# Detecting Unknown Malware from ASCII Strings with Natural Language Processing Techniques

Ryo Ito
*Department of Computer Science*
*National Defense Academy*
Yokosuka, Japan
em57031@nda.ac.jp

Mamoru Mimura
*Department of Computer Science*
*National Defense Academy*
Yokosuka, Japan
mim@nda.ac.jp

***Abstract*—Attackers often use an executable file (malware) as a tool to obtain sensitive information from specific companies and individuals. Anti-virus software attempts to detect the malware by pattern matching method etc. However, it is difficult to detect unknown malware in these methods. The unknown malware is detected by a sandbox, etc. We consider another method because the sandbox requires much time for running. ASCII strings extracted from executable files are helpful for analyzing malware. With the recent development of natural language processing (NLP) techniques, it is becoming possible to use these strings as a malware detection method. In this paper, we propose a malware detection method using ASCII strings with NLP techniques. Our method divides these strings into words, and distinguishes the difference of the words between benign and malicious executable files. To compare with the arrangement of words or the frequency of appearing words, uncommon words are unnecessary in NLP techniques. Thus, we consider that reducing the uncommon words improves the detection rate. Our method converts a corpus of frequent words into a feature vector with natural language processing techniques. In our experiments, we used a dataset containing more than 23,000 malware samples (more than 2,100 malware families) provided by FFRI and more than 16,000 benign files collected from "download.cnet.com". Our method achieves the F-measure more than 0.85. The experimental results show that our method detects unknown malware with high accuracy.**

***Index Terms*—executable file, malware, detection, machine learning, SVM, natural language processing, Doc2Vec, LSI**

## I. Introduction

Targeted attacks violate security measures of specific individuals or organizations. The targeted attacks require special attention because this is designed to capture the sensitive information of a particular individual or an organization. The targeted attacks may include threats delivered executable files (malware) via SMTP e-mail [1].

An attacker uses the targeted attack and attempts to persuade a victim to run an executable file. When the victim runs the executable file sent from the attacker, the computer is infected with malware. Thus, the attacker can compromise a system's security and steal the information held by the target.

An attacker preliminarily confirms that malware is not detected by anti-virus software updated with the latest pattern file. Therefore, it is difficult for antivirus software detecting unknown malware used in targeted attacks. To enable anti-virus software to detect the unknown malware, it is necessary to create a pattern file from the malware. However, it takes time to create a pattern file corresponding to the unknown malware. A sandbox is detecting unknown malware as a means without pattern files. The sandbox runs an executable file and detects the malware by checking its behavior. Because it requires time to analyze for detection, this method is not suitable for processing large amounts of files. Therefore, a fast and easy method to detect unknown malware is required.

In recent years, with the development of natural language processing (NLP) techniques, it is possible to output expression intended by people in machine translation, search engines and so on. In machine translation, every word in a sentence is translated one by one. The translated word is converted into an optimum meaning from its relation to the adjacent word. Therefore, the translated sentence becomes natural by machine translation using NLP techniques. As a result, it is becoming possible to use native code as a malware detection method.

For exmple, Nagano et al. [2] propose a detection method using NLP techniques. Their method is evaluated by 10-fold cross-validation with 3,600 executable files. Some NLP techniques use neural networks, thereby it is difficult to reveal how to convert into vectors. They convert features into vectors using Doc2Vec, one of the NLP techniques. The features acquired by static analysis are vectors: DLL Import, assembly code and hex dump. However, processing time is required to obtain an assembly code with disassembly. And, the information of 32-byte hex dumps includes no semantics.

To practice effective utilization of the information, we propose a new detection method using ASCII strings with NLP techniques.ASCII strings are known as the printable characters, represent letters, digits, punctuation marks, and a few miscellaneous symbols. There is no special technique for extracting ASCII characters. To the best of our knowledge, the detection method using ASCII strings with NLP techniques is the first method. The corpus of a NLP model is a collection of ASCII strings extracted from each executable file. To comparing with the arrangement of words or the frequency of appearing words, uncommon words are unnecessary in NLP techniques. Several words in the corpus are less frequent, and contribute less to classification. Therefore, the corpus should be created by selecting a high-frequency word. By contrive ways to create a corpus, we consider that our malware

detection rate will be improved.
In our method, frequent words are selected. We convert the words into feature vectors with a language model which is constructed using NLP techniques such as Doc2Vec or LSI. We construct a classifier with a Support Vector Machine (SVM) using the labeled feature vectors generated from the training data. Our method is evaluated by time series analysis with two times more data than their method [2]. Since conducting several experiments, we evaluate the detection performance.
In summary, the contributions are as follows:

- We propose new malware detection method using ASCII strings with NLP techniques.
- Our method is evaluated by time series analysis using a dataset. We used the dataset containing more than 23,000 malware samples (more than 2,100 malware families) provided by FFRI, and more than 16,000 benign files collected from ”download.cnet.com”.
- By focusing on the topic vectors of LSI, we reveal ASCII strings that contribute to malware detection.

