Hindawi
Security and Communication Networks
Volume 2019, Article ID 8485365, 9 pages
https://doi.org/10.1155/2019/8485365

WILEY | Hindawi

Research Article

Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks

Young-Seob Jeong, Jiyoung Woo, and Ah Reum Kang

SCH Media Labs, Soonchunhyang University, Asan 31538, Republic of Korea

Correspondence should be addressed to Ah Reum Kang; armk@arkang.net

Received 25 January 2019; Accepted 11 March 2019; Published 3 April 2019

Guest Editor: Pelin Angin

With increasing amount of data, the threat of malware keeps growing recently. The malicious actions embedded in nonexecutable documents especially (e.g., PDF files) can be more dangerous, because it is difficult to detect and most users are not aware of such type of malicious attacks. In this paper, we design a convolutional neural network to tackle the malware detection on the PDF files. We collect malicious and benign PDF files and manually label the byte sequences within the files. We intensively examine the structure of the input data and illustrate how we design the proposed network based on the characteristics of data. The proposed network is designed to interpret high-level patterns among collectable spatial clues, thereby predicting whether the given byte sequence has malicious actions or not. By experimental results, we demonstrate that the proposed network outperform several representative machine-learning models as well as other networks with different settings.

1. Introduction

Recently, as the file exchanges increase, intelligent attacks using documents with malicious code are increasing rapidly. Most Internet users are aware of the danger of files attached to mails or websites in forms of execution files. However, because people are not conscious at the documents, they become a good channel to deliver malware. Of the documented malware, PDF-based attack is one of the major attacks because of the flexibility of PDFs in contrast to other document formats. Most malicious PDF documents embed binary or JavaScript codes triggering specific vulnerabilities and perform malicious actions, as described in [1]. Various studies have been conducted to detect such malicious PDFs. The previous studies usually focused on feature extraction from the documents and applied the features to machine-learning models. Some of widely used features include the PDF structure information, entity property, metadata information, encoding method, content property, and lexicon-based features. Although such hand-crafted features have shown successful results, it requires much effort to design the features.

As the exponentially increasing amount of data, deep neural networks are drawing much attention in various fields such as image processing, natural language processing, sensor data processing, and speech recognition [2–6]. One of the main benefits of using the deep neural networks is that it is not necessary to define features because the networks automatically extract or compute features. In this work, we propose a novel approach using convolutional neural network (CNN) to tackle the malware detection. The contributions of this study can be summarized as follows: (1) we design a new CNN model well-suited to the malware detection on PDFs, (2) we demonstrate the performance of the proposed network by experiments using our manually labelled PDF dataset, and (3) we provide specific discussion about the experimental results.

The proposed approach does not require feature definition at all, but it is worth noting that it is still necessary to investigate or study the data structure in order to define better input data or design better network structures. As we target the PDFs in this work, we review the structure of the PDFs intensively and illustrate how we design the proposed network according to the characteristics of the input data (i.e., byte sequences). Although we conduct experiments only with the PDF documents, we expect that the proposed network is easily applicable to other formats (e.g., .rtf files).

In the following section, we review previous studies and the structure of PDF documents.

2. Background

2.1. Malware Detection on Stream Data. Malware is a program written to give an undesirable or harmful effect on a computer system. As the technology of malicious code generation by attackers becomes more intelligent, various researches have been conducted for detection and analysis of malicious codes. The malware can be divided into two categories: executables and nonexecutables. There have been many security programs to detect malicious actions in the form of portable executable files (e.g., Norton, Kaspersky). However, the nonexecutables (e.g., malicious actions in PDF documents) are easy to bypass some existing security programs and there is a high risk of false positives. Such document type malware is known to be more dangerous, as it is often considered as being insignificant by common users.

In order to detect the malicious documents, many studies focused on feature extraction based on the PDF structure analysis, where the features can be seen as a summary of the logical structure. In [7], a format-independent static analysis method, namely, a hierarchical document structure (hidost), was proposed. They adopted a structural multimap, where structural paths are represented by keys and the leaves are indicated by values. Several values of the multimap are reduced to a median value to constitute feature vectors that are used to train machine-learning algorithms. Cuan et al. [8] defined features using PDFiD Python script which verifies objects displayed in PDFs. As a simple trick called gradient-descent attack may bypass the proposed approach, they proposed two solutions: using a threshold for each feature and reducing the number of noncritical features. Smutz and Stavrou [9] extracted features from document metadata, such as the number of characters in titles and the size of images. Li et al. [10] developed a robust and secure feature extractor called FEPDF, which parses and extracts features from PDF documents. FEPDF consists of a matching method, detecting the PDF header, detecting all objects, detecting cross-reference, and detecting trailer. They emphasized that FEPDF can identify new malicious PDFs that have not been identified by existing feature extractors.

