Detecting Malware based on Analyzing Abnormal behaviors of PE File

Lai Van Duong[1], Cho Do Xuan[2]
Information Assurance Department
FPT University, Hanoi
Vietnam

***Abstract*—Attack by spreading malware is a dangerous attack form that is very difficult to detect and prevent. Attack techniques that spread malware through users and then escalate privileges in the system are increasingly used by attackers. The three main methods and techniques for tracking and detecting malware that is being currently studied and applied include signature-based, behavior-based, and hybrid techniques. In particular, the behavior-based technique with the support of machine learning algorithms has given high efficiency. On the other hand, in reality, attackers often find various ways and techniques to hide behaviors of the malware based on the Portable Executable File Format (PE File) of the malware. This makes it difficult for surveillance systems to detect malware. From the above reasons, in this paper, we propose a malware detection method based on the PE File analysis technique using machine learning and deep learning algorithms. Our main contribution in this paper is proposing some features that represent abnormal behaviors of malware based on PE File and the efficiency of some machine learning algorithms in the classification process.**

***Keywords*—Malware; portable executable file format; detection malware; abnormal behaviors; machine learning; deep learning**

I. INTRODUCTION

Malware is software that is purposefully designed to cause damage to a personal computer, server, or computer network system [1, 2]. The purpose of malware is to execute illegal acts such as unauthorized access, stealing user information, spreading spam email, and even performing blackmail, attack and damage to computer system, etc. for personal gain, economic gain, political or simply they can be created as just some malicious joke. The study [3, 4] listed some common types of malware including Virus, Worm, Trojan Horse, Malicious Mobile Code, Tracking Cookie, Attacker Tool, Phishing, Virus Hoax. According to the statistics [5], the malware distribution situation in 2020 increased by 75% compared to 2019. This is completely reasonable because hackers used to focus on attacking information systems but today they usually primarily chose to attack the user. Therefore, malware increases rapidly not only in the number of attacks but also in their danger level. Studies [6, 7, 8] listed a number of approaches to malware detection including signature-based detection and behavior-based detection. The signature-based detection method is the static analysis which analyzes the source code without executing the file [9]. Some techniques used in the static analysis include:

- Checking file format: the metadata of files can provide useful information. For example, Windows PE files can provide information such as execution time, import and export functions.
- String extraction: involves checking the output of the software (status message or error message) and inferring about the behavior of the malware.
- Trace: Before performing analysis, it is necessary to calculate the hash value of the file in order to verify whether the file has been modified or not. Commonly used hashing algorithms are Message-Digest algorithm 5, Secure Hash Algorithm 256-bit. Also can search for information in source code such as username, file name, registry string.
- Scan with anti-virus software: if the file being analyzed is a known malware, most anti-virus software will be able to detect it. This is often used to verify the results of the analysis.
- Disassembly: involves reversing the machine code into assembly language and thus knowing the logic and the purpose of the software. This is the most commonly used and reliable method in static analysis.

This detection method is only suitable for common types of malware with permanent signatures stored in the database. Modern malware usually attacks and exists for a short period of time.

The behavior-based detection method is based on dynamic analysis. This method will evaluate an object based on its behavior. When an object attempts to perform abnormal or unauthorized behavior, it denotes that the object is malicious or suspicious. A number of behaviors are considered dangerous such as disabling security controls, installing rootkits, autostart, modifying host files, establishing suspicious connections, etc. Each behavior may not be dangerous but when combined together can increase the suspicions of the subject. There is a predefined threshold. If any files exceed this threshold, it will be warned as malware [10, 11, 12, 13]. This method is used to detect malware that has capable of changing signature (polymorphism) or new types of malware (zero-day). However, some types of malware have the ability to detect the virtual environment, it will not execute malicious behavior in the sandbox environment [13]. Moreover, in fact, with the increasing amount of malware, this method is not really effective against new types of malware.

To fix the above disadvantages, in this paper, we propose a malware detection method based on the PE file analysis technique using machine learning and deep learning algorithms.