The structure of this paper is as follows. Section 2 explains the natural language processing techniques used in this research. Section 3 details the proposed method. Section 4 describes the verification experiment and its result. Section 5 discusses the performance and research ethics. Section 6 explains related research and clarifies the difference between the proposed method and the previous research. In the last section 7, we summarize this research and describe future tasks.

## II. Natural Language Processing Techniques

In this section, we explain the natural language processing (NLP) techniques used in this research. NLP techniques are used for processing natural language by a computer. The sentences are separated into words and the difference in appearance frequency of each word, etc. is converted into a vector with these techniques such as Bag-of-Words, Latent Semantic Indexing, etc.

### A. Bag-of-Words

Bag-of-Words (BoW) model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier [3]. The most common type of features calculated from the BoW model is term frequency.
Let $d$ is expressed as a document, $w$ as a word ($w_{i=1,2,3,...}$), and $n$ as a frequency of $w$, the document $d$ can be expressed by equation (1). For this equation (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 the equation (3), let construct the other documents to record the term frequencies of all the distinct words (other documents ordered as in the equation (2))
Since transforming the document into a ”Bag-of-Words”, we can calculate various measures to characterize the document.

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

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

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

Using BoW, each document is converted into a vector with a unique number of words (dimensionality). However the number of dimensions becomes too large and the calculation time is greatly increased.

### B. Latent Semantic Indexing

Latent Semantic Indexing (LSI) model is one of NLP techniques that analyze the relevance between a document group and words included in a document. LSI calculates singular value decomposition from a vector transformed by BoW. 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, we use Term Frequency - Inverse Document Frequency (TF-IDF). $|D|$ 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 expressed by equation (4).

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

Next, we calculate singular value decomposition for the vector weighted by TF-IDF. 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 $\Sigma$, from the theory of linear algebra. This is called a singular value decomposition. At this time, $U$ is a column orthogonal matrix and is 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 $U$ shows a latent meaning.

$$X = \begin{bmatrix} x_{1,1} \cdots x_{1,j} \\ \vdots \ddots \vdots \\ x_{i,1} \cdots x_{i,j} \end{bmatrix} = U \Sigma V^T$$
$$= \begin{bmatrix} u_{1,1} \cdots u_{1,r} \\ \vdots \ddots \vdots \\ u_{i,1} \cdots u_{i,r} \end{bmatrix} * \begin{bmatrix} \sigma_{1,1} \cdots 0 \\ \vdots \ddots \vdots \\ 0 \cdots \sigma_{r,r} \end{bmatrix} * \begin{bmatrix} v_{1,1} \cdots v_{1,j} \\ \vdots \ddots \vdots \\ v_{r,1} \cdots v_{r,j} \end{bmatrix}$$

### C. Paragraph Vector

Simplistic methods of BoW and LSI lost many subtleties of a possible good representation, e.g consideration of word ordering. Word2Vec is used to generate representation vectors out of words.Models of word2Vec are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Paragraph Vector is an extension of the Word2Vec [4] model which generates the word feature vector proposed by Mikolov et al [5]. Word2Vec can utilize either of two model architectures to produce a distributed representation of words: Continuous Bag-of-Words (CBoW) or continuous skip-gram. $queen = king - man + woman$ is an example of operation using each word vector generated by Word2Vec (CBoW model). Paragraph Vector, which extends Word2Vec can perform this type of computation. The Paragraph Vector is implemented two models that are Paragraph Vector with Distributed Memory (PV-DM) and Distributed Bag of Words (PV-DBoW). According to the Mikolov et al [5], if the words of the corpus used for the model are many, the number of dimensions of the word vector will also increase. However after reaching some point, marginal gain will diminish. In order to perform highly accurate classification, it is necessary to adjust the parameters (dimensionality and context window etc.).

## III. Proposed Method

In this section, we describe our malware detection method. An outline of the proposed method is shown in Fig1. The proposed method can be divided into a training phase and a test phase to actually classify. The training phase performs known benign and malicious executables. The test phase performs each unknown executable files. Below, we explain the outline of the proposed method in the order of Fig1.

