier. Complicated classifiers such as CNN require more time for training. The test time maintains flat regardless of the classifier. The time to classify a single file is almost 0.1s. This speed seems to be enough to examine all suspicious files from the Internet.

Table 7 The required time for training and test of FFRI 2016

Classifier	SVM	MLP	CNN	RF	XGB
model	Doc2vec				
training	7:07	12:19	27:59	7:02	23:40
test	15:18	15:11	15:19	15:11	15:11
test (s/file)	0.111	0.111	0.111	0.111	0.111
model	LSI				
training	5:45	13:44	1:41:49	6:06	15:44
test	15:26	15:20	15:36	15:20	15:20
test (s/file)	0.112	0.112	0.114	0.112	0.112

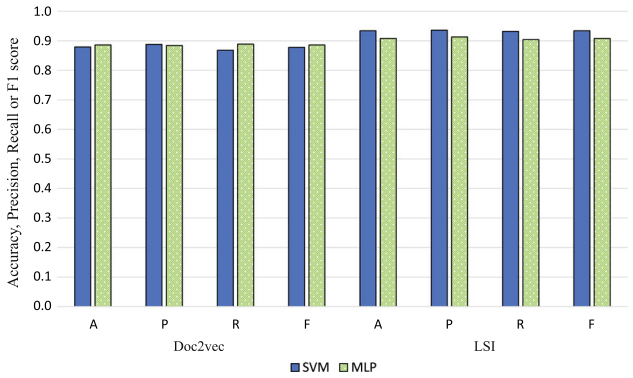

Fig. 8 The average result of the practical experiment

5.6 Practical performance

In practical environment, actual samples are more distributed. Hence, the experimental samples might not represent the population appropriately. To mitigate this problem, a more large-scale dataset has to be used. Moreover, the training samples should be smaller. To represent actual sample distribution, the FFRI 2019 and Clean A–C are used. They contain 500,000 samples. These samples are randomly divided into 10 groups. One of the groups is used as the training samples. The rest 9 groups are used as the test samples. The training and test are repeated 10 times. The average result of the practical experiment is shown in Fig. 8.

The vertical axis represents the Accuracy (A), Precision (P), Recall (R), or F1 score (F). Note that the training samples account for only 10 percent. This means the dataset is highly imbalanced. The LSI and SVM are the best combination. The best F1 score achieves 0.934. Thus, our detection model is effective in practical environment.

Table 8 Detection rate (recall) of the known and unknown malware (LSI and SVM)

	FFRI 2016		FFRI 2017		HA 2016	
	file	DR	file	DR	file	DR
Known	6,293	0.990	2,915	0.985	2,838	0.892
Unknown	1,950	0.989	3,336	0.964	2,944	0.967

6 Discussion

6.1 Detecting new malware

In the time series analysis, our detection model is effective to new malware on the imbalanced dataset. The new malware samples are categorized into known malware and unknown malware. In this study, we assume that known malware samples are ones previously defined by Microsoft defender. These samples are new but just subspecies of the existing malware. We also assume that unknown malware samples are ones not defined by Microsoft defender at that time. In this experiment, our detection model was trained by the samples before 2015. Hence, the newly defined samples after 2016 are assumed as new malware. The detection rate of the known and unknown malware is shown in Table 8.

The detection rate of unknown malware is on the same level with known malware. Thus, our detection model is effective to not only subspecies of the existing malware, but also new malware.

6.2 Defeating packed malware and anti-debugging techniques

Our detection model uses lexical features extracted from printable strings. These features include the API or argument names, which are useful to identify malware. These useful strings, however, are obfuscated in the packed malware. Several anti-debugging techniques might vary the lexical features. The test samples of the time series analysis are categorized into 4 types; packed and unpacked, or anti-debugging and no anti-debugging by PEiD [8]. PEiD detects most common packers and anti-debugging techniques with more than 470 different signatures in PE files. Since the FFRI dataset does not contain malware samples, we analyzed the HA dataset. Table 9 shows the detection rate of each malware type. Tables 10 and 11 show the top names.

