SCIS 2020 2020 Symposium on
Cryptography and Information Security
Kochi, Japan, Jan. 28 – 31, 2020
The Institute of Electronics,
Information and Communication Engineers

Enhancing Malware Detection by Modified Deep Abstraction and Weighted Feature Selection

Harry Chandra Tanuwidjaja* Kwangjo Kim*

Abstract: The current malware detection method is limited to two kinds of methods, static and dynamic. Static method is easy to use but difficult to detect new kinds of malware. On the other hand, dynamic method is strong against a new malware but needs an expert skill to manipulate it. For the last decades, machine learning has advanced rapidly as a new malware detection method. The goal of this paper is to propose a modified feature learning method for malware detection, which is based on Deep Abstraction and Weighted Feature Selection proposed (DFES) for Intrusion Detection System by Aminanto *et al* in 2017. The methodology consists of a combination between Stacked Autoencoder (SAE) for feature extraction and weight based Artificial Neural Network (ANN) for feature selection and classification. We did some experiment to find the optimum setting for both SAE and ANN and find that the best result is given by the combination of SAE with 2 hidden layers and ANN with 2 hidden layers. In order to correspond with rapid release of new malware variant, we evaluate only the selected features, and remove features that is highly related to static malware detection. We use malware dataset created by Tek that consists of malware files from Virus Share and benign files from windows binaries. Our experimental result shows that we can achieve 95.490% detection rate, 2.007% false alarm rate, and 6 times faster compared to DFES. To the best of our knowledge, this is the first paper that combines SAE and ANN as a feature learning method for malware detection

Keywords: malware detection, feature learning, feature extraction, feature selection, SAE, ANN

1 Introduction

The development of information technology has advanced rapidly. The utilization of Internet is essential to human life. However, there is a big threat that many people still do not realize. The lack of information security awareness exposes user to vulnerability from several cyber-attacks. Information security issues cannot be separated from malware problems. Several researches in recent years have focused on improving malware detection accuracy, as the variety of malware increases from time to time. In 2013, Kaspersky Lab stated that there were 200,000 new malware files that appeared every day [1]. However, most antiviruses still rely on a signature method, which compares their test subject to a black list that contains malware signatures. This leads to a vulnerability to new malware, whose signature is not included in this list. To overcome this problem, this paper proposes a new kind of a malware detection method that can identify malware efficiently using customized machine learning [2].

Previous publications on malware detection have been proposed as follows: Alkhateeb [3] suggested the use of API similarity for detection. Lansheng *et al.* [4] had an idea to do classification on malware based on their task behavior. Yin *et al.* [5] showed how to find malware by analyzing the network traffic. However, all these methods require expert analysis and consume too much time. Our method leverages machine learning as substitution of human experts that requires much shorter time compared to human analysis. The challenges in utilizing machine learning are how to improve detection accuracy and how to reduce the false alarm rate as low as possible, but still require some amount of processing time.

In this paper, we use malware dataset generated by Tek [6], which consists of benign and malware files. Benign files contain Windows binaries from Windows 2008, Windows XP, and Windows 7. There are 56 features in the dataset. The total number of files is more than forty thousands, come from both 32-bit and 64-bit versions of Windows. Malware files were taken from Virus Share. The size of malware files are more than ninety thousand files. Both benign and malware binaries were extracted into a list of feature in CSV format by using pefile [7]. This dataset then needs to be pre-processed in order to train the model during the learning process. Then, we use this pre-processed dataset as the input of our approach, from the idea of *Deep abstraction and weighted FEature Selection* (DFES) [2], which was verified to get the best performance to detect impersonation attack in Wi-Fi network.

In addition, after extending their approach [2] to detect malware, we try to implement modified DFES with less feature during feature selection process. The original DFES consists of four main parts: dataset pre-processing, feature extraction, feature selection, and classification. During feature extraction, important fea-

* School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Gwahak-ro, Yuseong-gu, Daejeon, 34141, Korea. {*elevantista*, *kkj*}@kaist.ac.kr.

tures of malware will be extracted from a dataset and the features were going to be stored as our sample. Then, those new features and original features are concatenated and important features are selected from the concatenated features in feature selection phase. The feature extraction expands new features and feature selection chooses important features from them. During the evaluation, we check the detection accuracy and false alarm rate of our proposed approach. We compare our result with [6], who used tree-based feature selection and random forest for classification and achieved 99.351% detection rate with 0.568% false alarm rate.