In particular, in this paper, we will analyze and extract abnormal behaviors in PE files to seek signs of malware and then use machine learning and deep learning algorithms to analyze and conclude about the existence of malware. The difference between our proposed approach and other traditional studies is we do not seek to extract malware behavior based on data that is collected in a virtualized environment. Instead, we analyze each different component in the PE file in detail in order to build behavior profiles of malware. With this approach, we could instantly collect behaviors and functions of malware designed and installed before by attackers.

Details of abnormal behaviors are defined in Section 3A of the paper. The classification algorithms selected for use are presented in Section 3C.

II. Related Works

Dragos Gavrilut [10] proposed a malware detection system based on the improved Perceptron algorithm. With different algorithms, accuracy fluctuates in the range of 69.90% to 96.18%. However, the algorithm with the highest accuracy also has the most false positive results. The most balanced algorithm has a low false positives and accuracy as 93.01%.

Singhal and Raul discussed a detection method based on an improved Random Forest (RF) algorithm combined with Information Gain for presenting more optimal feature [11]. The dataset used by the author includes only the executable file so the feature selection is simpler. The detection rate is 97% and the false positives rate is 0.03%.

Baldangombo et al introduced a feature selection method based on the PE header, DLL libraries, and Application Programming Interface (API) functions [12]. Algorithms used include Naïve Bayes, Decision Tree J48, and Support Vector Machines (SVM). The algorithm with the best results is J48 with an accuracy rate of up to 99%.

Alazab [13] proposed a method to use the API to represent malware features. The SVM algorithm gave the best results with the accuracy as 97.6% and the rate of false positives as 0.025%.

The results given by the above studies are not the same, because there has not been a unified method for feature detection and representation. The accuracy of each case also depends on the types of malware used to sample and the actual running process.

III. Malware Detection Method based on PE File Analysis

A. Proposed Model

From Fig. 1, in order to detect malware based on analyzing abnormal behaviors of PE files, we will conduct 2 main tasks:

- Extracting behaviors of PE files. In this process, the system finds ways to analyze the PE files to calculate and extract the values of behaviors in PE files. To accomplish this goal, we will pre-define the behaviors that need to be assessed as the basis for the system to check and extract. Details of these behaviors are presented in section 3.2 of the paper.
- Evaluating behavior profiles of PE files. This process evaluates and concludes about malware behaviors based on the behavior profiles of PE files that have been collected. To accomplish this goal, we propose to use machine learning and deep learning algorithms.

B. Selecting and Extracting Features

PE File [14] is a Win32-specific file format. All executable files on Win32 such as *.EXE, *.DLL (Dynamic Link Library () (32 bits), *.COM, *.NET, *.CPL, etc. are PE format, except for VxDs and *.DLL (16 bits) files. Even NT's kernel mode driver uses the PE file format. PE file is divided into two sections: Header and Section. In which, the Header is used to store file format values including information required for the process of loading files to memory. This structure consists of 3 parts defined in windows.inc: Signature is one DWORD starting in PE Header and containing PE signatures: 50h, 45h, 00h, 00; FILE_HEADER includes the next 20 bytes of the PE Header, this section contains information about the physical layout diagram and file features; OPTIONAL_HEADER include the next 224 bytes after FILE_HEADER. The Section Table is a component after the PE Header. It includes an array of IMAGE_SECTION_HEADER structures, each element contains information about a section in the PE file. In which, there are some important fields: VirtualSize is the actual size of the data on the section in bytes, this value may be smaller than the size on the disk (SizeOfRawData); VirtualAddress is the RVA of the section which is the value to map when the section is loaded into memory; SizeOfRawData is the size of the data section on the disk; PointerToRawData is the offset from the beginning of the file to the data section; Characteristic is the section properties including execution or data initialization.

From the brief overview of the PE file format, we can see that the PE header is quite complex with many variables and fields. Malware designers often use the PE header to conceal the malware version from the malware detection software. In this paper, we will examine and extract some features that represent malware behaviors in the PE header using the LEIF library. Table I below lists malware behaviors that are extracted based on different components in the PE header.