Contrary to expectations, each detection rate is on the same level. We also analyzed the benign samples. These samples contain 27,196 packed samples and 77,893 samples with anti-debugging techniques. Detection rate in each type is 0.988 to 0.997. Therefore, our method does not seem to

[8] https://www.aldeid.com/wiki/PEiD.

Table 9 Detection rate (recall) of each malware type (LSI and SVM)

	HA 2016		HA 2017		HA 2018	
	file	DR	file	DR	file	DR
packed	2,771	0.926	1,370	0.866	2,466	0.878
unpacked	3,016	0.931	1,464	0.899	3,157	0.889
anti	1,404	0.912	905	0.899	1,929	0.906
no anti	4,383	0.934	1,929	0.874	3,694	0.871

Table 10 Top 10 PEiD names in the HA dataset

Dataset	No.	PEiD name
	1	Microsoft_Visual_Cpp_v50v60_MFC
	2	Borland_Delphi_30_additional
	3	Borland_Delphi_30_
	4	Borland_Delphi_v40_v50
	5	Borland_Delphi_v30
HA 2016	6	Borland_Delphi_DLL
	7	Borland_Delphi_40_additional
	8	Borland_Delphi_Setup_Module
	9	Borland_Delphi_40
	10	yodas_Protector_v1033_dllocx _Ashkbiz_Danehkar_h
	1	Microsoft_Visual_Cpp_8
	2	VC8_Microsoft_Corporation
	3	Microsoft_Visual_Cpp_v50v60_MFC
	4	yodas_Protector_v1033_dllocx _Ashkbiz_Danehkar_h
HA 2017	5	Microsoft_Visual_Studio_NET
	6	Microsoft_Visual_Studio_NET_additional
	7	NET_executable_
	8	NET_executable
	9	Microsoft_Visual_Basic_v50
	10	Microsoft_Visual_C_Basic_NET
	1	Microsoft_Visual_Cpp_v50v60_MFC
	2	Borland_Delphi_30_additional
	3	Borland_Delphi_30_
	4	Borland_Delphi_v40_v50
	5	Borland_Delphi_v30
HA 2018	6	Borland_Delphi_40_additional
	7	Borland_Delphi_Setup_Module
	8	Borland_Delphi_40
	9	VC8_Microsoft_Corporation
	10	Borland_Delphi_DLL

detect the packer or anti-debugging techniques as malware. One possible reason for this is the API and argument names used for obfuscation. In addition, the typical instructions can be extracted as printable strings. They must be remained for deobfuscation. Thus, our method is effective against packed malware and anti-debugging techniques.

Table 11 Top 10 anti-debugging names in the HA dataset

Dataset	No.	Anti-debugging name
	1	DebuggerException__SetConsoleCtrl
	2	VC8_Microsoft_Corporation
	3	DebuggerCheck__QueryInfo
	4	Microsoft_Visual_Cpp_8
	5	Microsoft_Visual_Cpp_v50v60_MFC
HA 2016	6	ThreadControl__Context
	7	Borland_Delphi_30_additional
	8	Borland_Delphi_30_
	9	Borland_Delphi_v40_v50
	10	Borland_Delphi_v30
	1	DebuggerException__SetConsoleCtrl
	2	VC8_Microsoft_Corporation
	3	Microsoft_Visual_Cpp_8_MFC
	4	ThreadControl__Context
	5	DebuggerCheck__QueryInfo
HA 2017	6	DebuggerCheck__RemoteAPI
	7	SEH__vectored
	8	Microsoft_Visual_Cpp_v50v60_MFC
	9	Microsoft_Visual_Cpp_80_DLL
	10	Microsoft_Visual_Studio_NET
	1	DebuggerException__SetConsoleCtrl
	2	VC8_Microsoft_Corporation
	3	Microsoft_Visual_Cpp_8
	4	ThreadControl__Context
	5	DebuggerCheck__QueryInfo
HA 2018	6	Microsoft_Visual_Cpp_v50v60_MFC
	7	DebuggerCheck__RemoteAPI
	8	Borland_Delphi_30_additional
	9	Borland_Delphi_30_
	10	Borland_Delphi_v40_v50

6.3 Limitation

We are aware that our study may have some limitations.