1. Extract strings from the known executable files, and separated into words.
2. Select the words with high frequency and create a corpus.
3. Construct a language model using NLP techniques and convert from the words to a feature vector with the model.
4. Train an SVM classifier using the feature vectors generated from the model.
5. Extract strings from the unknown executable files.
6. Reduce the words, by extracting the words selected in 2. from the strings.
7. Convert the strings into feature vectors using the language model Constructed in 3.
8. Classify the feature vectors using the classifier trained with 4.

We describe details of each procedure, below.

### A. Extracting strings

In step 1: ”Extracting strings” of Fig.1, our method extracts ASCII strings from an executable file. ASCII strings consist of alphabets and symbols (e.g., ”$”, ”%”). Extracted strings are continuous sentences, however, they are separated into words by spaces. Separated words include many words which are not in an English dictionary.

### B. Creating a corpus

In step 2: ”Creating a corpus”, our method creates a corpus from the strings extracted from the executable file. Our method

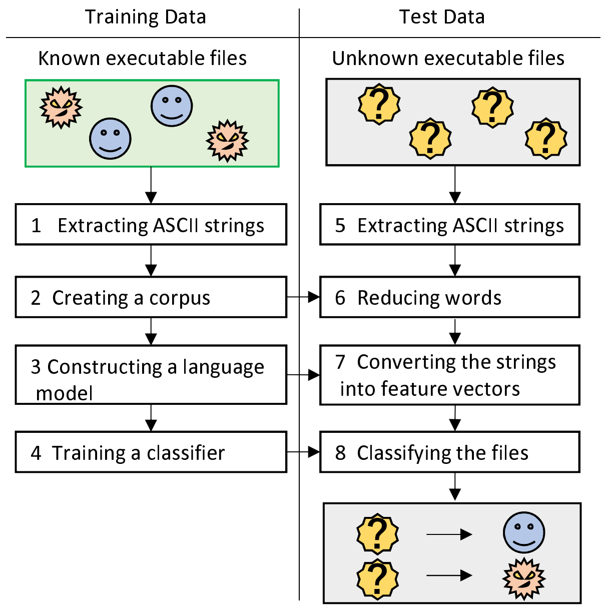

Fig. 1. Outline of the Proposed Method

is to select the strings with high occurrence frequency. We extract unique words from benign and malicious executable files. The corpus is collected the same number of unique words from each executable files. A number of unique words included in the corpus use the same number from each other. In addition, the unique word number is the number of a non-overlapping word, regardless of frequency. The optimum number of unique words used in the corpus is determined by the preliminary experiment. The method of the preliminary experiment will be explained in the verification experiment section.

### C. Constructing a language model

In step 3: ”Constructing a language model”, our method constructs a language model using NLP techniques.NLP technology used in our method is LSI and Doc2Vec. In LSI, a language model to convert the words to feature vectors is made by using singular value decomposition. In Doc2Vec, a language model to convert the words to feature vectors is made by using PV-DM model of Paragraph Vector.

### D. Training a classifier

In step 4: ”Training a classifier”, our method trains Support-Vector Machines [6] (SVM) classifiers with the training data. An SVM is one of the supervised learning models for classification. For example, given a set of training examples, each marked as belonging to one or the other of two categories. The SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. The effectiveness of the SVM depends on the selection of the parameters

(kernel, the kernel's parameters, and soft margin parameter C). In order to create an excellent classifier, adjustment of those parameters is necessary. Optimal parameters will be determined by searching in another experiment.
Training data used for classifier training is created from known executable files. The training data is labeled the feature vectors benign or malicious with the correct answers. The feature vectors are converted by Doc2Vec or LSI.

### *E. Detecting unknown malware*

In steps 5-8, We show the procedure to classify unknown executable files in test data phase of Fig.1. First, ASCII strings are extracted from the unknown executable files using the Strings command. We limit the strings included in the corpus created and reduce extracted strings. Next, the extracted strings are converted into the feature vector with the constructed language model. Finally, malware is detected by classifying the feature vector as benign or malicious with the classifier trained by training data.

### *F. Implementation*

In the implementation of our method, we use python 2.7 of programming language. NLP techniques and machine learning are implemented in the python. Therefore, it is easy to implement our method with the python module. For the machine learning library, we use scikit-learn [7] including SVM, random forest etc. The SVM used in this research is one of supervised machine learning models for pattern identification.
We set the control parameter $C$ , the default kernel function $\gamma$ etc as initial values. For a library of NLP techniques, we use gensim [8] including Bag-of-Words, LSI, Doc2Vec and others. We set $epochs = 30$ in parameters of Doc2Vec, and 100 in $Num\_topics$ of LSI. First, we set a default in other parameters. Thereafter, we adjust various parameters to suitable values.

