IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 1, 2022, accepted April 25, 2022, date of publication May 10, 2022, date of current version May 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3171775

Malware Detection Using Byte Streams of Different File Formats

YOUNG-SEOB JEONG[1], SANG-MIN LEE[2], JONG-HYUN KIM[2], JIYOUNG WOO[3], AND AH REUM KANG[4]

[1]Department of Computer Engineering, Chungbuk National University, Cheongju-si 28644, South Korea
[2]Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea
[3]Department of Big Data Engineering, Soonchunhyang University, Asan-si 31538, South Korea
[4]Department of Information Security, Pai Chai University, Daejeon 35345, South Korea

Corresponding author: Ah Reum Kang (armk@pcu.ac.kr)

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00026, ICT infrastructure protection against intelligent malware threats, 50%). This work was supported by Chungbuk National University BK21 program(2021) (50%).

ABSTRACT Malware detection is becoming more important task as we face more data on the Internet. Web users are vulnerable to non-executable files such as Word files and Hangul Word Processor files because they usually open such files without paying attention. As new infected non-executables keep appearing, deep-learning models are drawing attention because they are known to be effective and have better generalization power. Especially, the deep-learning models have been used to learn arbitrary patterns from byte streams, and they exhibited successful performance on malware detection task. Although there have been malware detection studies using the deep-learning models, they commonly aimed at a single file format and did not take using different formats into consideration. In this paper, we assume that different file formats may contribute to each other, and deep-learning models will have a better chance to learn more promising patterns for better performance. We demonstrate that this assumption is possible by experimental results with our annotated datasets of two different file formats (e.g., Portable Document Format (PDF) and Hangul Word Processor (HWP)).

INDEX TERMS Malware detection, byte stream, non-executables, deep learning, convolutional neural networks, Hangul word processor, portable document format.

I. INTRODUCTION

Malware detection is an important task as more data transfers on the Internet. Malware has been used to attack individuals, companies, or institutions. It may just simply remove or encrypt files of the victims, or utilize the victims as weapons (e.g., zombie hosts) to attack ultimate targets. As described in [1], the target files of attackers might be classified into two categories: non-executables (e.g., Portable Document Format (PDF) files) and executables (e.g., EXE files). The victims are more vulnerable to non-executables such as PDF and Microsoft Word files because they usually open such files without paying much attention. Therefore, it is becoming more important to automatically assess how malicious the files without opening them.

There are mainly two ways of malware detection: static analysis and dynamic analysis. The dynamic analysis detects malicious actions by looking at all of the step-by-step actions conducted in an isolated virtual environment (e.g., virtual box), whereas the static analysis finds clues of malicious actions by examining the files without running them. The dynamic analysis has a non-preferred side that different studies use different non-public emulation environments, meaning that they are usually not reproducible. Recently, there were few studies that analyzes byte streams or sequences within non-executables in the static manner [1]–[3]. The motivation of these studies is that the data-driven models, especially deep learning models, automatically find arbitrary patterns (e.g., relation between bytes) beneath the byte streams, so the trained models will probably be more robust to future variants. These studies, however, have particular target formats (e.g., PDF), and did not consider utilizing byte streams of different formats at the same time.

In this paper, we assume that byte streams of different non-executable formats complement each other, so we may expect performance improvements if deep learning (DL)

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

models learn from them. Specifically, we investigate two different non-executable formats (e.g., PDF and Hangul Word Processor (HWP)), and explain a motivation of using the two different formats for malware detection. We demonstrate the benefit of it by experimental results of malware detection using our annotated datasets. As far as we know, this is the first study that shows possibility of using different non-executable formats for performance improvements on malware detection task. We also examine the impact of data size and analyze some cases that might be related to our experimental results.