Note that these studies commonly require intensive feature engineering process, because it will almost determine the performance (i.e., accuracy) of the malware detection. In this paper, we designed a convolutional neural network to tackle the malware detection on nonexecutables. Although the proposed network allows getting results by just pushing the binary sequences into it without feature engineering, it is still important to investigate the structure of the target data. That is, the neural networks automatically capture features, but the features are obtained from input data which must be defined by an expert. Furthermore, understanding of the data structure helps to design better networks. In this paper, we target the PDF files because PDF-based attacks are known to be one of the major attacks recently. The following subsection provides detailed explanation about the structure of the PDFs.

2.2. Stream Data of PDF Files. The PDF document consists of header, body, cross-reference table, and trailer areas. The header contains document version information, and the body has objects that contain information about the actual document. The cross-reference table carries tables used to refer to objects. The trailer contains the root object and cross-reference table position information among the objects in the body region. The applications of PDFs (e.g., PDF reader) allow execution of JavaScript codes within the files, which means that any functions can be dynamically executed through JavaScript API.

The JavaScript code is contained in a stream, one of the PDF object types, where the object types include Boolean values, arrays, dictionaries, streams, and indirect objects. The stream is a collection of consecutive bytes of binary data that varies in length. The stream is normally supposed to contain large image files or page composition objects. A sample of stream object in a benign PDF file is shown in Figure 1. The object number is 141 and has 1,392 bytes. This stream can be decoded using FlateDecode filter, and the result is depicted in Figure 2. The function in this sample seems normal, but it is obvious that the users will be in danger if some malicious actions are contained in the stream.

In Figure 3, an example of a stream object obtained from a malicious PDF document is shown. The object number is 17 and its length is 1,279 bytes. The decoded beginning part of the malicious JavaScript code is shown in Figure 4, where it uses the 'unescape' function that decodes a string object encoded with 'escape' function. These two functions are intentionally employed to make the users confused, thereby making difficult to detect the malicious actions.

Figures 5 and 6 show another example of a malicious PDF stream object. The JavaScript code in Figure 6 calls the 'replace' function which uses regular expressions to replace certain characters with a desired string. As shown so far, the malicious JavaScript codes often appear obfuscated by encoding, and some functions (e.g., replace, unescape) trigger malicious actions.

There have been studies, of course, that focused on the JavaScript codes in the PDFs. Khitan et al. [12] defined features based on functions, constants, objects, methods, and keywords as well as lexical properties of JavaScript codes. Zhang [13] used features such as the number of objects, number of pages, and stream filtering information of JavaScript codes, together with the PDF structure information, entity characteristics, metadata information, and content statistics attributes. Liu et al. [1] proposed a context-aware approach based on the observation that the malicious JavaScript works differently from the normal JavaScript. This approach monitors suspicious activity based on JavaScript statements by putting the original code as an argument to 'eval' function and then opens the PDF document. The 'eval' function executes the delivered executable JavaScript code and returns a result. That is, it executes the code in a separated environment and finds suspicious codes running dropped malware.

In this study, we do not perform lexical analysis on JavaScript code or run PDF documents for dynamic analysis as in previous studies. We design a convolutional neural

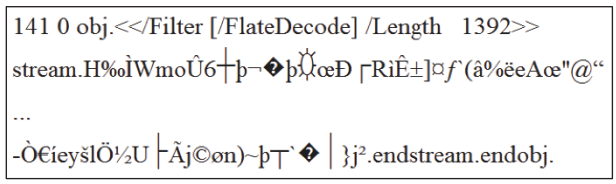

Figure 1: Example of a stream object in a positive PDF file.

```
//*********************************
// Section 1 functions

function SetSection1(oDlgLit)
{
    // Section 1 field data
            this.getField("UserTitle").value = Section1.strTitle;
            this.getField("UserFirstName").value = Section1.strFirstName;
            this.getField("UserLastName").value = Section1.strLastName;
            this.getField("UserStreet").value = Section1.strStreet;
            this.getField("UserCity").value = Section1.strCity;
            this.getField("UserZip").value = Section1.strZip.toString();
}
```

Figure 2: Example of a decoded stream content of a positive PDF file.

network that takes a byte sequence of a stream as an input and predicts whether the input sequence contains malicious actions or not. In next subsection, we briefly review the previous studies adopting neural networks to tackle the malware detection task.

2.3. Neural Networks for Malware Detection. There have been few studies thus far in applying neural networks to malicious software (malware) detection. Most recent works among them have used features extracted through dynamic analysis, so the features are extracted under the binary run in a virtualized environment. Kolosnjaji et al. [14] proposed a combination of convolutions and long short-term memory (LSTM) [15] to classify malware types based on the features of the API call sequences. Huang and Strokes [16] defined a manual 114 high-level features out of API calls as well as original function calls to predict malware types. This approach is essentially composed of two models, malware detection and malware type classification. The authors argued that the shared parameters of the two models contribute to improving the overall performance. These studies of dynamic analysis are performed on a certain nonpublic emulation environments, which makes it difficult to reproduce the works.