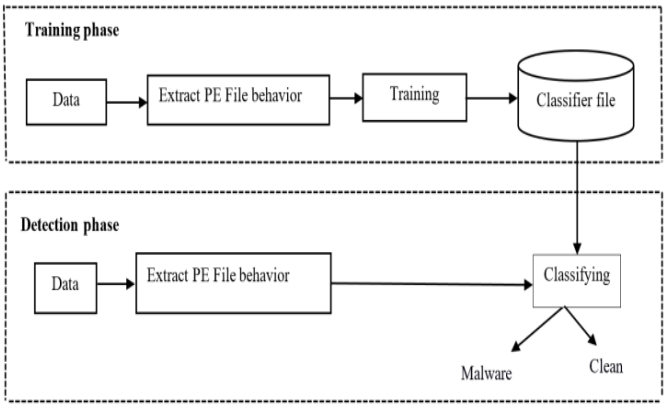

Fig. 1. Malware Detection Model based on Analyzing Abnormal behaviors in PE Files.

TABLE I. LIST OF MALWARE BEHAVIOR FEATURES IN PE HEADER

No.	Group	Name	Description	Data type
1	Properties*	pe.has_configuration	Contains the address and size of the load configuration	Integer
2		pe.has_debug	The address and size of the debug start point	Integer
3		pe.has_exceptions	Exception handling functions	Integer
4		pe.has_exports	Export special characters	Integer
5		pe.has_imports	Import special characters	Integer
6		pe.has_nx	The area of memory to use by storing processor instructions	Integer
7		pe.has_relocations	The address and size of the base relocation table	Integer
8		pe.has_resources	Indexed resources	Integer
9		pe.has_rich_header	Structure after MZ DOS header	Integer
10		pe.has_signature	Digital signatures	Integer
11		pe.has_tls	A special storage layer that Windows supports	Integer
12 . . . 75	PE entry point *	First 64 bits of Entry point	This function is in IMAGE_OPTIONAL_HEADER and contains the address of the base image	Real
76 . . . 331	ASCII	256 characters in ASCII code table	Character set and character encoding based on the Latin alphabet	Real
332 . . . 481	Liblabries	150 most commonly used libraries (Group B)	Dynamic Link Libraries	Real
482		Pe.virtual_size	Ratio of the size of the PE file on the disk and on the RAM	Real
483	PE Section	CNT_CODE	Total SECTION_CHARACTERISTICS.CNT_CODE divided by the total sections	Real
484		MEM_EXECUTE	Total SECTION_CHARACTERISTICS.MEM_EXECUTE divided by the total sections	Real
485		Entropy	Total entropy in sections	Real
486		Virtual_size	The ratio of the actual size of each section to the size on disk and total sections	Real

C. Malware Classification Algorithm

In this paper, we will use a number of deep learning and machine learning algorithms to classify files into normal or malicious. Accordingly, we choose to use the RF and SVM algorithms. Regarding deep learning algorithms, we use 3 main algorithms: Multi Layers Perceptron (MLP), Convolutional Neural Network (CNN), Long Short Term Memory (LSTM). The documents [15, 16, 17, 18] described in detail the mathematical basis and operating principle of these algorithms. Regarding MLP, CNN, LSTM algorithms, the documents [19, 20, 21, 22] presented about how they work and their applicability. In this paper, we will proceed to apply algorithms in the task of detecting malware. Based on the experimental results, we will have a basis to evaluate the effectiveness of each algorithm in the task of detecting malware.

IV. EXPERIMENTS AND EVALUATION

A. Experimental Dataset

In this paper, we use the datasets about malware and normal files provided at [19]. Specifically, the dataset includes 49,128 records consisting of 24,528 malware files and 24,602 normal files. The malware and normal files are selected and extracted into the fields and components listed in Table I.

B. Experimental Scenarios

1) For the experimental dataset: Based on the experimental dataset that was collected and described as in Section 4A, we will mix and randomly divide in which 80% of the number of records in the dataset will be used in the training process and the remaining 20% of the data set will be used in the test process.