## IV. Verification Experiment

We prepared data that extracted strings from malicious executable files by age from 2013 to 2017. In addition, we prepared data that extracted strings from benign executable files. In order to simulate unknown malware, we use malware separately by year. The details are described below.

### *A. Dataset*

The dataset used in this experiment is classified into benign data and malicious data. Table I shows the number of the data and unique words. Cnet is benign data. FFRI 2013-2017 are malicious data. The details about benign data and malicious data are described below.

*1) Benign Data:* Cnet is benign data which are ASCII strings extracted from executable files (.exe) collected from a web site of "Windows PC Software - Free Downloads and Reviews" [9]. Cnet is extracted from the executable file using the Strings command [10]. These executable files are regarded as benign executable files. The executable files were downloaded by crawling between 5,000 web pages in order from the top page of the web. The information of the collected files examined with PEiD [11] is shown in Table II. The examined information are compiler and source code. It is understood from Table II that there are many types in the compiler and source code. Therefore, the data set is considered to be unbiased and generalized.
Cnet files are sorted in alphabetical order by file name and divided into odd numbered files (called Cnet A) and even numbered files (called Cnet B). In this experiment, we use Cnet A and Cnet B.

TABLE I
Number of files and unique words in each dataset

| Data name | Number of files | Number of unique words |
|---|---|---|
| Cnet | 16,560 | 176,185,131 |
| FFRI 2013 | 2,637 | 1,317,912 |
| FFRI 2014 | 3,000 | 5,193,423 |
| FFRI 2015 | 3,000 | 3,224,583 |
| FFRI 2016 | 8,243 | 11,443,274 |
| FFRI 2017 | 6,251 | 1,534,580 |

TABLE II
Information on the executable files (cnet)

| Infomation of PEID | | Cnet |
|---|---|---|
| compiler | Visual Studio | 1,373 |
| | Borland Delphi | 6,230 |
| | Symantec Visual Cafe | 621 |
| | Installer | 2,985 |
| | unknown | 5,344 |
| source code | Cpp | 11,422 |
| | C#/.Net | 178 |
| | C | 17 |
| | Basic | 78 |
| | java | 621 |
| | unkonwn | 4,243 |
| Number of files | | 16,560 |

*2) Malicious Data:* Malicious data is the FFRI dataset provided by the FFRI corporation [12]. The FFRI dataset is provided as a part of MWS Datasets 2018 [13], and dynamic analysis results of the different sample is saved in JSON format. The FFRI corporation use a cuckoo sandbox [14] for dynamic analysis tool. In this research, we use the FFRI dataset from 2013 to 2017 and use only the strings items included in the dataset. As a result of investigating the malware family, the dataset includes the 2102 families.

### *B. Performance Metrics*

The confusion matrix is one of the most intuitive and easiest metrics used for finding the accuracy of the model. The predicted values are malicious (Positive) and benign (Negative). The actual value is decided with a correct value (True) or incorrect value (False). For example, if predict malicious data is benign data by mistake, the value is False Negative (FN). The four outputs formulated as $2 \times 2$ confusion matrix is shown in Table III. In this experiment, we use Accuracy, Precision, Recall and F-measure as Performance metrics. The definition of each Performance Indicator is shown in equations (4) to (7).

TABLE III
CONFUSION MATRIX BETWEEN PREDICTED VALUE AND ACTUAL VALUE

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

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

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

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

$$F - measure = \frac{2Recall * Precision}{Recall + Precision} \tag{8}$$

## C. Preliminary Experiment

In the preliminary experiment, the optimum number of the unique words used in a corpus is calculated. The corpus is collected the same number of unique words from benign and malicious executable files. As the training data, we use Cnet A for benign data and FFRI 2013,2014 for malicious data. Of the test data, we use Cnet B for benign data and FFRI 2015 for malicious data. The total number of corpus increases by 1,000 to 1,000, and it is calculated up to 10,000. The result is shown in Fig. 2. In Fig. 2, the vertical axis represents the evaluation metrics and the horizontal axis represents the total number of unique words used in the corpus. From Fig. 2, as the total number of corpus is 3,000, F-measure is the highest value. Therefore, 1,500 unique words are used in descending order of appearance frequency from benign and malicious data, respectively. In a later experiment, we use 3,000 corpus.

