6ème Conférence Internationale en Automatique & Traitement de Signal (ATS-2021)
Proceedings Book Series –PBS- Vol 1, pp.58-66

Classification of Anti-Malware Methods

Abderrahmane Hajraoui, Hoda El Merabet
Department of Physics, Faculty of Science, Abdelmalek Essaadi University
B.P. 2117 Quartier M'hanech II, Av. Palestine, Tetouan, Morocco
hajraouiabder@gmail.com
helmerabet@uae.ac.ma

Abstract— **Diverse malware programs are set up daily aiming the attack of computer systems without the knowledge of their users. While some authors of these programs intend to steal secret information, others try quietly to prove their competence and aptitude. In order to encounter these malicious codes, anti-virus programs rely basically on the traditional signature-based static technique. Although this technique excels at blocking known malware, it can never intercept new ones. Some antivirus programs may introduce the dynamic technique, which is often based on running the executables on a virtual environment. The major drawbacks of this technique are the long period of scanning and the high consumption of resources. Nowadays, recent antivirus programs introduce a third technique, the heuristic technique based on machine learning, which has proven its success in several areas based on the processing of huge amounts of data. In this paper, we expand on this latter technique, and classify its different approaches, according to their performance in the detection of malware.**

Keywords: Malware, Antivirus, Machine Learning, Feature Extraction, Random Forest, SVM, Neural Networks, Classification.

I. INTRODUCTION

The AV-TEST institute registers every day over 250,000 new malware [1]. Since novel malicious codes change constantly their signatures, static methods are not suitable to detect them. In the last two decades, the introduction of machine learning techniques had a great added value in detecting new malware. This is thanks to their generalization ability. Using relevant features as input, the chosen machine learning model learns and subsequently becomes able to make a good decision while confronted to a new file. The choice of both the appropriate input features and the classification model leads to the improvement of the results.

In this article, multiple kinds of input features used for malware detection will be reviewed. Different machine learning classification techniques deployed in this field will be examined and classified. The results will be analyzed.

II. FEATURE EXTRACTION

The choice of input features is a primary task in every machine learning research. In malware detection field, these features can be either some raw information contained in the files, or the result of processing this raw information. Both benign and malicious files are considered for the training of the chosen model. But, which features are worthy to adopt? In this section, we will review the most extracted features in machine learning researches for malware detection.

A. Signatures Extraction

Traditional antivirus programs rely on the signature-based static technique. This technique considers iteratively a known malware file, extracts some code from its header, or calculates a numerical value of it, like an MD5 hash for instance. All the obtained attributes, called signatures, are stored in a database to check each scanned file against them. Even if this technique generates no false positive, which is to say no benign file can be wrongfully designed as malicious, it would never detect new threats, as they use novel signatures.

Schultz et al. in their study in 2001 [2], entitled "Data mining methods for detection of new malicious executable", used machine learning for malware detection. As baseline, they used the signatures extraction technique. Each signature was considered as a concatenation of some existent stings in the header of each malware to be used for training.

B. DLL Function Calls Extraction

According to Schultz et al. [2], it is impossible to predict perfectly the behavior of a program without running it. However it is possible to estimate what it can do eventually. So they had the intuition that the information directing the behavior of the binary file is worthy to be extracted, that is the information related to DLL calls. Thereby, they extracted the following features in one of their three models:

- The list of DLLs used by each binary file,
- The list of DLL functions called by each file and
- The number of functions called from each DLL.

Those features were introduced using three different approaches. In the first approach, the feature vector included 30 Boolean values indicating whether a file calls a DLL or not. This vector is illustrated in Fig. 1.

$$-advapi32 \wedge avicap32 \wedge \ldots \wedge winmm \wedge -wsock32$$

Fig. 1 First feature vector: conjunction of DLL names [2]

In the second approach (Fig.2), each feature was constructed as a conjunction of a DLL name and a function call from that DLL. The feature vector consisted of 2229 Boolean values.

$$\begin{array}{ll} & dvapi32.AdjustTokenPrivileges() \\ \wedge & advapi32.GetFileSecurityA() \wedge \ldots \\ \wedge & wsock32.recv() \wedge wsock32.send() \end{array}$$