2) For the classification algorithm: We will use five different algorithms to conduct experiments on the dataset presented above. To evaluate the effectiveness of each algorithm, we will conduct experiments on each algorithm with the change in their parameters. Our purpose is to evaluate and find the most efficient algorithm as well as the most optimal parameters in that algorithm. Specifically, we proceed to refine the parameters of the algorithm as follows:

- For the Random Forest algorithm, we will conduct experiments and evaluate algorithms based on the change number of decision trees respectively as 20, 30, 50, 70, 100.
- For the SVM algorithm, we select the Kernel parameter as RBF, linear, sigmoid, polynomial.
- For the MLP algorithm, we will change Activation function = ("identity", "relu", "logistic", "tanh") and Solver = ("lbfgs", "adam").
- For the CNN algorithm, firstly, we convert the entire dataset to images with a certain size. With CNN model, we have the following model: Input image -> Convolution2D -> Pooling Layer -> Fully Connected layer -> Output. The input is 49128 images with a size of 27 * 18 (3 dimensions). Then we use alternately 3 Convolution2D layers to extract the features of the image, and 3 Pooling Layers (MaxPooling) to reduce the size of the input and still retain image characteristics. After going through many Convolution and Pooling Layers, the model has also learned the characteristics of the image and the Fully Connected Layer will combine the features of the image into the output of the model.
- For the LSTM algorithm, we will change Activation function = ("tanh", "relu", "softsign", "selu").

C. Methods of Evaluating a System

- Accuracy: is the ratio between the number of correctly predicted points and the total number of points in the test dataset. It is calculated by the following formula:

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

- Recall: is the ratio of the number of true positive points among actually positive points. It is calculated by the formula:

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

- Precision: is the ratio of the number of true positive points among those classified as positive. It is calculated by the formula:

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

- F1 Score: is harmonic mean of precision and recall (assuming that these two quantities are nonzero).

Where:

- True Positive (TP): Both the actual and predicted values are positive.
- True Negative (TN): Both the actual and predicted values are negative.
- False Positive (FP): The actual value is negative but the prediction is positive.
- False Negative (FN): The actual value is positive but the prediction is negative.

D. Experimental Results

1) Experimental results with random forest: From the experimental results in Table II, we found that the accuracy of the Random Forest algorithm increases gradually when the number of decision trees increases. The algorithm gives the best classification results with all metrics when the number of decision trees is 100. The best results in classification are Accuracy, Precision, Recall, F1-score as 97.62; 99.10; 96,123; 97.59 at the number of decision trees as 100. Besides, the result classification of the algorithm for normal files is relatively high from 98.82% to 99.10% while the result classification for malware reaches only from 95.19% to 96.123%. This result is relatively good because the experimental dataset is balanced in the number of malware and normal files. Fig. 2 shows the results when testing the malware detection model using the Random Forest algorithm with the number of decision trees as 100.

From Fig. 2, can see that the algorithm incorrectly predicted 211 malwares and 42 normal files. This result is acceptable when the dataset has a large number of malwares and normal files.

2) Experimental results with SVM: Table III shows the results of malware detection using SVM algorithm.

The experimental results in Table III show that with the 486 features of PE file and using the SVM algorithm, we obtained the results with accuracy as 95.77%. Obviously, the default kernel of the algorithm as RBF (C = 100.0) gave the highest accuracy compared to the remaining kernels. For the Sigmoid kernel, the result is quite low (only approximately 50%). With this result, the SVM algorithm is not really suitable for this PF file-based malware detection dataset. Fig. 3 below shows the evaluation results of the process of testing the model with the SVM algorithm with parameter as RBF (C = 100.0).

TABLE II. EXPERIMENTAL RESULTS WITH RANDOM FOREST ALGORITHM

N_estimator	Accuracy	Precision	Recall	F1_score
20	97.02	98.82	95.19	96.97
30	97.07	98.78	95.33	97.02
50	97.16	98.95	95.35	97.12
70	97.25	98.89	95.59	97.21
100	**97.62**	**99.10**	**96.123**	**97.59**

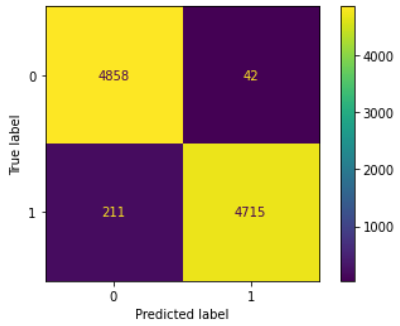

Fig. 2. Confusion Matrix when using Random Forest.