The first is attributed to our dataset. As described previously, this study used more than 500,000 samples. Actual samples, however, might be more distributed. Hence, our dataset might not represent the population appropriately. As the matter of fact, we cannot use all actual samples for evaluation. To the best of our knowledge, the possible solution is using large-scale and multiple sources.

The second is lack of detailed analysis. In this study, we used a simple signature-based packer detector. This program has approximately 30 percent of false negatives [38].

We did not identify the packer names of our samples completely. Hence, our experimental result may not be applicable to some sophisticated packers, which are not detected by signature-based packer detectors. We identified our samples with VirusTotal and Microsoft defender. As reported in a paper, there is a problem with label changes in VirusTotal [56]. This can affect the accuracy of our experiment. Further analysis is required to reveal these issues.

The third is lack of comparison. In this paper, we focused on the practical performance and did not compare our method with other related studies. Due to the issues about the dataset or implementation, it was not feasible to provide fair comparison. Further experiments are required to provide fair comparison.

7 Conclusion

In this paper, we apply NLP techniques to malware detection. This paper reveals that printable strings with NLP techniques are effective for detecting malware in a practical environment. Our dataset consists of more than 500,000 samples obtained from multiple sources. The training samples were selected from the earlier samples. The test samples contain many benign samples thereby be imbalanced. In the further experiment, the training samples account for only 10 percent thereby be highly imbalanced. Thus, our dataset represents more practical environment. Our experimental result shows that our detection model is effective to not only subspecies of the existing malware, but also new malware. Furthermore, our detection model is effective against packed malware and anti-debugging techniques.

Our study clearly has some limitations. Despite this, we believe our study could be a starting point to evaluate practical performance. Our method might be applicable to other architectures. In future work, we analyze the detail of the samples. A more precise packer detector will improve the reliability of this study.

Declaration

Conflict of interest The authors declare that they have no conflict of interest.

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

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.: Detection of new malicious code using n-grams signatures. In: PST, pp. 193–196 (2004). http://dev.hil.unb.ca/Texts/PST/pdf/assaleh.pdf
2. Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., Kruegel, C. When malware is packin' heat; limits of machine learning classifiers based on static analysis features Network and Distributed Systems Security (NDSS) Symposium (2020). https://doi.org/10.14722/ndss.2020.24310
3. Bilar, D.: Opcodes as predictor for malware. IJESDF. Int. J. Electron. Secur. Digit. Forensics **1**(2), 156–168 (2007)
4. Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., Glezer, C.: 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, *Lecture Notes in Computer Science*, vol. 4667, pp. 44–50. Springer (2007). https://doi.org/10.1007/978-3-540-74565-5_5
5. Hatada, M., Akiyama, M., Matsuki, T., Kasama, T.: Empowering anti-malware research in japan by sharing the MWS datasets. JIP **23**(5), 579–588 (2015). https://doi.org/10.2197/ipsjjip.23.579
6. Henchiri, O., Japkowicz, N.: A feature selection and evaluation scheme for computer virus detection. In: ICDM, pp. 891–895. IEEE Computer Society (2006). http://www.computer.org/csdl/proceedings/icdm/2006/2701/00/index.html
7. Ismail, I., Marsono, M.N., Nor, S.M.: Detecting worms using data mining techniques: Learning in the presence of class noise. In: K. YÃtongnon, 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, pp. 187–194. IEEE Computer Society (2010). http://www.computer.org/csdl/proceedings/sitis/2010/4319/00/index.html
8. Ito, R., Mimura, M.: Detecting unknown malware from ascii strings with natural language processing techniques. In: 2019 14th Asia Joint Conference on Information Security (AsiaJCIS), pp. 1–8 (2019). https://doi.org/10.1109/AsiaJCIS.2019.00-12
9. Jacob, G., Comparetti, P.M., Neugschwandtner, M., Kruegel, C., Vigna, G.: A static, packer-agnostic filter to detect similar malware samples. In: Flegel, U., Markatos, E.P., Robertson, W.K. (eds.) DIMVA. Lecture Notes in Computer Science, pp. 102–122. Springer, Berlin (2012)
10. Karim, M.E., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. J. Comput. Virol. **1**(1–2), 13–23 (2005)
11. Kolosnjaji, B., Eraisha, G., Webster, G.D., Zarras, A., Eckert, C.: 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, pp. 3838–3845 (2017). https://doi.org/10.1109/IJCNN.2017.7966340
12. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: W.K. 0001, R. Kohavi, J. Gehrke, W. DuMouchel (eds.) Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Min-