II. RELATED WORK

Malware detection is basically a binary classification task; we need to predict a label (e.g., benign, malware) of a given input (e.g., features extracted from files, byte stream). There have been many studies of data-driven approach to tackle the malware detection task in the static manner. The most widely-used machine-learning (ML) models are support vector machines (SVM) [4], logistic regression (LR), decision trees, Naive bayes, ensemble models (e.g., random forest (RF) [5], and extreme gradient boost (XGB) [6]). For example, the RF achieved about 89% and 96% of F1 scores on portable executable (PE) files and OpCode sequences, respectively [7]. Another study of [8] showed that the SVM learned arbitrary patterns from a frequency histogram obtained from executables and achieved 95% of true positive rate (TPR). The XGB was employed to predict potential malicious actions within the byte sequences of PE files [9], and achieved about 98% accuracy on the dataset of [10]. A variant of gradient boosting model using mutual information and feature importance was applied to byte n-grams [11], and the proposed model had F1 scores of 98∼99% for Android malware detection. Although these studies have shown successful performance, they have a common limitation that they require much effort of experts to feature engineering; so it requires extensive feature engineering for every newly appearing malware.

Deep-learning (DL) models are drawing much attention, and they are known to be a solution for the limitation of the traditional ML models because they are capable of capturing latent features automatically. The DL models are multi-layered perceptron (MLP) with a deep structure (i.e., many hidden layers), and the deep-learning technique is essentially a part of the machine-learning technique. As deeper structure is known to have a big power to discover patterns from data, there have been many studies that proposed different techniques (e.g., residual connection [12], batch normalization [13], drop-out [14]) for allowing the model to have more hidden layers without suffering from the problem of vanishing (or exploding) gradient. Other than the standard structure of MLP model, there are several well-known types of DL models such as recurrent neural networks (RNN) [15], convolutional neural networks (CNN) [16], attention-based models [17], and graph neural networks (GNN) [18]; of course, there are many studies that designed hybrid models consisting of two or more types.

There are few studies that utilized the power of the DL models for malware detection by analyzing byte streams, and CNN-based models are drawing attention because of its efficiency (e.g., speed) without losing much effectiveness (e.g., accuracy). The byte stream is a sequence of bytes, and every file is basically a byte stream. There is only a difference in the way of interpreting the information stored in the file depending on the file format. For example, a text file consists of a set of human-readable strings. However, in the end, since only bits such as 0 and 1 are stored in the computer memory, a byte stream can be extracted from all files in byte units.

A part of byte stream is also a byte stream, and some previous studies assumed that byte streams having malicious actions within files might have particular patterns that can be used for malware detection. For example, Raff *et al.* [3] designed a shallow structure using convolutional layers that analyze byte streams of portable executable (PE) headers. Their shallow architecture takes a byte stream of 1-2M length as an input, and achieved about 94% accuracy. Note that they used the byte stream as an input for the model without employing any other hand-crafted features. Another architecture using consecutive convolutional layers was designed for analyzing byte streams of PDF files [2], one of the most widely-used non-executables, and it achieved 98% of F1 score. Jeong *et al.* proposed a CNN model using spatial pyramid structure to grasp the underlying patterns of Hangul word processor (HWP) files that is a well-known non-executable in South Korea [1], where new malware attacks via the HWP files keep appearing due to the circumstance between the South and North Korea. Although these existing studies achieved successful performance, they commonly focused on a single file format (e.g., PDF). In this paper, we investigate to use data of multiple formats, and demonstrate it by experimental results of malware detection task.

III. PROPOSED METHOD

This paper aims at solving the malware detection task that is basically a binary classification; we want to develop a model that predicts a label (malware or benign) of a given byte stream of a non-executable. The overview of our method is depicted in Fig. 1. We basically utilize byte streams of multiple file formats for training a classification model for a specific file format.

A. MALWARE DETECTION USING BYTE STREAMS OF MULTIPLE FORMATS

Assuming that we have a target file format f_t of the malware detection task, and we have training datasets $\mathbf{D}^F_{train}$ for a certain list of file formats $F = \{f_1, f_2, \ldots, f_{|F|}\}$ including the target format f_t as well as a test dataset $D^{f_t}_{test}$. Note that the training datasets cover all file formats, whereas the test dataset is only for the target format, as shown in Fig. 1.