This paper is organized as follows: Section 2 reviews several related researches. In Section 3, we provide our proposed approach. In Section 4, we give our experimental results and analysis. Section 5 states our conclusion and future work of this paper.

2 Related Researches

Malware detection has been studied for decades. The old detection technique includes static detection and dynamic detection. Lately, several machine learning based malware detection methods have been proposed.

Some methods use malware behaviors as the features of machine learning for malware detection. Shibahara *et al.* [8] proposed a deep learning approach based on characteristic of malware communication using Recurrent Neural Network (RNN). They showed that the proposed method reduced 67.1% of analysis time while keeping the range of covered URL to 97.9% compared to full analysis method. Kolosnjaji *et al.* [9] proposed deep learning based malware detection using system call sequence. They showed that the combination of Convolutional Neural Network (CNN) and Long-Short Term Memory (LSTM) gives better accuracy, compared to feedforward network and convolutional network. Firdausi *et al.* [10] proposed an automatic behavior based malware detection using machine learning. They used five classifiers including k-Nearest Neighbors (k-NN), Naïve Bayes, J.48 Decision Tree, Support Vector Machine (SVM), and Multilayer Perceptron Neural Network (MLP). Their experiment showed that J.48 classifier gave the best performance. Rieck *et al.* [11] proposed a malware detection scheme based on malware behavior using machine learning. They showed that the incremental technique in malware behavior based analysis successfully decreased run time and memory requirement, compared to regular clustering.

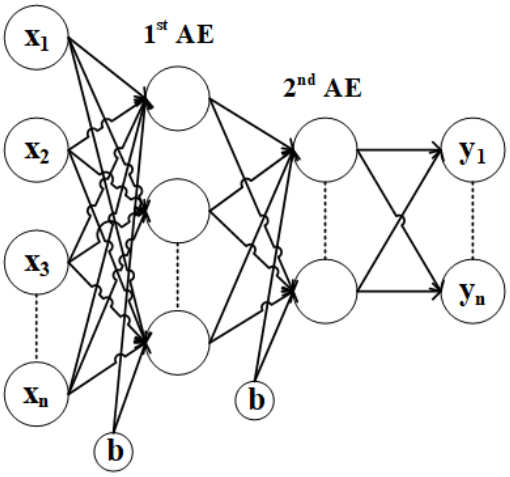
There are methods that combine feature extraction and classification in machine learing for malaware detection. Tobiyama *et al.* [12] proposed a malware detection method using deep neural network based on data traffic on computer. They used RNN for feature extraction and CNN for classification. Their proposed method achieved 92% detection accuracy. David *et al.* [13] proposed Deepsign, a deep learning approach for automatic malware signature generation and classification. They used Deep Belief Network (DBN) to produce malware signatures. Their proposed approach reached 98.6% accuracy with 0.2 input noise and 0.001 learning rate. Xu *et al.* [14] proposed machine learning based malware detection by using virtual memory access patterns. They used three classifiers (SVM, Random Forest, and Logistic Regression) to do training phase. They showed that the best performance was achieved by random forest with 99% true positive rate and 1% false positive rate. Liu *et al.* [15] proposed a combination between image processing and machine learning. They used opcode n-gram with gray scale images to extract malware features. Then, they did clustering process using Shared Nearest Neighbor (SNN) clustering algorithm. They reached 96.5% accuracy by using random forest classifier.

Rathore *et al.* [16] proposed random forest based deep learning with opcode frequency as feature vector for malware detection. Vinayakumar *et al.* [17] combined image processing with deep learning for hybrid zero-day malware detection. While Xiao *et al.* [18] proposed behavior based deep learning framework to detect malware in cloud service environment. The extracted API calls and use it as features during the learning process. Zhong *et al.* [19] proposed multi level deep learning structure that utilizes tree structure to do clustering on malware detection system. Liu *et al.* [20] implemented malware detection system by leveraging deep learning on API calls. Karbab *et al.* [21] extraced API from Android devices as features for deep learning based malware detection on IoT devices.