TABLE III. EXPERIMENTAL RESULTS OF DETECTING MALWARE USING SVM ALGORITHM

Kernel	C	Accuracy	F1_Score	Recall	Precision
RBF	1	93.40	93.40	93.40	93.46
	10	95.47	95.47	95.48	95.51
	100	95.77	95.78	95.78	95.78
Linear	1	87.06	87.06	87.07	87.14
Polynomial	1	92.45	92.45	92.45	92.45
	10	95.13	95.12	95.13	95.15
Sigmoid	1	49.56	49.56	49.56	49.56
	10	49.31	49.31	49.31	49.31

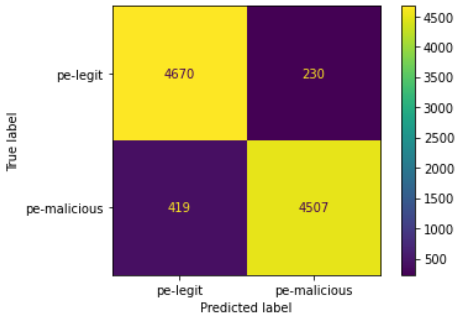

Fig. 3. Confusion Matrix when using SVM with kernel RBF/C=100.0.

From Fig. 3, we can see that the results with the test dataset are as follows: the algorithm correctly predicted 4,507 malware, incorrectly predicted 230 normal files into malware and predicted missing 419 malware.

Based on the experimental results in Table III, we noticed that the Random Forest algorithm is more efficient than the SVM algorithm.

3) Experimental results with MLP: Table IV shows experimental results of detecting malware using the MLP algorithm in some cases with custom activation and solver.

From the experimental results in Table IV, we noticed that the more layers and complex the architecture, the higher the classification result of MLP model is. However, the case given the best classification result of MLP model had the number of Hidden Layer as 256, activation as "tanh" and solver as "adam". With this result, the MLP model has improved the efficiency in the malware classification process compared to the SVM and Random Forest algorithms. However, this model is not as efficient as the Random Forest algorithm in normal file classification.

TABLE IV. LIST OF MALWARE BEHAVIOR FEATURES IN PE HEADER EXPERIMENTAL RESULTS OF DETECTING MALWARE USING MLP ALGORITHM

Activa-tion	Layer	Accuracy	F1 Score	Recall	Precision
relu	256	97.13	97.13	97.52	96.73
	128-256	97.03	97.03	97.04	96.98
	128-128-256	96.97	96.97	97.71	96.24
	128-256-512-512	97.09	97.08	97.7	96.46
tanh	256	97.16	97.14	97.99	96.11
	128-256	96.79	96.78	97.33	96.24
	128-128-256	96.99	96.98	97,75	96.22
	128-256-512-512	97.05	97.03	97.77	96.31
logistic	256	96.96	96.95	97.41	96.44
	128-256	96.41	96.42	96.65	96.18
	128-128-256	96.4	96.4	96.43	96.39
	128-256-512-512	96.58	96.59	96.57	96.61
identify	256	89.11	88.94	90.57	87.37
	128-256	89.23	89.07	90.65	87.55
	128-128-256	89.07	88.44	90.23	87.7
	128-256-512-512	89.61	89.51	90.62	88.43

4) Experimental results with LSTM: Table V shows some experimental results of detecting malware using LSTM model with different activation functions including "tanh", "relu", "logistic", and "identity". Corresponding to the activation functions, we have the different number of hidden layers.

From Table V, it can be seen that when using the model trained with the activation function as "relu" (default) and the number of hidden layers as 1 (1024), we had the best results with accuracy as 98.73%, precision as 98.81%, recall as 99.55% and f1 score as 99.18%. These results are quite high. However, with the malware detection problem, if precision is 98.81%, with 49128 files (dataset used in the experiments), the model will detect incorrectly 584 files. Leaking 584 files is considered quite bad because there may be malware files in these files which leads to affecting the work as well as personal data of users and businesses. Considering recall, with recall as 99.55%, the rate of mistakenly detecting malware files to normal files is at an acceptable level (0.45%). Assuming have 1000 malware files, the model can only detect 995 files, the remaining 5 files are classified as normal files. When the number of files need to be detected increases, the rate is pretty bad. As is well known, f1 score is the harmonic mean of recall and precision. However, the loss ratio of the f1 score is still approximately 0.82%. This is acceptable in terms of training

ratio but it would be bad when the data need to be detected is very large. Overall, hidden layers with the "relu" activation function give better results (accuracy, f1 score, recall, and precision) than ones with the other activation function. Therefore, for the problem of detecting malware using the LSTM algorithm, to optimize it, we will use the "relu" activation function and the corresponding hidden layers.