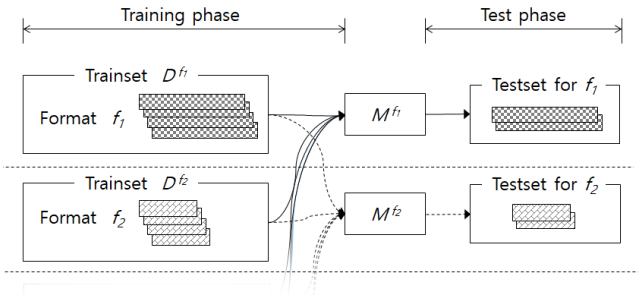

FIGURE 1. Overview of the proposed method.

When the malware detection model M^{f_t} is trained in a supervised manner for the target format, the model is trained not only with $D^{f_t}_{train}$ but also with all other datasets $\mathbf{D}^{F}_{train}$. If we want to have a separated validation set $D^{f_t}_{val}$ out of the training datasets, the validation set should be sampled from $D^{f_t}_{train}$ but not from other datasets in order to make the validation set to have similar characteristics to the test set.

There is a learning concept, *multi-task learning* (MTL), that allows a model to learn and solve two or more tasks at the same time. The motivation behind the MTL concept is that the model may learn better through the relationship between multiple tasks. The MTL is different from *transfer learning*; the MTL aims at solving all the tasks at the same time whereas the transfer learning exploits a source task to solve a target task. The MTL is mostly implemented by parameter sharing that allows the parameters are trained for two or more different tasks at the same time. There are two parameter sharing ways of the MTL concept: soft sharing and hard sharing. Most previous work of MTL concept adopts the hard sharing; it has a single model that has parameters shared across different tasks. The soft sharing way assumes that there are task-specific models that share their parameters. These ways are somewhat related to our proposed method, but they are different because the MTL concept assumes that the model takes the same or similar input for solving multiple tasks closely related to each other. For example, [19] applied the MTL concept for malware detection and malware classification tasks, and their model takes the same shape of sparse binary input vector for solving the tasks; that is, it assumes that we want to solve the two tasks at the same time for a given input. Similarly in [20], their model takes the same input sequence of API calls for solving malware classification and file access pattern generation task. On the other hand, we aim at malware detection task for a specific file format because we will not want to have to run the model on two or more file formats every time. We assume that byte streams of multiple different file formats may complement to each other, and propose a way of utilizing the byte streams of different file formats for training malware detection model.

Even though our method is generally applicable to any set of file formats, we focused two non-executables, PDF and HWP, in this paper; the number of file formats $|F| = 2$. There are two reasons for this. First, the PDF is one of the most widely-used file formats in the world as reported in [21], so we chose the PDF as it seems to contain more diverse malicious patterns than other formats. Second, because of the special situation between South Korea and North Korea, North Korea continuously cyber-attack South Korea, and malicious code attacks on HWP files is increasing. As the HWP files are widely used by South Korean governments and institutions, it becomes important to develop a robust malware detection model for HWP files.