Other methods combine feature selection and classification in machine learing for malware detection. Raman *et al.* [22] proposed an approach to do feature selection in malware classification, with the addition of using intuitive method during feature selection. They used random forest algorithm to do feature selection. Then, four classifiers, PART, IBk, J48Graft, and J48 were used to choose the highest seven features. Gandotra *et al.* [23] proposed zero-day malware detection by combining static and dynamic malware analysis with machine learning algorithm. They generated their own dataset from Virus Share (for malwares) and Windows system directories (for benign files). They did feature selection by using information gain method, an entropy based technique for selecting features. Then, for classification, they used seven classifiers from Weka, including IB1, Naïve Bayes, J48, Random Forest, Bagging, Decision Table, and Multi-layer Perceptron. The best performance was achieved by Random Forest with 99.97% accuracy.

3 Our Approach

In this paper, we propose a new kind of malware detection method by leveraging machine learning, which offers a fast detection process with the ability to recognize new malwares. The problem in utilizing machine learning is to create accurate samples while we generate features to learn. Like a human, if we teach the machine a wrong thing, it will not be able to detect



htbp

Figure 1: Stacked Autoencoder (SAE) network with two hidden layers

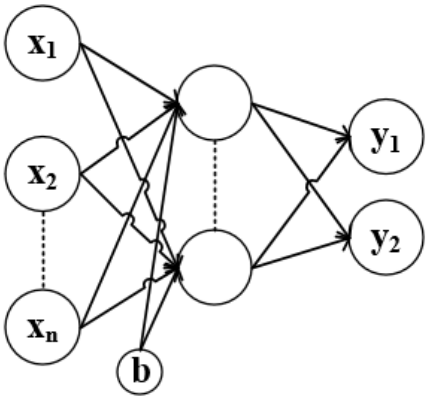


Figure 2: The structure of Artificial Neural Network

malwares well. So, it is very important to produce training samples with important features. We propose a general detection method for all kinds of malwares that leverages machine learning. Generally, DFES can be described into four parts: dataset pre-processing, feature extraction, feature selection, and classification. We use Stacked Autoencoder (SAE) for feature extraction as well as Artificial Neural Network (ANN) for feature selection and classification. We also propose a modified DFES, which based on original DFES, but has lighter feature selection with reduced number of features.

3.1 Dataset Pre-processing

The malware dataset contains various values with different data types. The data varies from negative number, decimal number, big integer, Boolean, and also string. As a result, we cannot use the dataset directly as an input to our machine learner, due to multiple types of data [24]. Because of that background, we decide to do dataset pre-processing. The purpose of this step is to convert the data to real number format with range between zero and one.

3.2 Stacked Autoencoder

Autoencoder [25] is a neural network model for dimensionality reduction in a learning process. When we have high dimensional data with many features, the feature can be spread. As a result, we need many training data during the learning process. Another way to address this issue is by reducing the data dimension. That is the reason why we need an autoencoder. An autoencoder has one hidden layer and same number of input and output [26]. Stacked Autoencoder (SAE), as shown in Fig. 1, is a neural network that consists of multiple encoders [27]. In our approach, we use two autoencoders for feature extraction [28] and train them separately. However, the two autoencoders are not dependent from each other. The second autoencoder uses the hidden layer from the first autoencoder and the number of neurons in each hidden layer will decrease accordingly. During the SAE training, we use unlabeled dataset. The labeled dataset can be used during classification process. The classification process can be done by softmax regression function in the last step of the training process [29].

3.3 Artificial Neural Network

Artificial Neural Network (ANN) [30], as shown in Fig. 2, is a neural network model that consists of many neurons. Each neuron receives input, multiplies the input with weight, and adds bias, resulting in parameters for activation function. The activation function decides whether that neuron should be active or not, based on the weighted sum [31]. In our proposed approach, we use ANN to do feature selection and classification. The ANN can be trained with two target classes: benign class and malware class.

We measure the weight of each feature in dataset to decide which feature is important. The weight here represents the level of influence from input feature to the first hidden layer [32]. If the value is small (nearly zero), it means that the feature is not a deciding factor to pick whether a file is a malware or benign file. We will measure the average weight of all features and set a threshold value. We pick all features that have weight value higher than the threshold. This is how we do the feature selection [33]. We also use ANN for classification process. Our scheme executes minimum global error function with a scale conjugate gradient optimizer [34] in supervised learning environment. We decide to use supervised approach because it will increase the performance of our classifier.