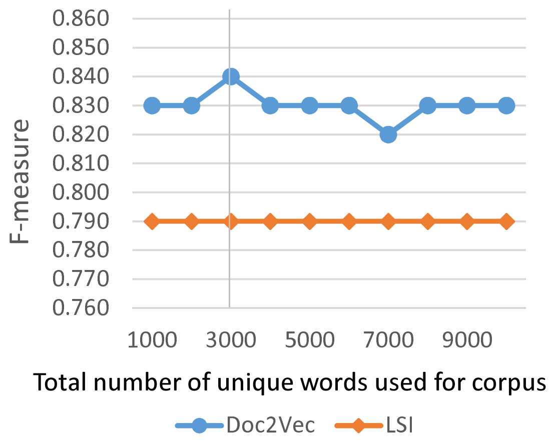

Fig. 2. Result of preliminary experiment

## D. Search for optimal parameters

In this subsection, we adjust various parameters. First, we adjust the dimension number of the language model with Doc2Vec and LSI. Next, we adjust the parameters of SVM. Grid search [7] was used for parameter adjustment of SVM. Finally, we adjust the detailed parameters of Doc2Vec. The adjustment range of parameters and adjustment result are shown in Table IV. Optimal parameter values were different for LSI and Doc2Vec. In a later experiment, we use each optimum value obtained in this experiment.

TABLE IV
PARAMETERS TO BE SEARCHED AND OPTIMUM VALUES

| Parameter | Search range | Doc2Vec | LSI |
|---|---|---|---|
| dimensions | 100, 200, ... , 1,000 | 400 | 100 |
| C | 1, 10, 100, 1000 | 10 | 1 |
| kernel | linear, rbf, poly, sigmoid | rbf | rbf |
| gamma(rbf) | 0.1, 0.01, 0.001 | 0.01 | 0.01 |
| degree(poly) | 2, 3, 4 | - | - |
| alpha | 0.025, 0.05, 0.075 | 0.075 | - |
| min_count | 1, 2, 3, 4 | 2 | - |
| window | 1, 2, 3, 4 | 1 | - |
| epochs | 1, 10, 20 | 20 | - |

## E. Cross-Validation

The goal of cross-validation is to test the model ability to predict new data that was not used in estimating it. To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model predictive performance. In this research, we perform 10-fold cross-validation using a language model with Doc2Vec and LSI. The dataset of cross-validation is cnet A (benign data), and FFRI 2013 and 2014 (malicious data). In this experiment, we experiment with two methods (A and B). Method A is a classification method using all words. Next, method B is a classification method by reducing words that we proposed so far. We evaluate our method by experimenting with these methods.
The results are shown in Table V. Since the evaluation metrics is high, we considere that there is generalization performance. As the results of this experiment, Method B is better than Method A. Thereafter, only method B is evaluated with an experiment.

TABLE V
EXPERIMENT RESULT OF CROSS-VALIDATION

| | | Method A | Method B |
|---|---|---|---|
| Doc2Vec | Accuracy | $0.95 \pm 0.01$ | $0.96 \pm 0.01$ |
| | Precision | $0.94 \pm 0.01$ | $0.96 \pm 0.01$ |
| | Recall | $0.94 \pm 0.01$ | $0.96 \pm 0.01$ |
| | F-measure | $0.94 \pm 0.01$ | $0.96 \pm 0.01$ |
| LSI | Accuracy | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ |
| | Precision | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ |
| | Recall | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ |
| | F-measure | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ |

## F. Verification Experiment

In this subsection, we experiment to detect a simulated unknown malware on time series. Table VI shows the combination of the training data and test data in the verification experiment. The training data contains Cnet A as benign data and, the FFRI 2013 and 2014 as malicious data. The test data contain Cnet B as benign data, and the FFRI 2015 through the FFRI 2017 as malicious data. The environment of the verification experiment is shown in Table VII.

TABLE VI
COMBINATION OF TRAINING DATA AND TEST DATA USED IN VERIFICATION EXPERIMENT

| Training data | | Test data | |
|---|---|---|---|
| Benign data | Malicious data | Benign data | Malicious data |
| Cnet A | FFRI 2013<br>FFRI 2014 | Cnet B | FFRI 2015 |
| | | | FFRI 2016 |
| | | | FFRI 2017 |

TABLE VII
EXPERIMENT ENVIRONMENT

| | |
|---|---|
| CPU [Memory] | Core i7-5820K 3.30GHz [32.0GB] |
| OS | Windows 10 Pro |
| Programming Language | python v2.7.14 |
| Using modules | scikit-learn v0.19.1, gensim v3.4.0 |

### G. *Experimental Result*

The result of the verification experiment is shown in Table VIII. ”Converting time” of Table is the number of seconds to convert the string of test data into a feature vector. ”Classifying time” of Table is the number of seconds to classify the vector of test data. As a result of the verification experiment, the performance of Doc2Vec is the best, the processing time of LSI is the shortest.