We use and compare two recently designed CNN models for measuring our method: MalConv [3] and SPAPConv [1]. The MalConv has a shallow and wide architecture using a gate mechanism, and the SPAPConv enhanced embedding representation by employing spatial pyramid average pooling. These models commonly adopt an embedding layer because each byte token is not numerical but a categorical value; the embedding layer generates a distributed representation (i.e., real-numbered vector) from a given byte token. Although these CNN models are known to be able to capture arbitrary features automatically without feature engineering, it is still important to study data characteristics for better model design and parameter engineering. The byte streams of the PDF and HWP files have different lengths on average. As reported in [22], the PDF byte streams are normally 1,000 bytes long, whereas the HWP byte streams often have millions of bytes. Following the previous studies [1], [22], we set the input length I^{pdf} of the PDF-specific detection model M^{pdf} to be 1,000, and the input length of the HWP-specific detection model I^{hwp} is 100,000. To manage the huge gap of input length between the two models, we compare two different padding strategies: post padding and stretch padding [1]. The post padding is widely used in many previous studies, whereas the stretch padding is to spread the element values of the stream evenly and make it a longer stream. Specifically, the PDF streams are much shorter than the HWP streams, so just padding the back of the PDF stream may degrade the performance of CNN model because many convolutional operations run on byte sequences made of only padding tokens. On the other hand, the stretch padding has a potential to alleviate this problem as it allows the convolutional operations run on the evenly spread byte tokens; more details of the stretch padding can be found in [1].

B. BYTE STREAMS OF HWP AND PDF FORMATS

To better understand and find motivation of utilizing the byte streams of HWP and PDF files, we investigated the byte streams and studied some cases related to our method. One or more byte streams exist in a single file, as depicted in Fig. 2, and malicious actions can appear in any of these streams. If a byte stream has at least one malicious action, then the stream is called malicious or malware stream, whereas the benign stream contains no malicious action at all.

The purpose of malware includes stealing industrial secrets from infrastructure, collecting personal information, and making money demands using ransomware. In the past, PE files were mainly used to achieve this purpose, but

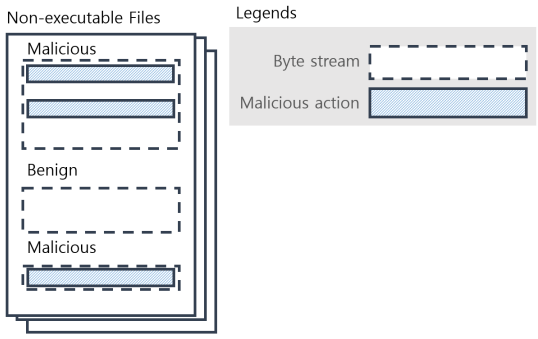

FIGURE 2. Malicious and benign byte streams within files.

recently, attacks using document files are increasing. This attack exploits vulnerabilities and functions provided by programs that read and edit document files. Hangul word processor (HWP) provides JavaScript and Visual Basic For Application (VBA) macro functions, and Encapulated PostScript (EPS) files for high-quality pictures can be inserted into documents. Especially, the JavaScript language is often used as a means to make writing interactive web pages easier, and PDFs uses the JavaScript to format data, calculate data, validate data, or specify actions. For this reason, even if the document format is different, when the program supports the same language, such as JavaScript, it can be helpful to use a part of a specific document-type malware for deep learning training in a different document format. In addition, attackers are always devising new attack methods to circumvent existing detection rules; for example, as one of the ideas, attackers try to apply malicious code of a different file format to create a new malicious code that operates on the target file. In many cases, other vulnerabilities for different document types do not work properly in a specific document type, but the potential for malware execution still exists. For example, CVE-2014-1761 utilizes a vulnerability in Rich Text Format (RTF), which has been included in the PrvText stream in HWP. As another example, CVE-2017-11882 uses a vulnerability in Microsoft Office Word. Since the purpose of the malicious behavior is the same, it may be advantageous to use a document-type malware sample in a different format for deep learning training of the target document.

IV. EXPERIMENT

A. DATA AND ENVIRONMENT

We got 1,856 HWP files and 12,367 PDF files from anti-virus company, and its statistics are summarized in Table 1, where every malware file has at least one malicious byte stream; for example, as shown in Fig. 2, there are two malicious streams and one benign stream in the sample file, so this sample is a malware file as it contains malicious streams. The byte streams are extracted from the files using the algorithm of [22]. The per-stream annotation was performed by using malware detection tools and manual confirmation by human

TABLE 1. Number of data files.