The other line of malware detection is the static analysis, in which features are obtained from the files without running them. Raff et al. [17] applied neural networks to the malware identification problem using the features in forms of 300 raw bytes of the PE-header of each file. This work showed

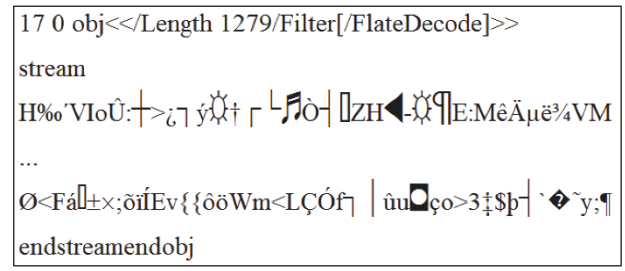

Figure 3: Example of a stream object in a malicious PDF file.

```
var unes=unescape;
var pGvRIJZpqdN = unes("%u4143%u494b%u11EB%u5BFC%u334B
%u66C9%ub0B9%u8001%u0B34%uE2f9"+
"\x25\x75EBFA\x25\x75E805\x25\x75FFEB\x25\x75FFFF"+
"%uF911%uF9F9%uA3F9%u72AC%u7815%u9D15%uF9FD%u72F9"+
"%u110D%uF869%uF9F9%u0172%u1611%uF9F9%u70F9%u06FF"+
"%u91CF%u6254%u2684%uED11%uF9F8%u70F9%uF5BF%uCF06"+
"%uD091%u3FEB%u11AF%uF8FC%uF9F9%uBF70%u06E9%u91CF"+
"%uC5A0%u82FE%u0F11%uF9F9%u70F9%uEDBF%uCF06%u8791"+
```

Figure 4: Example of a decoded stream content of a malicious PDF file.

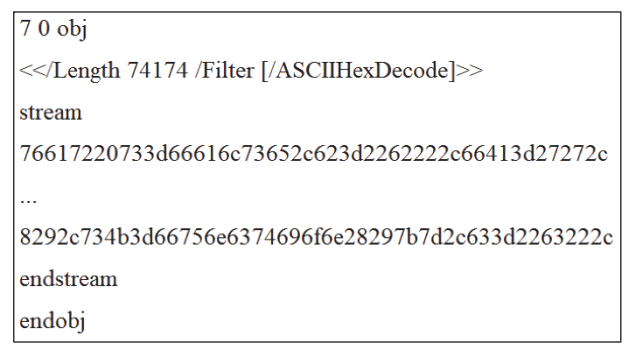

Figure 5: Another example of a stream object in a malicious PDF file.

```
var s=false,b="b",fA=",array=",l=new Date(),i="i",jR=new Date(),var
h=false,jX="jX",m=2571,bdktvx=String,ehkrvz=String,dlnr="94",fhvy=0,
adgv="",adjo=bdktvx['eMvBaMlB'.replace(/[B3M7f]/g,'')],dejoqw=
[70,62,66,154,174,162,156,168,162,163,167,84,159,157,177,147,162,1
68,97,173,154,166,172,164,101,84,165,153,167,93,180,65,67,61,66,17
1,161,157,165,153,89,92,178,149,171,167,169,98,165,153,167,155,17
3,156,89,94,89,102,89,112,89,160,158,162,98,175,70,62,66,61,66,173,
154,166,172,164,89,95,118,84,178,149,171,167,169,111,70,62,66,61,1
82,65,67,61,66,173,154,166,172,164,89,113,89,173,154,166,172,164,1
03,167,174,150,172,168,171,157,167,155,97,100,101,84,165,153,167,
```

Figure 6: Another example of a decoded stream content of a malicious PDF file.

that the neural networks are capable of extracting underlying high-level interpretation from the raw bytes, which in turn makes it possible to develop malware detectors without hand-crafted features. Saxe and Berlin [18] utilized a histogram of byte entropy values from the entire files and defined a fixed length feature vector as the input of the neural networks. Le et al. [19] designed CNN-BiLSTM architecture, where the rational of taking bidirectional LSTM (BiLSTM) layer on top of CNN layer is that the BiLSTM layer may interpret sequential dependencies between different pieces generated by the CNN layer. They showed that the CNN layer is effective in representing local patterns of fixed length, and the BiLSTM has a potential to capture arbitrary sequential dependencies of executables. Raff et al. [20] derived a feature vector from the raw bytes and designed a shallow convolutional neural network with a gated convolutional layer, a global max-pooling layer, and a fully connected output layer. They insisted that their work is the first to define a feature vector from the entire binary, and it was hard to develop deeper networks due to the extraordinarily long byte strings (e.g., 1-2M length). Their biggest contribution is that the feature vector is obtained from the entire binary, so that it may grasp the global context of the entire binary. That is, the contents of a binary may have the high amount of positional variation or can be rearranged in arbitrary ordering, so they adopted the global-level feature vectors with a very large dimension. Although their network is designed to have ability to scale well with variable length of binary strings, it essentially will not be applicable to any longer binary strings (e.g., 3-4M length). All of these studies commonly utilized raw bytes of executables.