4 Evaluation

We evaluate the proposed scheme on Tek dataset. We implement SAE and ANN algorithm using MATLAB R2016b running on an Intel Xeon E-3-1230v2

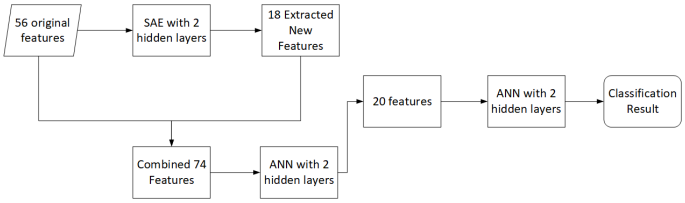


Figure 3a: Case I: Dataset consists of 56 features with original DFES

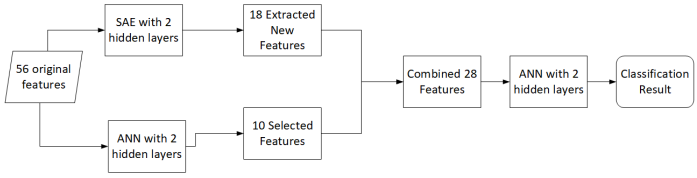


Figure 3b: Case II: Dataset consists of 56 features with modified DFES

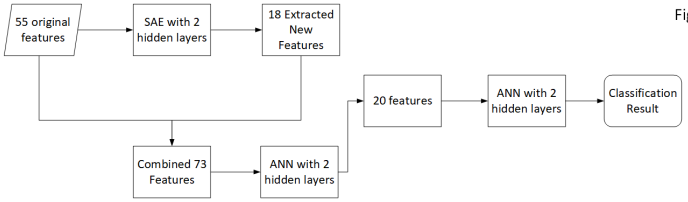


Figure 3c: Case III: Dataset consists of 55 features with original DFES

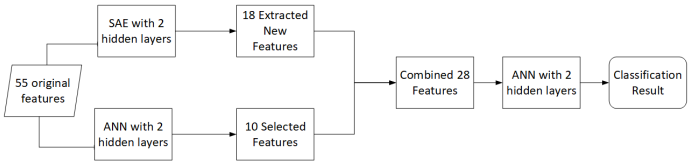


Figure 3d: Case IV: Dataset consists of 55 features with modified DFES

Figure 3: All experiment scenarios used in this paper

CPU @3.70 GHz with 16 GB RAM. We evaluate the performance of our proposed approach in four cases and analyze them.

4.1 Malware Dataset

Tek malware dataset contains two kinds of data, benign and malware files. Benign files are Windows binaries from 32-bit and 64-bit versions of Windows 2008, Windows XP, and Windows 7. Malware files were taken from Virus Share, and then extracted into a CSV file using pefile [7]. The benign files contain 41,323 binaries while malware files contain 96,724 binaries, with a total of 56 features. There are several types of data on the dataset, including integer, decimal number, Boolean, and also string. Since there are various types of data with various value ranges, we need to process the data before giving it as the input of machine learning. The main idea is to convert all data into real number, then do normalization, which means converting the value into a value between zero and one. For string number, the normalization process is different compared to other type of data. We can convert integer, decimal number, and Boolean directly into float type data. However, for string, we cannot process it directly. So, we do iteration and give each string a number label. If the same string appears, it is assigned with the same number value as its predecessor. Normalization process is done by calculating the difference of value with minimum value, divided by the difference of maximum value and minimum value of instances in a feature. Table.1 shows the top 10 features that have biggest weight value in this experiment. Extracted feature means the artificial feature produced during feature extraction process, so that we can not identify what kind of feature it is.

Table 1: Top 10 features with weight value

Feature Name	Weight
MajorLinkerVersion	5.048819093
ImportsNbOrdinal	5.067934552
SectionsNb	5.097262066
SizeOfStackReserve	5.228942805
SizeOfHeapCommit	5.234657548
Extracted Feature 1	5.277306705
Extracted Feature 2	5.489399303
ResourcesMinEntropy	5.496055098
SectionsMaxEntropy	6.191716059
Subsystem	6.209451558

4.2 Experimental Setup

We examine four cases to evaluate our approach by the heuristic way. The four cases are described as follows:

Case I (Fig. 3a): Dataset consists of 56 features with original DFES

In the case I, we use dataset with 56 features. First, we use the dataset as an input to SAE to do feature extraction process. This process give 18 new extracted features as the result. Then, we concatenate the 18 features with the original 56 features, resulting in 74 features. Then, we use the 74 features as the input of ANN during the feature selection process, which gives 20 features as the output. After that, we use the 20

features as the input of ANN for the classification process. The output is classification result. During the testing, we use the model to assess the test dataset, resulting in confusion matrix that contains how many benign files detected as benign, how many benign files detected as malwares, how many malwares detected as benign files, and how many malwares detected as malwares. Later, we will use these parameter to evaluate the performance of our system.