TABLE V. EXPERIMENTAL RESULTS OF DETECTING MALWARE USING LSTM

Activation	Layer	Accuracy	F1 Score	Recall	Precision
tanh	1024	98.17	98.53	98.38	98.68
	32-32-32-32	97.39	97.39	97.41	97.38
	32-64-64-128	97.46	97.46	97.47	97.46
	128-128-256-512	97.76	97.76	97.78	97.76
	128-512-512-512	97.56	97.57	97.61	97.57
relu	1024	98.73	99.18	98.81	99.55
	32-32-32-32	97.7	97.71	97.03	97
	32-64-64-128	96.94	96.93	96.97	96.93
	128-128-256-512	97.79	97.79	97.8	97.89
	128-512-512-512	97.89	97.88	97.9	97.89
softsign	1024	98.37	98.74	98.72	98.77
	32-32-32-32	97.49	97.48	97.5	97.48
	32-64-64-128	97.3	97.29	97.36	97.29
	128-128-256-512	97.91	97.91	97.93	97.9
	128-512-512-512	97.96	97.57	97.96	97.96
selu	1024	98.23	98.63	98.39	98.86
	32-32-32-32	97.19	97.17	97.22	97.17
	32-64-64-128	97.36	97.59	97.31	97.88
	128-128-256-512	97.52	97.77	97.28	98.28
	128-512-512-512	97.69	97.69	97.7	97.69

5) Experimental results with CNN: Table VI shows some experimental results of detecting malware using CNN model with different activation functions including "Image", "No image", "1D".

We noticed that when the input data is converted to images, we had the best results and the difference between the layers is very small (approximately 0.0001 - the results are rounded). The accuracy and f1 score are very good (99.97%). We think this is a very good classification model. With approximately 4% lower, 1D gave the second best results. With 1D, the best model is the model had hidden layer (64-128-256). This model is better because its f1 score is higher than the other two models. As shown in the previous sections, the f1 score helps to choose the best model since it is the harmonic mean of recall and precision. To test the accuracy of CNN after training, we put in a test set including 30,000 images consisting of malware and normal files, the results are similar to the trained model. The algorithm detects completely correct input data. Of course, when the data set is larger, there will be errors in detection. Fig. 4 below shows the evaluation results of the process of testing the model with the CNN algorithm.

Based on the confusion matrix, it can be seen that the model detected very well with the test dataset because there is no file that the model detected incorrectly. The following is a graph that shows the accuracy, loss, f1 score, recall, precision of train and test data during 20 epochs. It can be seen that the train and test ratio increased sharply in the 3rd epoch and stayed the same until the end. Fig. 5 describes in detail the results of this experiment.

E. General Evaluation

After conduct experiments with 5 different algorithms that are SVM, Random Forest, CNN, MLP, LSTM, we have the best results of each algorithm.

Comment: Based on the comparison table (Table VII) of algorithms when analyzing the same file data, we can see that CNN gave the results with the highest accuracy of 99.99%. The algorithm detects completely correct input data.

TABLE VI. EXPERIMENTAL RESULTS OF DETECTING MALWARE USING CNN

Train method	Model train	Accuracy	F1 Score	Recall	Precision
Image	256	99.97	99.97	99.94	1
	16-32-32	99.97	99.97	99.94	1
	32-32-64-64	99.97	99.97	99.94	1
	64-64-128-128-256	99.97	99.97	99.94	1
No image	32	91.4	91.42	91.4	91.4
	32-64	93.81	93.83	93.81	93.81
	64-128	93.86	93.88	93.86	93.86
1D	32-64	94.08	94.06	94.08	94.08
	32-64-128	95.41	95.42	95.41	95.41
	64-128-256	95.29	95.64	93.28	98.11