The proposed network in this paper is designed to take a byte sequence within the nonexecutables as an input and generates an output based on high-level patterns of collectable spatial clues, which implies that the proposed network is applicable to byte sequences with variable lengths. In the following section, we illustrate how we design the proposed network according to the characteristics of the input data (i.e., byte sequences).

3. Proposed Method

To detect malicious actions without heavy feature engineering, we designed a deep learning model against stream objects and discriminate the maliciousness in the object level. The stream objects have no size limitation, and a certain part of the stream exhibits malicious actions while the other part does not. Such high-level location invariance makes it difficult to detect the maliciousness of the object. Among deep learning models, convolutional neural networks (CNN) are known as successful in detecting locally contextual patterns. The CNN models have brought dramatic performance advance in the area of image processing [21, 22], and one of its benefits is that it works with smaller amount of data, compared to other deep learning models (e.g., recurrent neural networks, fullyconnected neural networks).

A graphical representation of our proposed network is depicted in Figure 7, where the network consists of an embedding layer, two convolutional layers, a pooling layer, a fully connected layer, and an output layer. Note that the figure just shows a structure of the network, and the dimensions of the layers and the number of channels must be much larger. The E denotes the embedding size and S means a sequence length. Rather than directly submitting the raw byte values into convolution (i.e., using a scaled version of a byte's value from 0 to 255), we adopt an embedding layer to map each byte to a E-dimensional feature vector because the byte values do not imply intensity but convey some contextual information. That is, given a byte sequence of length S, the $E \times S$ real-valued embedding matrix is computed during the training process, so that the matrix will help to grasp a wider breadth of input patterns.

The embedding layer makes it possible to represent meaning of each byte by incorporating all the byte sequences. As discussed in [20], the raw values of byte sequences do not simply represent intensity, and it will be better to find an alternative way to see the values. For example, a byte value 160 does not imply 'better' or 'stronger' intensity than 130, but the two values must convey different meaning. In the 'word embedding' concept in the natural language processing (NLP) field, similar words (e.g., 'hi' and 'hello') are close to each other in the embedding space, whereas opposite words are far from each other. Likewise, our embedding layer interprets the contextual meaning of byte values and represents them on the embedding space.

Several locally adjacent E-dimensional vectors generated from the embedding layer are then fed into the first convolutional layer followed by another convolutional layer. The first convolutional layer is designed to take a $C_1 \times E$ matrix which is supposed to carry spatial clues of malicious actions, in the hope that the collected clues will be enough for the entire network to make a wise decision. In the recent work [20], a convolutional neural network designed for analysing the entire sequence at once was proposed, but this network will not be applicable to longer sequences. In contrast, our network collects simple local clues and generates high-level representation, which means that the proposed network is available to all sequences with variable lengths. Each convolutional layer takes one or more adjacent vectors (or values) from its previous layer as an input and generates an output value by a summation of element-wise multiplication with a filter. This filter, also known as a kernel, is computed during the training process.

The two convolutional layers have K_1 and K_2 distinct channels (i.e., kernels), respectively, to find different spatial patterns. The convolutional layer slides or convolves from the top-left corner to the bottom-right corner with an arbitrary stride, thereby resulting matrix or vector will carry a compilation of spatial patterns throughout the input. Similar to many studies related to CNN [2, 23, 24], our network has consecutive convolutional layers because multiple stacked convolutional layers are known to have better (higher-level) representation than a single convolutional layer. We have two convolutional layers stacked along the network depth, where the first convolutional layer takes a $C_1 \times E$ matrix as an input, and the second convolutional layer takes a $C_2 \times 1$ vector as an input, respectively. The first layer is supposed to catch simple clues, which will be used for the second

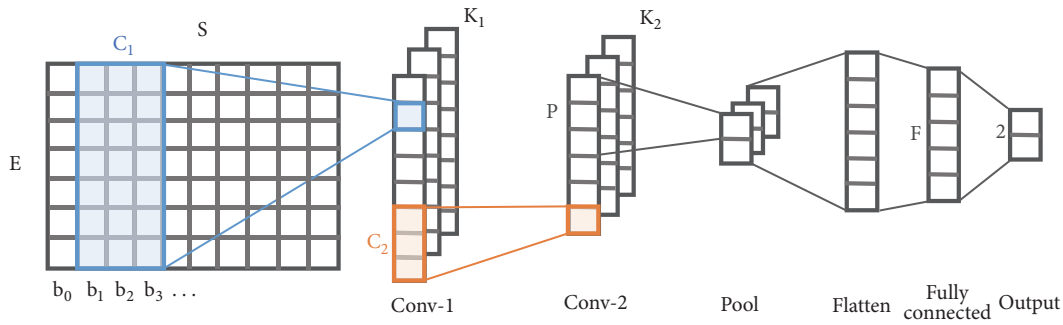

Figure 7: Graphical representation of the proposed convolutional neural network.

layer to understand underlying higher-level patterns. For example, the first layer may capture simple clues of JavaScript statements (e.g., replace), while the second layer represents contextual information about their types or arguments.