Case II (Fig. 3b): Dataset consists of 56 features with modified DFES

In the case II, we use dataset with 56 features. The main difference from the first case is we do feature selection using 56 original features only. After that, we concatenate the selected features with 18 new extracted features from SAE. By doing this, the number of features for selection process are reduced, compared to original DFES. As a result, the feature selection process becomes faster and lighter than original DFES, which we call it modified DFES. The main purpose of this scheme is that we want to compare the performance of modified DFES with original DFES. We combine 18 new extracted features with 10 selected features, resulting in the combined 28 features. Then, we use the 28 features as the input of ANN during classification process. In the testing phase, we use the model from training phase to evaluate the test dataset.

Case III (Fig. 3c): Dataset consists of 55 features with original DFES

In the case III, the methodology is the same with the first case. However, we reduce the input into 55 original features. We decide to do this after we analyze the result of the first case. During the feature selection of the first case, we check the result of selected features. One of the features called *name*, has a weight of 25. On the other hand, the average weight of all features is 14.74. Because of that reason, we assume that this features highly affect the detection rate. It is reasonable because we use malwares from Virus Share. The malwares are named Trojan, worm, Botnet, etc. So, the *name* feature here will act like a database of static antivirus detector. As a result, we decide to omit the *name* feature from our features in order to make our system resembles a real life that contain various kind of malwares.

Case IV (Fig. 3d): Dataset consists of 55 features with modified DFES

In the case IV, the methodology is the same with the second case, however, we use 55 original features as the input. We leverage modified DFES with fewer features to process during feature selection. We also omit the *name* feature, as in the case III. The purpose of this case is to check the performance of modified DFES in the environment that resembles the real life.

4.3 Evaluation Metrics

In order to measure the performance of our proposed method, we use several evaluation metrics. We use the most well referenced parameter measurements [35], including accuracy, detection rate, false alarm rate, false negative rate, F1 score, and precision. Accuracy (Acc) means the proximity of measured result to the true value. Detection rate (DR) refers to the number of malwares detected, divided by the total number of malwares. False Alarm Rate (FAR) is the number of benign files that is detected as malwares, divided by the total number of benign files in the dataset. False Negative Rate (FNR) is the number of malwares that is wrongly detected as benign, divided by the total number of malwares in the dataset. F1 score is a measurement of harmonic mean between precision and recall. Precision is the number of correctly detected malwares, divided by the number of files that is detected as malwares. The measurement formulas can be defined as shown in Eqs. (1) to (5):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$DR = Recall = \frac{TP}{TP + FN} \tag{2}$$

$$FAR = \frac{FP}{TN + FP} \tag{3}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

where True Positive (TP) is the number of malware files correctly classified as malware, True Negative (TN) is the number of benign files correctly classified as benign, False Positive (FP) is the number of benign files incorrectly classified as malware, and False Negative (FN) is the number of malware files incorrectly classified as benign.

4.4 Experimental Results

We implement our proposed approach based on the four cases that has been stated in Section 4.2. We did some experiment to find the optimum setting for both SAE and ANN and find that the best result is given by the combination of SAE with 2 hidden layers and ANN with 2 hidden layers. We use two hidden layers in SAE network with 35 and 10 hidden neuron for the case I and III, respectively. On the other hand, we use two hidden layers in SAE network with 25 and 10 hidden neuron for case II and case IV, respectively. We present the experiment result of each case with 2 tables. The first table is the experiment result that shows the number of instances for these parameters: TN, FP, TP, and FN. The second table provides the statistical result of the experiment including DR, FAR, F1, and Acc.

Case I: Dataset consists of 56 features with original DFES

The result of case I (Table 2 and Table 3) shows that our approach achieves 99.836% detection rate with

Table 2: The experiment result of case I

Case I	Number of instances			
	TN	FP	TP	FN
Train	67,539	36	29,015	42
Validation	14,590	7	6,095	15
Testing	14,526	4	6,166	11
All	96,655	47	41,276	68

Table 3: The statistical result of case I

Case I	Statistical result (%)			
	DR	FAR	F1	Acc
Train	99.855	0.053	99.866	99.919
Validation	99.755	0.048	99.820	99.894
Testing	99.822	0.028	99.879	99.928
All	99.836	0.049	99.861	99.917

99.917% accuracy. It also has 0.049% FAR and 99.816% F1 score. Our approach gives good performance, however it still contains the *name* feature. We will compare the performance of case I with case III to check the effect of omitting *name* feature from our dataset.