Format	Malware	Benign	Total
HWP	627	1,229	1,856
PDF	9,860	2,507	12,367
Total	10,487	3,736	14,223

TABLE 2. Number of byte streams.

Target	Source	Use	Malware	Benign	Total
PDF	PDF	Train	153,347	8,693	162,040
		Validation	16,752	814	17,566
		Test	19,882	1,168	21,050
	HWP	Train	4,016	1,221	5,237
HWP	PDF	Train	7,462	28,812	36,274
	HWP	Train	3,465	26,178	29,643
		Validation	339	3,044	3,383
		Test	505	3,285	3,790

experts. Table 2 summarizes the statistics of byte streams. Note that, for each target, byte streams of the two formats are used to train the corresponding model. For example, when we train M^{pdf}, the byte streams of PDF and HWP are used for training, where the HWP byte streams are only for training; in this case, the HWP byte streams for M^{pdf} are sampled from the set of all HWP files while we kept its label ratio as similar as possible to the label ratio of PDF files.

We adopted two recent malware detection models for experiments: MalConv and SPAPConv. The SPAPConv was implemented exactly same as in [1], and MalConv was implemented as done in [3] but with 2-dimensional convolution instead of 1-dimensional convolution. This modification is borrowing the idea of SPAPConv, and it exhibited performance improvements (e.g., 4-5% of accuracy). The input length of PDF byte streams was 1K, and the input length of HWP byte streams was 100K. We used a machine having Intel(R) Core(TM) i9-10900X CPU@3.70GHz and two graphics processing unit (GPU) of GeForce RTX 3090. The models were implemented using Python3 language with Tensorflow packages.

B. RESULTS

We independently conducted experiments for different target formats (e.g., PDF, HWP). For each target format, we performed three experiments and averaged per-class precision, recall, and F1 scores. Following the training recipe of SPAPConv [1], we applied batch normalization [13] for the convolutional layer and drop-out technique [14] for the fully-connected layer. For MalConv, we also took the drop-out technique for the fully-connected layer as done in [3]. For both models, we adopted the cost function of cross-entropy and Adam optimizer [23] with initial learning rate 0.001. The label ratio is skewed as shown in Table 2, so we employed the cost-sensitive learning technique. With the validation dataset, we got the proper number of epochs (e.g., 5-10 epochs) by a grid searching.

The per-class performance of HWP malware detection is described in Table 3, where HWP+PDF indicates that

TABLE 3. Malware detection performance on HWP byte streams, where P, R, and F1 represent precision, recall, and F1 score (%), respectively.

Model	Class	HWP+PDF(stretch)			HWP+PDF(post)			HWP		
		P	R	F1	P	R	F1	P	R	F1
MalConv	Malware	99.17	94.65	96.86	97.55	94.46	95.98	96.57	94.65	95.60
	Benign	99.18	99.88	99.53	99.15	99.63	99.39	99.18	99.48	99.13
SPAPConv	Malware	97.33	93.66	95.46	94.37	96.24	95.29	95.64	95.64	95.64
	Benign	99.03	99.60	99.32	99.42	99.12	99.27	99.33	99.33	99.33

TABLE 4. Malware detection performance on PDF byte streams, where P, R, and F1 represent precision, recall, and F1 score (%), respectively.

Model	Class	PDF+HWP			PDF		
		P	R	F1	P	R	F1
MalConv	Malware	99.94	99.86	99.90	99.98	99.78	99.88
	Benign	97.64	98.97	98.30	96.36	99.74	98.02
SPAPConv	Malware	99.96	99.57	99.76	99.96	99.87	99.92
	Benign	93.10	99.32	96.11	97.89	99.40	98.64