One may argue that stacking more convolutional layers might be better, because the deeper network is often known to be capable of extracting more complicated patterns. This might be true, but it should be noted that the deeper network is not always better than the shallow networks. The length of network must be considered carefully by investigating the data; too complicated network will probably overfit, whereas too simple network will underfit. By examination into the sequence data, we conclude that the two convolutional layers will be enough to capture the variety of the spatial patterns of malicious actions. We also, of course, show that this structure is indeed better than deeper networks through experiments.

The high-level representation obtained from the two consecutive convolutional layers is submitted to the pooling layer that helps to focus on some representative or primary patterns. Among several types of pooling layer such as average-pooling and L2-norm pooling, we adopt the max-pooling layer as it is generally known to be effective in various tasks. Similar to the convolutional layer, the pooling layer slides from the top-left corner to the bottom-right corner with an arbitrary stride, resulting in output vectors of much smaller size. The output vectors are flattened or concatenated to form a one-dimensional vector that will be passed to the F-dimensional fully connected (FC) layer. The FC layer collects the primary patterns from the pooled values, and the last output layer represents how likely the given byte sequence embeds malicious actions.

4. Experiments

4.1. Dataset. We collected PDF files from various antivirus or antimalware companies and constructed a dataset through manual labelling process. There often exist several stream objects in a malicious PDF file, and one of more streams contains malicious actions. As we observed that all malicious streams contain JavaScript codes, we mark the objects as 'malicious' if they contain JavaScript codes, otherwise 'benign'. The objects of benign PDF files are all marked as 'benign'. Note that we do not employ whether to contain JavaScript as a feature at all. The proposed network will see only the raw byte sequence. The object streams are hexadecimal representations of text streams, as shown in Figure 8, where each line of the figure is a part of the conversion of Figures 2, 4, and 6, respectively.

Table 1: Data statistics, where the positive and negative percentages imply the proportions of the malicious streams and the benign streams, respectively.

Size (# of instances)	1,978
Positive : Negative (%)	50.0 : 50.0
Length (dimension)	1,000

The data statistics are summarized in Table 1, where each data instance is defined as a byte-level stream object. As shown in Figure 9, for each data file, multiple byte streams are collected, each of which is manually labelled with a positive (malicious) or a negative (benign) tag. The total amount of originally collected byte sequences was 4,371, but randomly downsampled 989 negative instances.

Note that the byte streams have different lengths as described in Figure 10, but the proposed network cannot handle such sequences of variable lengths. As the network is designed to grasp high-level patterns from collectable spatial clues of malicious actions, it is not necessary to use the whole sequence of different length at a time. Rather, we padded short sequences and cut away the remaining part from long sequences, so that all sequences have the same length of 1,000. When the network is trained with these fixed length sequences, it can be applied to divided byte sequences with the same size in a target file, so that it will predict whether the target file contains malware or not.

4.2. Model. We compared the proposed network with several representative classification methods such as support vector machine [25], decision tree, naïve bayes, and random forest [26]. The brief description and parameter settings of the methods are summarized in Table 2.

We also examined various settings for our network, and the best setting was found as follows: (1) the embedding dimension $E = 25$, (2) C_1 and C_2 are set to 3, respectively, (3) pooling layer dimension $P = 100$, (4) K_1 and K_2 are 32 and 64, respectively, and (5) fully connected layer dimension $F = 128$, where the notations can be found in Figure 7. The two

```
2F 2F 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A
76 61 72 20 75 6E 65 73 3D 75 6E 65 73 63 61 70 65 3B 0D 0A 76 61 72 20 70 47 76 52 49 4A 5A 70 71
76 61 72 20 73 3D 66 61 6C 73 65 2C 62 3D 22 62 22 2C 66 41 3D 27 27 2C 61 72 72 61 79 3D 27 27 2C
```

Figure 8: Byte streams.

Table 2: Description of comparative classifiers and parameter settings.

Model	Description
Naïve bayes (NB)	(i) Probabilistic classifier based on the bayes' theorem (ii) Assumes the independence between features
Decision tree (DT)	(i) C4.5 classifier using J48 algorithm (ii) Confidence factor for pruning: 0.25
Support vector machine (SVM)	(i) Non-probabilistic binary classification model that finds a decision boundary with a maximum distance between two classes (ii) Kernel: Poly (iii) Exponent=1.0, Complexity c=1.0 (iv) Trained via sequential minimal optimization algorithm [11]
Random forest (RF)	(i) Kind of ensemble model that generates final result by incorporating results of multiple decision-trees (ii) #trees=100 (iii) #features=log(#trees)+1 (iv) Each tree has no depth-limitation