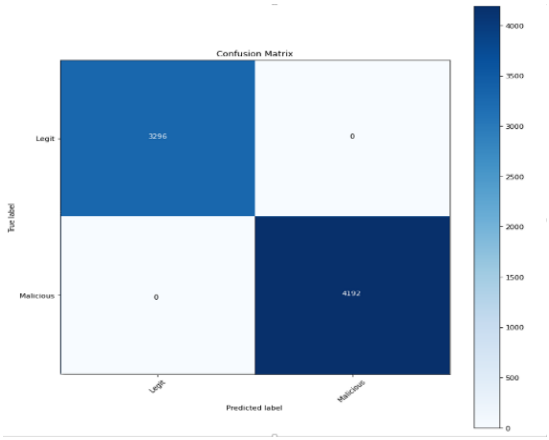

Fig. 4. Confusion Matrix when using CNN.

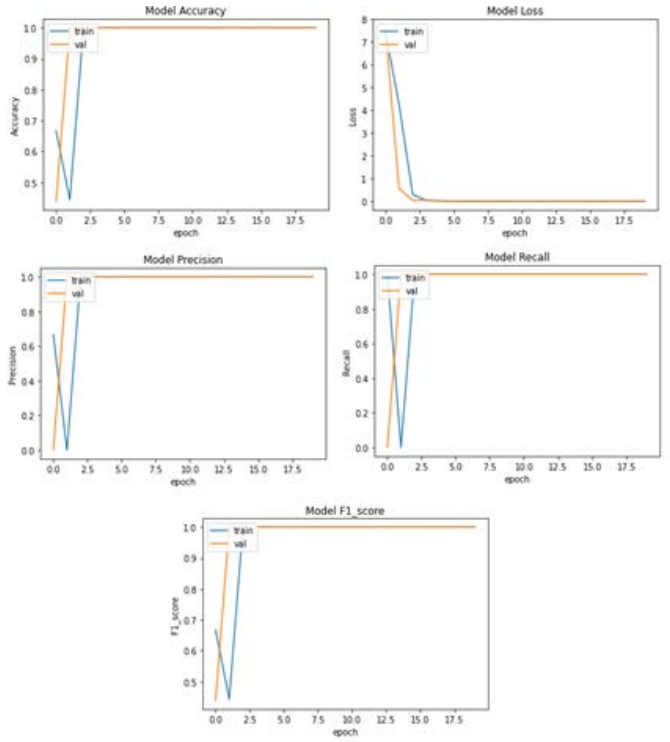

Fig. 5. Accuracy and Loss after 20 Epoch.

TABLE VII. COMPARING ALGORITHMS WHEN ANALYZING THE SAME PE FILE DATA

Algorithm	Accuracy	F1 Score	Recall	Precision
CNN	**99.97**	**99.97**	**99.94**	**1**
LSTM (activation ="relu", layer=256)	98.46	98.94	98,72	99.15
RF(N_estimator = 100)	97.62	99.1	97.59	96.12
MLP (Layer=256, activation ="tanh", solver ="adam")	97.16	97.14	97.99	96.11
MVC (kernel = RBF, C=100)	95.78	95.78	95.78	95.78

V. CONCLUSION

In this paper, based on the PE File analysis technique, we proposed some features that represent abnormal behaviors of malware. The experimental results in section 4.3 have demonstrated that the features that are extracted from the PE File and selected and proposed by us gave good results, it correctly classified not only for normal files but also for malware. Besides, based on the experimental results of algorithms with different parameters, we have proven that the CNN algorithm gave better efficiency than the remaining algorithms in all aspects. Especially, in this dataset with a relatively high number of features (485 features), the CNN algorithm brought the best results. In the future, in order to improve the efficiency of the malware detection process based on PE File analysis, we need to improve two main issues: i) extracting additional features of malware based on PE File. We found that the PE File consists of many different components and has many important components that are exploited by malware developers to conceal information about malware behavior. Therefore, analyzing detailed and generalizing these features will significantly improve the efficiency of the malware detection process in the context of increasing malware in both quantity and form of distribution; ii) use other advanced machine learning algorithms. Obviously, classical machine learning algorithms have brought good efficiency to the classification process. However, due to the real situation about the rapid increase in the number of malware behaviors as well as the amount of experimental data, other advanced classification algorithms are required to ensure the effectiveness of the detection and monitoring process.