the model is trained using HWP and PDF byte streams together, and *stretch* and *post* represent the stretch padding and the conventional post padding, respectively. As the PDF byte streams are relatively shorter than the HWP byte streams in length, the PDF byte streams are mostly padded in either of the two padding ways. Interestingly, with MalConv, training with both byte streams together exhibits performance improvements; especially, precision of malware case was dramatically improved. In terms of the F1 score, using the two byte streams (i.e., HWP+PDF) was superior to using only HWP byte streams. The best F1 score was achieved when we use the stretch padding, and this is consistent with the results of [1]. On the other hand, with SPAPConv, we did not observe any performance improvement. Fig. 3 shows the per-class receiver operating characteristic (ROC) curves of MalConv and SPAPConv, where the padding way is chosen based on their recall values. The SPAPConv exhibits smooth curves, and it might be preferable if we want a detection model with smaller false negative rates.

Table 4 summarizes the per-class performance of PDF malware detection. As the HWP byte streams are much longer than the PDF byte streams, we got samples of 1000-bytes from the HWP byte streams so the PDF+HWP byte streams have the same length. As shown in this table, we observed that the MalConv has shown the performance improvement but it was not that much, whereas the SPAPConv does not exhibit any improvement. This result is similar to the HWP malware detection result of Table 3, and this might be related to two factors as follows. The first factor is the model size. The number of parameters of MalConv and SPAPConv are 1M and 70K, respectively, therefore the SPAPConv might not have enough power to encode the underlying patterns of different file formats. The second factor is the model architecture, especially what representation it takes for convolutional operations. The MalConv feeds the conventional embedding representations to the convolutional operations, whereas the convolutional layer of SPAPConv takes as input the result of 1-level spatial pyramid average pooling (SPAP) layer. The

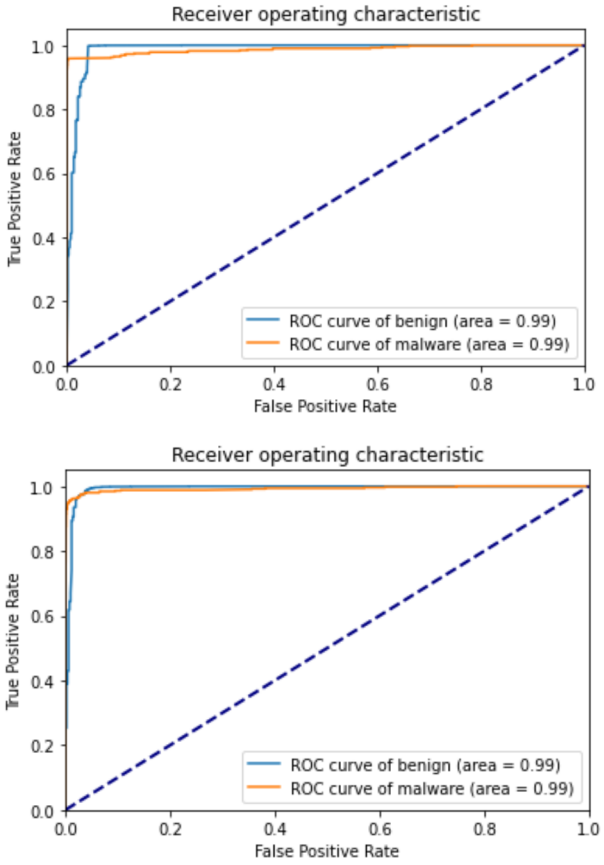

FIGURE 3. Receiver operating characteristic (ROC) curves of (up) MalConv with stretch padding and (down) SPAPConv with post padding, where *area* indicates area under the curve (AUC) score.

SPAP layer extracts a representative value for each region of an arbitrary size that allows it to yield an output of a fixed size; that is, long streams will be roughly looked at whereas short streams are closely looked at. Since the difference in length between the PDF and HWP byte streams is large, the SPAP layer might act like a telescope for two objects with a huge difference in distance, so it eventually confused the entire model.