Case II: Dataset consists of 56 features with modified DFES

Table 4: The experiment result of case II

Case II	Number of instances			
	TN	FP	TP	FN
Train	67,627	4	28,980	21
Validation	14,478	3	6,223	3
Testing	14,590	1	6,112	4
All	96,695	8	41,315	28

Table 5: The statistical result of case II

Case II	Statistical result (%)			
	DR	FAR	F1	Acc
Train	99.928	0.006	99.957	99.974
Validation	99.952	0.021	99.952	99.971
Testing	99.935	0.007	99.959	99.976
All	99.932	0.008	99.956	99.974

The result of case II (Table 4 and Table 5) shows that our approach achieves 99.932% detection rate with 99.974% accuracy. It also has 0.008% FAR and 99.956% F1 score. Overall, the modified DFES gives slightly better performance compared to original DFES in case I. We will compare the performance of case II with case IV later.

Case III: Dataset consists of 55 features with original DFES

The result of case III (Table 6 and Table 7) shows that our approach achieves 97.519% detection rate with 98.477% accuracy. It also has 1.114% false alarm rate and 97.455% F1 score. The effect of omitting *name* feature is clearly shown here. The detection rate is reduced by 2.3% with 1.5% less accuracy compared to case I. The total time needed during this experiment is 261 seconds.

Table 6: The experiment result of case III

Case III	Number of instances			
	TN	FP	TP	FN
Train	66,867	758	28,296	711
Validation	14,452	164	5,937	154
Testing	14,380	156	6,012	159
All	95,699	1,078	40,245	1,024

Table 7: The statistical result of case III

Case III	Statistical result (%)			
	DR	FAR	F1	Acc
Train	97.549	1.121	97.470	98.480
Validation	97.472	1.122	97.392	98.464
Testing	97.423	1.073	97.447	98.479
All	97.519	1.114	97.455	98.477

Case IV: Dataset consists of 55 features with modified DFES

Table 8: The experiment result of case IV

Case IV	Number of instances			
	TN	FP	TP	FN
Train	66,277	1,366	27,685	1,304
Validation	14,326	270	5,832	279
Testing	14,260	307	5,863	277
All	94,863	1,943	39,380	1,860

Table 9: The statistical result of case IV

Case IV	Statistical result (%)			
	DR	FAR	F1	Acc
Train	95.502	2.019	95.400	97.237
Validation	95.434	1.850	95.505	97.349
Testing	95.489	2.108	95.256	97.180
All	95.490	2.007	95.394	97.245

The result of case IV (Table 8 and Table 9) shows that our approach achieves 95.490% detection rate with 97.245% accuracy. It also has 2.007% FAR and 95.394% F1 score. This case gives the worst performance compared to the others. This is caused by the omission of *name* feature and the use of modified DFES. It also shows that the performance of modified DFES scheme in second case is highly affected by *name* feature. After we leave out that feature, the detection is dropped by 4.34%. The total time needed during this experiment is 44 seconds, which is 6 times faster compared to original DFES in case III.

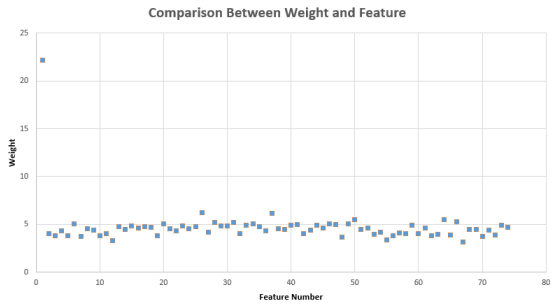


Figure 4: Weight value of each feature

4.5 Weight analysis of weight feature

Figure 4 shows that feature number 1, which is the *name* feature, has weight value around 23, much higher compared to another features that have weight value around 4. This kind of feature will cause overfitting since it will correspond exactly to some specific data, so it should not be used during training session. It acts like static antivirus detector that use database for detection process.