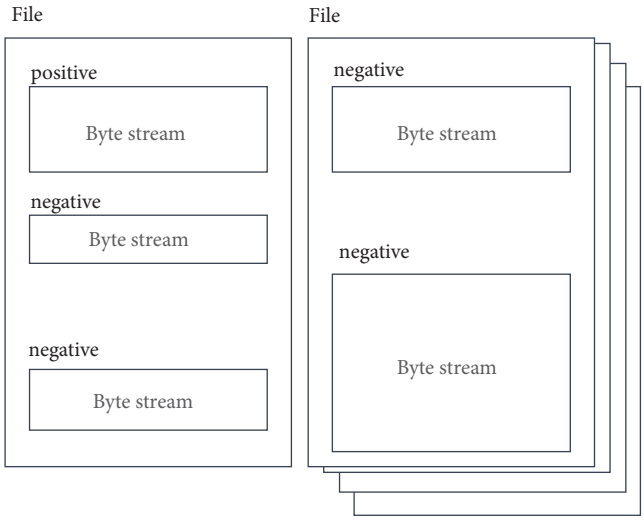

Figure 9: Positive or negative tags for each byte stream in the data files.

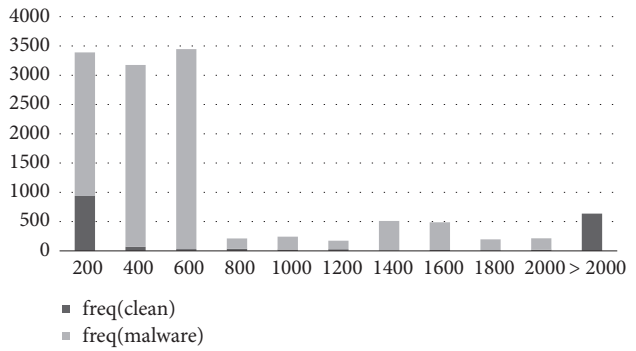

Figure 10: Frequencies of clean and malware PDF byte streams with different lengths, where the x-axis represents stream lengths and the y-axis shows frequencies.

convolutional layers have strides (1, 25) and (1, 1), respectively, and the pooling layer has a nonoverlapping stride (100, 1). The sequence length S must be 1,000. Every layer takes rectified linear unit (ReLU) as an activation function except for the output layer that takes a softmax function. The cost function is defined as a cross entropy over all nodes of the output layer. Consequently, the total numbers of trainable parameters are 89,371.

We adopt Adam's optimizer [27] with an initial learning rate of 0.001 to train our network. Our training recipe is as follows: (1) L2 gradient-clipping with 0.5, (2) drop-out [28] with a keeping probability 0.25 for the fully connected layer, and (3) batch normalization [29] for the two convolutional layers. We tried to use the regularization methods such as L2 regularization and decov [30] and observed no performance improvements, as the batch normalization and drop-out are known to have regularization effect themselves. The weight matrices of the convolutional layers, the FC layer, and the output layer are initialized by He's algorithm [31], and bias vectors are initialized with zeros. In the following

Table 3: Experimental results with the PDF dataset, where the two values of each cell are of 'benign' and 'malicious', respectively.

Model	Precision	Recall	F1
DT	96.00 / 90.30	89.70 / 96.30	92.70 / 93.20
NB	88.40 / 99.70	99.70 / 87.00	93.70 / 92.90
SVM	94.70 / 98.90	99.00 / 94.40	96.80 / 96.60
RF	93.50 / 99.40	99.40 / 93.10	96.40 / 96.10
Emb+Conv+Conv+Pool+FC	99.76 / 100.0	97.37 / 97.37	*98.48 / 98.65*
Conv+Conv+Pool+FC	99.78 / 100.0	92.62 / 97.27	95.71 / 98.61
Emb+Conv+Pool+FC	99.73 / 100.0	94.94 / 97.78	97.12 / 98.87
Emb+Conv+Conv+FC	99.67 / 100.0	97.78 / 92.32	98.55 / 96.00
Emb+Conv+Conv+Conv+Pool+FC	99.70 / 100.0	92.21 / 95.35	95.36 / 97.60

subsection, we demonstrate the performance of our network by experimental comparison with the other classifiers and different networks.

4.3. Result. One unfortunate aspect of the field of malware detection is that there is no available public dataset for various reasons. The dataset easily obtainable from public is often not of a sufficient quality, so previous studies could not compare performance (i.e., accuracy) across works because of different data characteristics and labelling procedures. It is inevitably hard to compare our performance with other state-of-the-art studies for the same reason. We compare our network with some comparative machine-learning methods and CNN models with different settings. The measurement is conducted using 10-fold cross validation, where the performance values (i.e., F1 score, precision, and recall) are averaged values of three distinct trials.