V. DISCUSSION

By experimental results, we observed that using byte streams of different formats helps to increase performance of malware detection. One may ask how much impact the amount of different formats have (e.g., 50% of PDF byte streams for HWP malware detection). Table 5 shows the results of HWP

TABLE 5. Malware detection performance on HWP byte streams with different amount of PDF byte streams with the stretch padding for training, where P, R, and F1 represent precision, recall, and F1 score (%), respectively.

Model	Class	HWP+PDF(50%)			PDF(100%)		
		P	R	F1	P	R	F1
MalConv	Malware	96.59	95.45	96.02	16.34	60.59	25.74
	Benign	99.30	99.48	99.39	89.62	52.30	66.05

malware detection, where HWP+PDF(50%) indicates that we used HWP training data and only a half of the PDF byte streams with the stretch padding to train the model. The half of PDF byte streams was randomly sampled from the PDF byte streams used for HWP+PDF(stretch) of Table 3. The performance of HWP+PDF(50%) was about halfway between HWP+PDF(stretch) and HWP only, so it can be seen that the performance increased in proportion to the amount of the PDF byte streams. Based on this result, we can say that the reason for small improvements of MalConv in Table 4 is that the amount of HWP byte streams was relatively much smaller than that of the PDF byte streams. We also tried to feed only PDF byte streams to the HWP malware detection model, and its performance is very low as shown as PDF(100%) in the Table 5. We may conclude from this result that the byte streams of different file formats have their own distinct underlying patterns, but using additional format streams allows the detection model to learn more complicated patterns so that the model has a chance to achieve better performance.

We analyzed cases that finally succeeded in malware detection using MalConv model trained with HWP+PDF streams but failed by the model trained with only HWP streams. We found that there are some keywords commonly appeared in HWP and PDF streams. For example, a malicious HWP sample of hash value 'e83f8064e' had a stream containing JavaScript code that includes functions such as SaveToFile, RegWrite, GetSpecialFolder, push, fromCharCode, parseInt, substr, and charAt. These functions have turned out to often appear in PDF streams as well, so they were used as keyword-based features in our previous study [24]. This implies that the model found and exploited promising patterns of PDF streams that can be used for HWP malware detection.

VI. CONCLUSION

We examined the possibility of using byte streams of different file formats for malware detection of non-executables of a specific format. By experimental results, we showed that using byte streams of different formats may contribute to performance improvements. We also analyzed the impact of data size, and found that some Javascript functions commonly appeared in HWP and PDF formats that might be related to the performance improvements by using PDF and HWP streams together. As a future work, we are collecting and annotating more data with malware types, and will investigate how much using different stream formats affects different malware types. We will also expand to other formats such as Word and Powerpoint.