REFERENCES

[1] Daniel Gibert, Carles Mateu, Jordi Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," Journal of Network and Computer Applications, vol. 153, pp. 1-22, 2020.

[2] Ucci, Daniele & Aniello, Leonardo, "Survey on the Usage of Machine Learning Techniques for Malware Analysis," Computers & Security, 2017, 81. 10.1016/j.cose.2018.11.001.

[3] Sanjay Sharma, C. Rama Krishna, Sanjay K. Sahay, "Detection of Advanced Malware by Machine Learning Techniques," 2019, arXiv:1903.02966.

[4] Alireza Souri, Rahil Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," Human-centric Computing and Information Sciences, vol. 8(1), pp. 1-22.

[5] Kaspersky-Lab, "Machine Learning Methods for Malware Detection," 2020.

[6] R. Islam, R. Tian, L. M. Batten, S. Versteeg, "Classification of malware based on integrated static and dynamic features," Journal of Network and Computer Applications, vol. 36 (2), pp. 646–656, 2013.

[7] C.-T. Lin, N.-J. Wang, H. Xiao, C. Eckert, "Feature selection and extraction for malware classification," Journal of Information Science and Engineering, vol. 31 (3), pp. 965–992, 2015.

[8] A. Mohaisen, O. Alrawi, M. Mohaisen, "Amal: High-fidelity, behaviorbased automated malware analysis and classification," Computers & Security, vol. 52, pp. 251–266, 2015.

[9] S. Palahan, D. Babi´c, S. Chaudhuri, D. Kifer, "Extraction of statistically significant malware behaviors," Computer Security Applications Conference, ACM, pp. 69–78, 2013.

[10] Gavrilut, Dragos, Mihai Cimpoesu, Dan Anton, Liviu Ciortuz, "Malware Detection Using Machine Learning," The International Multiconference on Computer Science and Information Technology, 2009.

[11] Priyank Singhal, Nataasha Raul, "Malware Detection Module using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks," 2015.

[12] Baldangombo Usukhbayar, Nyamjav Jambaljav, Shi-Jinn Horng, "A Static Malware Detection System Using Data Mining Methods", Cornell University, 2013.

[13] Alazab, Mamoun, Sitalakshmi Venkatraman, Paul Watters, Moutaz Alazab, "Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures," Proceedings of the 9-th Australasian Data Mining Conference, pp. 171-181, 2011.

[14] Nakajima, Tatsuo & Ishikawa, Hiroo & Kinebuchi, Yuki & Sugaya, Midori & Lei, Sun & Courbot, Alexandre & Zee, Andrej & Aalto, Aleksi & Duk, Kwon, "An Operating System Architecture for Future Information Appliances," pp. 292-303, 2008, 10.1007/978-3-540-87785-1_26.

[15] C. Corinna, V. Vladimir, "Support-vector networks," Machine Learning, vol. 20, pp. 273-297, 1995.

[16] S.S. Shai, B.D. Shai, "Understanding Machine Learning: From Theory to Algorithms," Cambridge University Press, 2014.

[17] JohnShawe-Taylor, ShiliangSun, "Kernel Methods and Support Vector Machines," Academic Press Library in Signal Processing, vol. 1, pp. 857-881, 2014.

[18] LEO BREIMAN, "Random Forests", Machine Learning, vol. 45, Issue 1, pp. 5–32, 2001.

[19] Daniel Svozil, Vladimir Kvasnicka, Jiří Pospíchal, "Introduction to multi-layer feed-forward neural networks," Chemometrics and Intelligent Laboratory Systems, vol. 39(1), pp. 43-62.

[20] Zewen Li, Wenjie Yang, Shouheng Peng, Fan Liu, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," 2020, arXiv, arXiv:2004.02806.

[21] Keiron O'Shea, Ryan Nash, "An Introduction to Convolutional Neural Networks," 2015, arXiv, arXiv:1511.08458.

[22] Sepp Hochreiter, Jürgen Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9(8), pp. 1735 – 1780, 1997.