## V. DISCUSSION

### A. *Verification Performance*

Detection accuracy of Doc2Vec with Method B (reducing words) is better than the others. The difference between the two methods is whether to reduce words. Therefore, to improve accuracy, we consider that reducing words is effective to improve performance. The corpus with reducing words seems to clarify the distinction between benign data and malicious data by excluding words with low occurrence frequency.
From cross-validation, the F-measure of method B are more than 0.96. Comparing the cross-validation results with our method and research of Nagano et al. [2], our results are higher. Comparing the number of dataset and the balance of dataset, our experiment is worse condition.
From the time series analysis, the performance of F-measure did not decrease greatly during 2017. It is considered that our method with Doc2Vec is effective for a long term.
The LSI model has less performance than Doc2Vec. As a cause of low performance, we consider that training amount using malware is inadequate. As an improvement method, we consider that it is better to increase malware training data.
We consider the words included in topic vectors of the LSI model. The LSI converts it into topic vectors based on words related to the document group. Table IX shows some of the contents of topic vectors. Topic 1 includes VB’s API often used by malware. Topic 2 includes a general API and UPX (packer) [15] . However, ordinary files rarely use ”advapi32.dll” and ”oleaut32.dll”. ”advapi32.dll” and credential management are related, ”oleaut32.dll” contains a vulnerability to take control. Therefore, topics 1 and 2 can be expected to be close to malignancy. Topic 3 contains words that shows easy-to-understand words. Topic 4 contains good image words such as ”Microsoft” and ”Copyright”. Therefore, topics 3 and 4 can be expected to be close to benign. It is understand that there are many similar words in the topic vector. As a result, we consider that it could be classified by LSI.

### B. *Processing Speed*

As a result of the verification experiment, Method B of LSI is faster than the others.
First, we study the speed of converting a word into a feature vector. Since LSI has no hidden layer unlike Doc2Vec, the conversion speed of LSI is fast. Since the total number of words is very small, it is easy to convert method B to feature vector rather than method A. As a result, we consider that the converting time got faster.
Next, we study the time to classify. The classification time is influenced by the dimension number of a feature vector. The dimension number of LSI is set to 100, and Doc2Vec is set to 300. As a result, we consider LSI to be faster.

### C. *Cause of False Negative*

In Method B, we analyze the false negative data. First, we consider that test data contains malware (new malware family) not found in the training data. Table X shows the malware families that occurred after 2015 in the FFRI data set. ”New malware families” of Table X is a malware family that did not appear in FFRI 2014. Table XI shows the number of new malware families among the false negatives. From Table XI, we estimate that our methods also detecting new malware families. However, the false negative data contain many new malware families.
Next, we consider that false negative data contain malware families that are difficult with our methods. Table XII shows the top 5 of the malware families that cannot be detected from 2015 to 2017. ”Qty” in Table XII is the number of data. Bagsu, Dynamer and Banker are similar malware. We consider that these malware had become popular around 2015, therefore the amount included in the training data was small. Zbot and Fareit are close to malware. These malware have existed for a long time. An attacker packs and obfuscates their malware in different ways [16]. The common points of the malware families in Table XII are practical to target for financial data. The LSI model detects a difference in the frequency of words. Therefore, the detection will be avoided by converting ASCII strings into different strings. The Doc2Vec model detects differences in word context. Therefore, the detection will be avoided by converting the order of the words. As a result, we consider that detection differences occurred in each method.

TABLE VIII
EXPERIMENT RESULT

| | FFRI 2015 | | FFRI 2016 | | FFRI 2017 | |
|---|---|---|---|---|---|---|
| | Doc2Vec | LSI | Doc2Vec | LSI | Doc2Vec | LSI |
| Accuracy | 0.92 | 0.89 | 0.89 | 0.83 | 0.91 | 0.83 |
| Precision | 0.89 | 0.82 | 0.95 | 0.92 | 0.94 | 0.89 |
| Recall | 0.82 | 0.77 | 0.82 | 0.72 | 0.83 | 0.69 |
| F-measure | 0.85 | 0.80 | 0.88 | 0.81 | 0.89 | 0.78 |
| Converting time | 97(s) | 23(s) | 118(s) | 30(s) | 98(s) | 25(s) |
| Classifying time | 8 (s) | 6(s) | 13(s) | 10(s) | 11(s) | 9(s) |

TABLE IX
THE CONTENTS OF TOPIC VECTORS

| Topic No. | Words of a Contents of Topic Vecter |
|---|---|
| 1 | MSVBVM60.DLL, __vbaFreeStr, __vbaStrMove, __vbaFreeVar, __vbaHresultCheckObj, __vbaFreeVarList, __vbaFreeStrList, __vbaFreeObj, __vbaNew2, __vbaStrCat" |
| 2 | KERNEL32.DLL, Failed, VirtualProtect, "%s", DigiCert, advapi32.dll, UPX, oleaut32.dll, user32.dll, library" |
| 3 | gE, Instructs, COMODO, hF, CA, Prevents, Overrides, Setup, Specifies, that" |
| 4 | "%s", Sans, Serif, InstallShield, fffffox, Microsoft, MS, "Copyright, SysUtils, [[[[Q&" |