5 Conclusion and Future Work

From our experiments, we conclude that for 56 features as input, original DFES gives good performance in detecting malware with 99.836% detection rate and 0.049% false alarm rate, higher than Tek's [6], which has 99.351% detection rate and 0.568% false alarm rate. On the other hand, modified DFES offers lighter feature selection process, which requires 1/6 time of original DFES. However, the performance of original DFES is still better than modified DFES, which is shown by slightly higher detection rate and accuracy. We also find that in malware detection, the name of file has a big role, behaving like a database of static antivirus detector, which successfully affects 4.442% detection rate in our scheme. Overall, modified DFES achieved 95.490% detection rate, 2.007% false alarm rate, and 6 times faster processing time compared to original DFES. To the best of our knowledge, our experiment has achieved the best performance for Tek [6] dataset. In the near future, we will try to improve the performance of modified DFES by using another classifiers like SVM, J.48 tree, random forest, and k-NN.

Acknowledgement

This work was partly supported by Indonesia Endowment Fund for Education (LPDP) and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00555, Towards Provable-secure Multi-party Authenticated Key Exchange Protocol based on Lattices in a Quantum World).

References

[1] "2012 by the numbers, kaspersky lab now detects 200,000 new malicious programs every day," https://usa.kaspersky.com/, accessed: 2019-11-05.

[2] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection for wi-fi impersonation detection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 621–636, 2017.

[3] E. M. S. Alkhateeb, "Dynamic malware detection using api similarity," in *2017 IEEE International Conference on Computer and Information Technology (CIT)*. IEEE, 2017, pp. 297–301.

[4] L. Han, C. Fu, D. Zou, C. Lee, and W. Jia, "Task-based behavior detection of illegal codes," *Mathematical and Computer Modelling*, vol. 55, no. 1-2, pp. 80–86, 2012.

[5] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 116–127.

[6] "Machine learning for malware detection," https://www.randhome.io/blog/2016/07/16/machine-learning-for-malware-detection/, accessed: 2019-11-10.

[7] "pefile," https://github.com/erocarrera/pefile, accessed: 2019-11-05.

[8] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[9] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[10] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *2010 second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010, pp. 201–203.

[11] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[12] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 577–582.

[13] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[14] Z. Xu, S. Ray, P. Subramanyan, and S. Malik, "Malware detection using machine learning based analysis of virtual memory access patterns," in *Proceedings of the conference on design, automation & test in Europe*. European Design and Automation Association, 2017, pp. 169–174.

[15] L. Liu, B. Wang, B. Yu, and Q. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336–1347, 2017.

[16] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," in *International Conference on Big Data Analytics*. Springer, 2018, pp. 402–411.

[17] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[18] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[19] W. Zhong and F. Gu, "A multi-level deep learning system for malware detection," *Expert Systems with Applications*, vol. 133, pp. 151–162, 2019.

[20] Y. Liu and Y. Wang, "A robust malware detection system using deep learning on api calls," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 1456–1460.

[21] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Android malware detection using deep learning on api method sequences," *arXiv preprint arXiv:1712.08996*, 2017.

[22] K. Raman *et al.*, "Selecting features to classify malware," *InfoSec Southwest*, vol. 2012, 2012.

[23] E. Gandotra, D. Bansal, and S. Sofat, "Zero-day malware detection," in *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*. IEEE, 2016, pp. 171–175.

[24] M. E. Aminanto and K. Kim, "Detecting impersonation attack in wifi networks using deep learning approach," in *International Workshop on Information Security Applications*. Springer, 2016, pp. 136–147.

[25] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[26] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[27] K. Kim and M. E. Aminanto, "Improving detection of wi-fi impersonation by fully unsupervised deep learning," in *International Workshop on Information Security Applications*. Springer, 2017, pp. 212–223.

[28] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*. Springer, 2008, vol. 207.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[30] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[31] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[32] X. Wang, Y. Wang, and L. Wang, "Improving fuzzy c-means clustering based on feature-weight learning," *Pattern recognition letters*, vol. 25, no. 10, pp. 1123–1132, 2004.

[33] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[34] S. Mishra, R. Prusty, and P. K. Hota, "Analysis of levenberg-marquardt and scaled conjugate gradient training algorithms for artificial neural network based ls and mmse estimated channel equalizers," in *2015 International Conference on Man and Machine Interfacing (MAMI)*. IEEE, 2015, pp. 1–7.

[35] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, "Data randomization and cluster-based partitioning for botnet intrusion detection," *IEEE transactions on cybernetics*, vol. 46, no. 8, pp. 1796–1806, 2015.