The experimental results are described in Table 3, where the two values for each cell correspond to the 'benign' and 'malicious' classes, respectively. For example, the F1 scores of random forest (RF) are 96.4 for 'benign' and 96.1 for 'malicious'. For the four traditional machine-learning models (e.g., DT, NB, SVM, and RF), the values of the input sequence are treated as nominal values. We tested five different structures of CNNs. The first network, Emb+Conv+Conv+Pool+FC, is the best structure which has an embedding layer, two consecutive convolutional layers, a pooling layer, and a fully connected layer followed by an output layer. The number of epoch differs from networks according to the complexity (i.e., the number of layers and parameters) of the networks and the dataset size. For instance, the first network is trained through 30 epochs, whereas training of the second network requires 25 epochs.

Among the traditional machine-learning models, the support vector machine (SVM) exhibits the best F1 scores and random forest (RF) has the comparable results. As also shown in Table 3, it seems obvious that the five CNNs outperform the traditional machine-learning models. From the results of the five networks, we can find two main observations. First, the embedding layer (Emb) seems to play a significant role in better representation of byte sequences, as the second network without the embedding layer exhibits much worse F1 scores than the other networks. Second, the stacked convolutional layers enable interpreting high-level patterns, and the optimal number of layers seems to be two. The third network having a single convolutional layer and the fifth network having three convolutional layers are worse than the first network. The fifth network especially has the worst F1 score among five networks. This indicates that more stacked layers are not always better than shallow networks.

4.4. Discussion. The experimental results can be summarized in two aspects. First, the convolutional neural networks showed superior performance than the traditional machine-learning models. The F1 score of the proposed network is almost 2% greater than the SVM, which can be explained that the convolutional neural networks have better comprehensive power to analyse the underlying spatial patterns of the byte sequences. Second, it seems that the embedding layer followed by the two convolutional layers is best suited to representing high-level patterns of malicious actions. Less or more stacked convolutional layers gave worse results.

Other than the two aspects, we need to discuss about the parameter settings for training. The results of Table 3 are obtained from the network trained with the dimensions and the parameters as aforementioned in Model part. We found the optimal parameter setting by grid search, and some remarkable results with different settings and dimensions are summarized in Table 4. The first two rows correspond to the embedding size E, and the last two rows are associated with the pooling size P. Other remaining rows are related to the training recipe, such as drop-out, gradient-clipping, and batch normalization. The drop-out together with the batch normalization was helpful to generalize the network, and we observed no improvements with additional regularization methods. The gradient-clipping made the network more robust, as it prevented from tripping over desirable points of the gradient space.

We also checked the training time of the comparable methods. Table 5 shows the elapsed seconds for training different methods. The training of the four traditional methods is performed on a machine of Xeon E5-2620 V4 with 128GB RAM, while the CNN models are trained using a machine of i7-9700K with 64GB RAM and two RTX 2080 Ti. The training time of the CNN models strongly depends on the performance of GPUs. All methods are trained using the 1,978 instances, and the number of epochs is equally 30 for the CNN models. Note that the fourth CNN model without a

Table 4: Experimental results with different settings, where the two values of each cell are of 'benign' and 'malicious', respectively.

Dimensions & Training options	Precision	Recall	F1
$E = 15$	99.68 / 100.0	92.52 / 95.75	95.34 / 97.82
$E = 35$	99.73 / 100.0	94.03 / 97.47	96.58 / 98.71
Drop-out with 0.5	99.70 / 100.0	93.53 / 97.37	96.23 / 98.66
L2 gradient-clipping with 0.3	99.74 / 100.0	95.25 / 97.17	97.17 / 98.55
L2 gradient-clipping with 0.7	99.70 / 100.0	92.11 / 97.78	95.14 / 98.86
w/o batch-normalization	99.70 / 100.0	95.96 / 97.27	97.68 / 98.61
$P = 50$	99.70 / 100.0	95.05 / 97.07	97.13 / 98.50
$P = 150$	99.76 / 100.0	93.83 / 97.88	96.29 / 98.92

Table 5: Training time of comparable methods.

Model	Time (seconds)
DT	0.69
NB	0.16
SVM	750.88
RF	1.56
Emb+Conv+Conv+Pool+FC	22.07
Conv+Conv+Pool+FC	21.12
Emb+Conv+Pool+FC	13.85
Emb+Conv+Conv+FC	31.52
Emb+Conv+Conv+Conv+Pool+FC	23.16

pooling layer takes much longer time than the other three CNN models. The reason is that the pooling layer has an effect of reducing the filter sizes, so the fourth CNN model has greater number of parameters to train. Except for the SVM, the traditional models seem computationally more efficient than the CNN models. One of the reasons for the lagging of training SVM must be the use of Poly kernel function. We may use another kernel functions (e.g., linear kernel), of course, but it will probably degrade accuracy. As the training of random forest (RF) is much faster than the CNN models, it might be preferred to choose the RF model if we want efficiency with a small loss of effectiveness (i.e., accuracy).