TABLE X
MALWARE FAMILY THAT OCCURRED AFTER 2015 ( DATA NUMBER OF NEW FAMILIES / DATA NUMBER OF TOTAL )

| | FFRI 2015 | FFRI 2016 | FFRI 2017 |
|---|---|---|---|
| New Families | 1,545 / 3,000 | 4,017 / 8,243 | 3,975 / 6,251 |

### D. Research Ethics

First, we consider this research to be reproducible. We use general home computer, therefore special equipment is not required. For the data used in this research, we created the dataset that can be obtained by anyone. In addition, our method can be easily implemented using the open source programming language and machine learning libraries. Therefore, the reproducibility of this research is extremely high.
Secondly, we do not cause trouble to others with the collection of data used in this research. A large amount of training data is required for machine learning. Crawling is efficient to obtain a large amount of training data. However, continuous downloads overload a server. In order to avoid loading the server, we set a standby time to avoid continuous access and collected it. Therefore, in this research, it has little impact on servers that provide services and their users.
Finally, we consider this research to be the public benefits. Even if we publish our research results, we judged that it is difficult for an attacker to avoid this detection method. Even if an attacker creates new malware that avoids our method, an SVM can detect it with training that malware. The SVM will be able to classify new malware by machine learning algorithms.

### E. Limitations

In this research, we evaluated the malware detection method using ASCII strings. As a result, we revealed the effectiveness of the method. However, there are some limitations that would allow attackers to bypass its detection.
First, attackers use obfuscation to evade our approach. We could not check whether the data set used in this experiment was obfuscated. Therefore, we consider that the verification is insufficient for obfuscation. In the future, we will require to evaluate our method using new obfuscated dataset.
Second, strict parameter adjustments must be determined. In this experiment, the parameters are determined by a rough grid search. To want better results, we require to do a finer grid search. However, it is difficult because of the huge processing time required.
Third, the SVM classifier is changed to other classifiers such as XGBoost [17], etc.XGBoost is a different classification method from SVM, with Gradient Boosting method and Random Forest method. Therefore, changing to other classifiers may improve detection accuracy.
By addressing these issues in the future, we plan to further improve the detection method.

TABLE XI
AMONG FALSE NEGATIVES, THE NUMBER OF NEW MALWARE FAMILIES (NEW FAMILIES / FN)

| | FFRI 2015 | FFRI 2016 | FFRI 2017 |
|---|---|---|---|
| Doc2Vec | 288 / 543 | 301 / 1,302 | 434 / 1,003 |
| LSI | 337 / 668 | 1,102 / 2,253 | 1,263 / 1,890 |

TABLE XII
MALWARE FAMILY THAT WAS FALSE NEGATIVES BETWEEN 2015 AND 2017

| Doc2Vec | | LSI | |
|---|---|---|---|
| Family Name | Qty | Family Name | Qty |
| Trojan:Win32/Dynamer!ac | 31 | Trojan:Win32/Dynamer!ac | 44 |
| PWS:Win32/Zbot | 22 | PWS:Win32/Zbot | 32 |
| Trojan:Win32/Bagsu!rfn | 21 | Trojan:Win32/Bagsu!rfn | 26 |
| TrojanSpy:Win32/Banker | 10 | PWS:Win32/Fareit | 11 |
| PWS:Win32/Fareit | 8 | TrojanSpy:Win32/Banker | 9 |

## VI. RELATED WORK

Many detection methods for malware of executable file have been proposed. Many virus vendors detect new malware in various ways. In this section, we will describe related works similar to our research. The feature vectors of our method use native code contained in the executable files and are useful for detection. In the following, we introduce related works using native code and describe the differences with us.

Lee et al. [18], Mastjik et al. [19], and Ye et al. [20] propose a detection method which calculates the similarity between two files to be executed through by comparing strings to identify and classify malware. The strings of their method select ASCII strings that are valid for measuring the similarity. However, our research uses all ASCII strings and does not use the similarity. In our method, we convert the strings into vectors with NLP techniques and linearly separate the vectors with SVM. As a result, we can detect malware.