ing, Seattle, Washington, USA, August 22-25, 2004, pp. 470–478. ACM (2004)
13. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res **7**, 2721–2744 (2006)
14. Kong, D., Yan, G.: 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, pp. 1357–1365. ACM (2013). http://dl.acm.org/citation.cfm?id=2487575
15. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, pp. 1188–1196 (2014). http://jmlr.org/proceedings/papers/v32/le14.html
16. Lee, J., Im, C., Jeong, H.: 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, p. 75 (2011). https://doi.org/10.1145/1968613.1968704
17. Li, B., Roundy, K.A., Gates, C.S., Vorobeychik, Y.: 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, pp. 227–238. ACM (2017). https://doi.org/10.1145/3029806.3029815
18. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware. In: ACSAC, pp. 431–441. IEEE Computer Society (2007). http://www.computer.org/csdl/proceedings/acsac/2007/3060/00/index.html
19. Mastjik, F., Varol, C., Varol, A.: Comparison of pattern matching techniques on identification of same family malware. Int. J. Inf. Secur. Sci. **4**(3), 104–111 (2015)
20. Masud, M., Khan, L., Thuraisingham, B.: A scalable multi-level feature extraction technique to detect malicious executables. Inf. Syst. Front. - ISF **10**, 33–45 (2008). https://doi.org/10.1007/s10796-007-9054-3
21. Mikolov, T., Yih, W., Zweig, G.: Linguistic regularities in continuous space word representations. In: Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA, pp. 746–751 (2013). http://aclweb.org/anthology/N/N13/N13-1090.pdf
22. Mimura, M.: An attempt to read network traffic with doc2vec. J. Inf. Proces. **27**, 711–719 (2019). https://doi.org/10.2197/ipsjjip.27.711
23. Mimura, M.: Adjusting lexical features of actual proxy logs for intrusion detection. J. Inf. Secur. Appl. **50**, 102408 (2020)
24. Mimura, M.: An improved method of detecting macro malware on an imbalanced dataset. IEEE Access **8**, 204709–204717 (2020). https://doi.org/10.1109/ACCESS.2020.3037330
25. Mimura, M.: Using fake text vectors to improve the sensitivity of minority class for macro malware detection. J. Inf. Secur. Appl. **54**, 102600 (2020)
26. Mimura, M., Miura, H.: Detecting unseen malicious VBA macros with NLP techniques. JIP **27**, 555–563 (2019). https://doi.org/10.2197/ipsjjip.27.555
27. Mimura, M., Ohminami, T.: 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, *Lecture Notes in Computer Science*, vol. 11689, pp. 168–185. Springer (2019)
28. Mimura, M., Otsubo, Y., Tanaka, H.: Evaluation of a brute forcing tool that extracts the rat from a malicious document file. In: AsiaJCIS, pp. 147–154. IEEE Computer Society (2016). http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7781470
29. Mimura, M., Suga, Y.: Filtering malicious javascript code with doc2vec on an imbalanced dataset. In: 2019 14th Asia Joint Conference on Information Security (AsiaJCIS), pp. 24–31 (2019). https://doi.org/10.1109/AsiaJCIS.2019.000-9