5. Conclusion

The threat of malicious documents keeps growing, because it is difficult to detect the malicious actions within the documents. In this work, we proposed a new convolutional neural network designed to take a byte sequence of nonexecutables as an input and predicts whether the given sequence has malicious actions or not. We illustrated how we design the network according to the characteristics of the input data and provided discussion about the experiments using the manually labelled dataset. The experimental results showed that the proposed network outperforms several representative machine-learning models as well as other convolutional neural networks with different settings. Though we conducted experiments only with the PDF files, we expect that this approach can be applicable to other types of data if they contain byte streams. Therefore, as a future work, we will collect data of other file types (e.g., .rtf files) and perform further investigation.

Data Availability

We disclose our dataset as well as a code to public (https://sites.google.com/view/datasets-for-public).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Soonchunhyang University Research Fund (no. 20170265). This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A3B030360 50).

References

[1] D. Liu, H. Wang, and A. Stavrou, "Detecting malicious JavaScript in pdf through document instrumentation," in *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 100–111, IEEE, Atlanta, GA, USA, June 2014.

[2] T. T. Um, F. M. J. Pfister, D. Pichler et al., "Data augmentation of wearable sensor data for Parkinson's disease monitoring using convolutional neural networks," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pp. 216–220, ACM, Glasgow, Scotland, 2017.

[3] Z. M. Kim, Y. S. Jeong, H. R. Oh et al., "Investigating the impact of possession-way of a smartphone on action recognition," *Sensors*, vol. 16, no. 6, pp. 1–5, 2016.

[4] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1746–1751, Association for Computational Linguistics, Doha, Qatar, October 2014.

[5] A. Hannun, C. Case, and J. Casper, "Deep speech: scaling up end-to-end speech recognition," *Computing Research Repository*, pp. 1–12, 2014.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern*

Recognition, pp. 779–788, IEEE, Las Vegas Valley, NV, USA, July 2016.

[7] N. Šrndić and P. Laskov, "Hidost: a static machine-learning-based detector of malicious files," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 22, 2016.

[8] B. Cuan, A. Damien, C. Delaplace et al., *Malware Detection in PDF Files Using Machine Learning [PhD. Thesis]*, REDOCS, 2018.

[9] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 239–248, Orlando, Fla, USA, December 2012.

[10] M. Li, Y. Liu, M. Yu et al., "FEPDF: a robust feature extractor for malicious PDF detection," in *Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 218–224, IEEE, Sydney, Australia, August 2017.

[11] J. C. Platt, "Sequential minimal optimization: a fast algorithm for training support vector machines," in *Advances in Kernel Methods – Support Vector Learning*, MIT Press, Cambridge, Mass, USA, 1998.

[12] S. J. Khitan, A. Hadi, and J. Atoum, "PDF forensic analysis system using YARA," *International Journal of Computer Science and Network Security*, vol. 17, no. 5, pp. 77–85, 2017.

[13] J. Zhang, "MLPdf: an effective machine learning based approach for PDF malware detection," *Security and Cryptography*, 2018.

[14] B. Kolosnjaji, A. Zarras, G. Webster et al., "Deep learning for classification of malware system call sequences," *Lecture Notes in Computer Science*, vol. 9992, pp. 137–149, 2016.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] W. Huang and J. W. Stokes, "MtNet: a multi-task neural network for dynamic malware classification," *Lecture Notes in Computer Science*, vol. 9721, pp. 399–418, 2016.

[17] E. Raff, J. Sylvester, and C. Nicholas, "Learning the PE header, malware detection with minimal domain knowledge," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 121–132, ACM, Dallas, TX, USA, 2017.

[18] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of the 10th International Conference on Malicious and Unwanted Software*, pp. 11–20, IEEE, Fajardo, PR, USA, October 2015.

[19] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digital Investigation*, vol. 26, pp. S118–S126, 2018.

[20] E. Raff, J. B. Barker, J. Sylvester et al., "Malware detection by eating a whole EXE," in *Proceedings of the in Proceedings of the Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 268–276, New Orleans, LA, USA, 2018.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems 25*, Lake Tahoe, NV, USA, December 2012.

[23] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, IEEE, Boston, MA, USA, June 2015.

[24] G. Huang, Z. Liu, L. Maaten et al., "Densely connected convolutional networks," in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269, IEEE, Honolulu, HI, USA, July 2017.

[25] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, ACM, Pittsburgh, PA, USA, July 1992.

[26] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," in *Proceedings of the in Proceedings of the 3rd International Conference for Learning Representations*, San Diego, Calif, USA, 2015.

[28] N. Srivastava, G. Hinton, A. Krizhevsky et al., "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[29] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456, ACM, Lille, France, July 2015.

[30] M. Cogswell, F. Ahmed, R. Girshick et al., "Reducing overfitting in deep networks by decorrelating representations," in *Proceedings of the 4th International Conference for Learning Representations*, San Juan, PR, USA, 2016.

[31] K. He, X. Zhang, S. Ren et al., "Delving deep into rectifiers: surpassing human-level performance on imagenet classification," in *Proceedings of the 15th IEEE International Conference on Computer Vision*, pp. 1026–1034, Santiago, Chile, December 2015.