Nataraj et al. [21], [22] and Hiromu et al. [23] propose a simple yet effective method for visualizing and classifying malware using image processing techniques. Malware binaries are visualized as gray-scale images. Motivated by this visual similarity, a classification method using standard image features is proposed. Classification methods are k-nearest neighbor and CNN. They classify malware families by these methods. The purpose of their method is not detection method of malware, though our method is detection method.

Nagano et al. [2] and Saxe et al. [24] propose a new method which automatically detects new malware subspecies by static analysis of execution files and machine learning. Their method is evaluated by 10-fold cross-validation with 3,600 executable files. The feature acquired by static analysis are vectors : DLL Import, assembly code and hex dump. However, processing time is required to obtain an assembly code with disassembly. And, the information of 32-byte hex dumps includes no semantics. To use the information of semantics such as API, ASCII strings should be used by a detection method. Our method can be improved by using ASCII strings and no disassembly is required. Therefore, our method is faster and better than their method. Compared with their evaluation, our method is evaluated by time series analysis with 10 times more data than theirs.

## VII. Conclusion

In this research, we propose a method to detect unknown malware using feature vectors converted by natural language processing techniques from ASCII strings extracted from executable files. An SVM is used to classify benign execution file and malware. The corpus of language models is collected unique words with high occurrence frequency from each benign and malicious executable file. Our detection method is evaluated by time series analysis using FFRI data set. As a result of the verification experiment, the average F-measure of the proposed method using Doc2Vec is 0.87. Moreover, the proposed method is effective even after years. Therefore, we consider that the proposed method is effective. By focusing on the topic vectors of LSI, we reveal ASCII strings that contribute to malware detection.

In future work, we plan to use datasets consisting of obfuscated executable files. We evaluate the effects of our method on obfuscation.

## References

[1] Symantec Corporation. Internet security threat report istr email threats 2018. https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf. Accessed: 2019-2-20.

[2] Yuta Nagano and Ryuya Uda. Static analysis with paragraph vector for malware detection. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017, Beppu, Japan, January 5-7, 2017*, page 80, 2017.

[3] Zellig Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

[4] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[5] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 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*, pages 746–751, 2013.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[7] Funding provided by INRIA et al. scikit-learn. http://scikit-learn.org/stable/. Accessed: 2019-2-20.

[8] Copyright Radim Rehurek. gensim. https://radimrehurek.com/gensim/. Accessed: 2019-2-20.

[9] CBS Interactive Inc. Windows pc software - free downloads and reviews. https://download.cnet.com/windows/. Accessed: 2019-2-20.

[10] jbremer. cuckoosandbox/cuckoo. https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/processing/strings.py. Accessed: 2019-2-20.

[11] K-atc. Yet another implementation of peid with yara. https://github.com/K-atc/PEiD. Accessed: 2019-2-20.

[12] FFRI corporation. Home ffri, inc. Accessed: 2019-2-20.

[13] Yuta Takata, Masato Terada, Takahiro Matsuki, Takahiro Kasama, Shoko Araki, and Mitsuhiro Hatada. Datasets for anti-malware research - mws datasets 2018 -. *IPSJ SIG Technical Report*, 2018.

[14] Stichting Cuckoo Foundation. Cuckoo sandbox - automated malware analysis. https://cuckoosandbox.org/. Accessed: 2019-2-20.

[15] Markus F.X.J. Oberhumer et al. Upx the ultimate packer for executables. https://upx.github.io/. Accessed: 2019-2-20.

[16] James Wyke. What is zeus? a sophoslabs technical paper. https://www.sophos.com/en-us/medialibrary/pdfs/technical%20papers/sophos%20what%20is%20zeus%20tp.pdf. Accessed: 2019-2-20.

[17] Sphinx. Xgboost documentation - xgboost 0.83.dev0 documentation. https://xgboost.readthedocs.io/en/latest/. Accessed: 2019-2-20.

[18] Jinkyung Lee, Chaetae Im, and Hyuncheol 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*, page 75, 2011.

[19] Ferdiansyah Mastjik, Cihan Varol, and Asaf Varol. Comparison of pattern matching techniques on identification of same family malware. *International Journal of Information Security Science*, 4(3):104–111, 2015.

[20] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging. *Journal in Computer Virology*, 5(4):283–293, 2009.

[21] Lakshmanan Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *2011 International Symposium on Visualization for Cyber Security, VizSec '11, Pittsburgh, PA, USA, July 20, 2011*, page 4, 2011.

[22] Lakshmanan Nataraj, Vinod Yegneswaran, Phillip A. Porras, and Jian Zhang. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011*, pages 21–30, 2011.

[23] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, Tempe, AZ, USA, March 19-21, 2018*, pages 127–134, 2018.

[24] Joshua Saxe and Konstantin 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*, pages 11–20, 2015.