30. Mimura, M., Tanaka, H.: Heavy log reader: Learning the context of cyber attacks automatically with paragraph vector. In: Information Systems Security - 13th International Conference, ICISS 2017, Mumbai, India, December 16-20, 2017, Proceedings, pp. 146–163 (2017). https://doi.org/10.1007/978-3-319-72598-7_9
31. Mimura, M., Tanaka, H.: Reading network packets as a natural language for intrusion detection. In: Information Security and Cryptology - ICISC 2017 - 20th International Conference, Seoul, South Korea, November 29 - December 1, 2017, Revised Selected Papers, pp. 339–350 (2017). https://doi.org/10.1007/978-3-319-78556-1_19
32. Mimura, M., Tanaka, H.: Leaving all proxy server logs to paragraph vector. J. Inf. Process. **26**, 804–812 (2018). https://doi.org/10.2197/ipsjjip.26.804
33. Mimura, M., Tanaka, H.: A linguistic approach towards intrusion detection in actual proxy logs: 20th international conference, icics 2018, lille, france, october 29-31, 2018, proceedings. pp. 708–718 (2018). https://doi.org/10.1007/978-3-030-01950-1_42
34. Miura, H., Mimura, M., Tanaka, H.: Macros finder: Do you remember loveletter? In: Information Security Practice and Experience - 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings, pp. 3–18 (2018). https://doi.org/10.1007/978-3-319-99807-7_1
35. Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown malcode detection via text categorization and the imbalance problem. International Conference on Intelligence and Security Informatics. In: ISI, pp. 156–161. IEEE (2008). https://doi.org/10.1109/ISI.2008.4565046
36. Nagano, Y., Uda, R.: Static analysis with paragraph vector for malware detection. In: IMCOM, p. 80. ACM (2017). http://dl.acm.org/citation.cfm?id=3022306
37. Ndichu, S., Kim, S., Ozawa, S., Misu, T., Makishima, K.: A machine learning approach to detection of javascript-based attacks using ast features and paragraph vectors. Appl. Soft Comput. **84**, 105721 (2019)
38. Perdisci, R., Lanzi, A., Lee, W.: Classification of packed executables for accurate computer virus detection. Pattern Recognit. Lett. **29**(14), 1941–1946 (2008). https://doi.org/10.1016/j.patrec.2008.06.016
39. Phung, N.M., Mimura, M.: Detection of malicious javascript on an imbalanced dataset. Internet of Things **13**, 100357 (2021). https://doi.org/10.1016/j.iot.2021.100357
40. Raff, E., Sylvester, J., Nicholas, C.: 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, pp. 121–132. ACM (2017). https://doi.org/10.1145/3128572.3140442
41. Raff, E., Sylvester, J., Nicholas, C.K.: Learning the pe header, malware detection with minimal domain knowledge. CoRR **abs/1709.01471** (2017). https://doi.org/10.1145/3128572.3140442
42. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop

on New Challenges for NLP Frameworks, pp. 45–50. ELRA, Valletta, Malta (2010). http://is.muni.cz/publication/884893/en
43. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Commun. ACM **18**(11), 613–620 (1975). https://doi.org/10.1145/361219.361220
44. Sathyanarayan, V.S., Kohli, P., Bruhadeshwar, B.: Signature generation and detection of malware families. In: Y.M. 0001, W. Susilo, J. Seberry (eds.) Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008, Proceedings, *Lecture Notes in Computer Science*, vol. 5107, pp. 336–349. Springer (2008)
45. Saxe, J., Berlin, K.: 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, pp. 11–20. IEEE Computer Society (2015). https://doi.org/10.1109/MALWARE.2015.7413680
46. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: 2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001, pp. 38–49. IEEE Computer Society (2001). https://doi.org/10.1109/SECPRI.2001.924286
47. Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C.: Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. Inf. Sec. Techn. Rep. **14**(1), 16–29 (2009)
48. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: 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, *Lecture Notes in Computer Science*, vol. 5758, pp. 121–141. Springer (2009). https://doi.org/10.1007/978-3-642-04342-0_7
49. Symantec: Internet Security Threat Report **24** (2019)
50. Tian, R., Batten, L.M., Versteeg, S.: Function length as a tool for malware classification. In: MALWARE, pp. 69–76. IEEE Computer Society (2008). http://doi.ieeecomputersociety.org/10.1109/MALWARE.2008.4690860
51. Tran, T.K., Sato, H.: Nlp-based approaches for malware classification from api sequences. In: 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), pp. 101–105 (2017). https://doi.org/10.1109/IESYS.2017.8233569
52. Wang, J., Ma, S., Zhang, Y., Li, J., Ma, Z., Mai, L., Chen, T., Gu, D.: NLP-EYE: detecting memory corruptions via semantic-aware memory operation function identification. In: 22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019, Chaoyang District, Beijing, China, September 23-25, 2019., pp. 309–321 (2019). https://www.usenix.org/conference/raid2019/presentation/wang-0
53. Webster, G.D., Kolosnjaji, B., von Pentz, C., Kirsch, J., Hanif, Z.D., Zarras, A., Eckert, C.: 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, *Lecture Notes in Computer Science*, vol. 10327, pp. 119–138. Springer (2017). https://doi.org/10.1007/978-3-319-60876-1_6
54. Ye, Y., Chen, L., Wang, D., Li, T., Jiang, Q., Zhao, M.: SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging. J Comput Virol **5**(4), 283–293 (2009). https://doi.org/10.1007/s11416-008-0108-y
55. Zhang, B., Yin, J., Hao, J., Zhang, D., Wang, S.: Malicious codes detection based on ensemble learning. In: B.X. 0001, L.T. Yang, J. Ma, C. MÃ1/4ller-Schloer, Y.H. 0001 (eds.) Autonomic and Trusted Computing, 4th International Conference, ATC 2007, Hong Kong, China, July 11-13, 2007, Proceedings, *Lecture Notes in Computer Science*, vol. 4610, pp. 468–477. Springer (2007)
56. Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G.: Measuring and modeling the label dynamics of online anti-malware engines. In: S. Capkun, F. Roesner (eds.) 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020, pp. 2361–2378. USENIX Association (2020). https://www.usenix.org/conference/usenixsecurity20/presentation/zhu
57. Zolotukhin, M., Hamalainen, T.: Detection of zero-day malware based on the analysis of opcode sequences. In: 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), pp. 386–391 (2014). https://doi.org/10.1109/CCNC.2014.6866599

Mamoru Mimura received his B.E. and M.E. in Engineering from National Defense Academy of Japan, in 2001 and 2008, respectively. He received his Ph.D. in Informatics from the Institute of Information Security in 2011 and M.B.A. from Hosei University in 2014. During 2001–2017, he was a member of the Japan Maritime Self-Defense Force. During 2011–2013, he was with the National Information Security Center. Since 2014, he has been a researcher in the Institute of Information Security. Since 2015, he has been with the National Center of Incident Readiness and Strategy for Cybersecurity. Currently, he is an Associate Professor in the Dept. of Computer Science, National Defense Academy of Japan.

Ryo Ito recived his B.E and M.E from National Defense Academy of Japan in 2013 and 2020 respectively. Currently, he is a member of the Japan Ground Self Defense Force.