REFERENCES

[1] Y.-S. Jeong, J. Woo, S. Lee, and A. R. Kang, "Malware detection of hangul word processor files using spatial pyramid average pooling," *Sensors*, vol. 20, no. 18, pp. 1–12, 2020.
[2] Y.-S. Jeong, J. Woo, and A. R. Kang, "Malware detection on byte streams of PDF files using convolutional neural networks," *Secur. Commun. Netw.*, vol. 2019, pp. 1–9, Apr. 2019.
[3] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and A. Nicholas, "Malware detection by eating a whole exe," in *Proc. 32nd AAAI Conf. Artif. Intell.*, New Orleans, LA, USA, 2018, pp. 268–276.
[4] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[5] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[6] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 785–794.
[7] C. D. Morales-Molina, D. Santamaria-Guerrero, G. Sanchez-Perez, H. Perez-Meana, and A. Hernandez-Suarez, "Methodology for malware classification using a random forest classifier," in *Proc. IEEE Int. Autumn Meeting Power, Electron. Comput. (ROPEC)*, Ixtapa, Mexico, Nov. 2018, pp. 1–6.
[8] S. Ranveer and S. Hiray, "SVM based effective malware detection system," *Int. J. Comput. Sci. Inf. Technol.*, vol. 6, no. 4, pp. 3361–3365, 2015.
[9] R. Kumar and S. Geetha, "Malware classification using XGboost-gradient boosted decision tree," *Adv. Sci., Technol. Eng. Syst. J.*, vol. 5, no. 5, pp. 536–549, 2020.
[10] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv:1804.04637*.
[11] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and S. Chen, "Mutual information and feature importance gradient boosting: Automatic byte n-gram feature reranking for Android malware detection," *Softw., Pract. Exper.*, vol. 51, no. 7, pp. 1518–1539, Apr. 2021.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–788.
[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, Lille, France, 2015, pp. 448–456.
[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
[15] L. Medsker and C. L. Jain, *Recurrent Neural Networks: Design and Applications*. Boca Raton, FL, USA: CRC Press, 1999.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[17] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Lisbon, Portugal, 2015, pp. 1412–1421.
[18] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," 2019, *arXiv:1901.00596*.
[19] W. Huang and W. J. Stokes, "MtNet: A multi-task neural network for dynamic malware classification," in *Proc. 13th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, San Sebastian, Spain, 2016, pp. 399–418.
[20] X. Wang and S. M. Yiu, "A multi-task learning model for malware classification with useful file access pattern from API call sequence," 2016, *arXiv:1610.05945*.
[21] D. Johnson. *PDF's Popularity Online*. Apr. 1. [Online]. Available: https://www.pdfa.org/pdfs-popularity-online/
[22] Y.-S. Jeong, J. Woo, and A. R. Kang, "Malware detection on byte streams of hangul word processor files," *Appl. Sci.*, vol. 9, no. 23, pp. 1–13, 2019.

[23] P. D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015, pp. 1–15.
[24] A. R. Kang, Y.-S. Jeong, S. L. Kim, and J. Woo, "Malicious PDF detection model against adversarial attack built from benign pdf containing Javascript," *Appl. Sci.*, vol. 9, no. 22, pp. 1–17, 2019.

YOUNG-SEOB JEONG received the M.S. and Ph.D. degrees in computer science from KAIST, Daejeon, South Korea, in 2012 and 2016, respectively.

He is a Faculty Member of the Department of Computer Engineering, Chungbuk National University, Cheongju-si, South Korea. His current research interests include malware detection using deep learning techniques, language models for low-resourced languages, healthcare systems for patients, and pharmaceutics.

SANG-MIN LEE received the B.S. and M.S. degrees in electronics engineering from Kyungpook National University, Daegu, South Korea, in 1994 and 1996, respectively.

He is a Principal Researcher with the Electronics Telecommunications Research Institute, Daejeon, South Korea. His research interests include SDN/NFV, cloud security, AI-based malware detection, and 5G/6G security.

JONG-HYUN KIM received the Ph.D. degree in computer science from The University of Oklahoma, USA, in 2005. He is currently a Principal Researcher with the Electronics Telecommunications Research Institute, Daejeon, South Korea. He is also involved in standardization activities as the Vice Chair of WP1 and a Rapporteur of Q.4 (cybersecurity) with ITU SG17. His research interests include information security, cyber security, cloud security, AI-based malware detection, and 5G/6G security.

JIYOUNG WOO received the B.S., M.S., and Ph.D. degrees in industrial engineering from KAIST, in 2000, 2002, and 2006, respectively. From 2008 to 2010, she was a Researcher with the AI Laboratory, The University of Arizona, USA. She was a Research Professor with the Graduate School of Information Security, Korea University, from 2010 to 2016. She is a currently a Professor with the Big Data Engineering Department, Soonchunhyang University. Her research interests include game log and medical data analytics.

AH REUM KANG received the M.S. and Ph.D. degrees in information security from Korea University, South Korea, in 2012 and 2016, respectively. She is a Faculty Member of the Department of Information Security, Pai Chai University, Daejeon, South Korea. Her current research interests include security, artificial intelligence, malware, medical data analysis, and online game security.