Fig. 2 Second feature vector: conjunction of DLLs and function calls [2]

ISSN: 2961-6611

In the third approach (Fig. 3), they counted the number of different functions called within each DLL. In this case, the vector of features included 30 integer values.

```
advapi32 = 2 ∧ avicap32 = 10 ∧ ...
∧ winmm = 8 ∧ wsock32 = 2
```

Fig. 3 Third feature vector: conjunction of DLLs and the number of functions called from each DLL [2]

Masud et al., in their study in 2007 [4], entitled "A hybrid model to detect malicious executables", used the PEDisassem tool [5] to disassemble the binaries. The disassembled files were analyzed, and DLL function calls were extracted. They defined an n-gram of DLL function calls as a sequence of n consecutive DLL calls appearing in a disassembled file.

The sequence in Fig.4 represents a part of a disassembled file by omitting all the instructions but the DLL calls. Fig. 5 shows the two corresponding 2-gram DLL sequences.

```
"..."; "call KERNEL32.LoadResource"; "..."; "call
USER32.TranslateMessage"; "..."; "call
USER32.DispatchMessageA"
```

Fig. 4 Part of a disassembled file (only DLL calls taken into account) [4]

```
(1) "KERNEL32.LoadResource, USER32.TranslateMessage"
(2) "USER32.TranslateMessage, USER32.DispatchMessageA"
```

Fig. 5 '2-gram' DLL sequences [4]

C. Binary Sequences Extraction

In a first approach of this technique, one takes the hexadecimal code of each binary file contained in the training database. The hexadecimal code can be seen as lines of code. Each single line, which is a sequence of sixteen consecutive bytes, is therefore considered as a single feature. Schultz et al. [2] used the hexdump utility (Miller, 2000) to convert each executable into hexadecimal code and extract the different binary sequences. Fig.6 shows an example of a hexadecimal code.

```
1f0e 0eba b400 cd09 b821 4c01 21cd 6854
7369 7020 6f72 7267 6d61 7220 7165 6975
6572 2073 694d 7263 736f 666f 2074 6957
646e 776f 2e73 0a0d 0024 0000 0000 0000
454e 3c05 026c 0009 0000 0000 0302 0004
0400 2800 3924 0001 0000 0004 0004 0006
000c 0040 0060 021e 0238 0244 02f5 0000
0001 0004 0000 0802 0032 1304 0000 030a
```

Fig. 6 Hexadecimal code example [2]

A second approach of this technique consists of converting binary sequences to n-grams. In their study entitled "Learning to detect and classify malicious executable in the wild" [3], Kolter and Maloof used the hexdump utility (Miller, 1999) to convert each executable to hexadecimal code. They produced n-grams choosing n=4, by combining each sequence of four consecutive bytes in a single term. For instance, the sequence (ff 00 ab 3e 12 b3) corresponds to the following three 4-gram sequences: (ff00ab3e), (00ab3e12) and (ab3e12b3). Each n-gram is considered as a Boolean attribute that can be either present (True) or absent (False) in the executable.

Barker et al., in their study in 2017, entitled "Malware detection by eating a whole EXE" [12], chose similarly to extract byte sequences from the files. Instead of considering raw byte values as features, they used an embedding layer to map each byte to a feature vector of fixed and learned length. They chose to avoid raw byte values, as some of them are intrinsically closer to each other, they might be seen to have a close interpretation, which is considered false a priori, since the meaning of the bytes depends on the context.

D. Assembly Sequences Extraction

Similarly to the binary n-grams extraction method, assembly n-grams can be extracted as well. After disassembling the binaries, Masud et al. [4] extracted all the n-grams from the assembly instructions.

In order to illustrate this approach, we take the sequence of assembly instructions represented by Fig. 7. The result will be the two 2-gram assembly sequences shown in Fig. 8.

```
"push eax"; "mov eax, dword [0f34]"; "add ecx, eax"
```

Fig. 7 Assembly instructions sequence [4]

```
(1) "push eax"; "mov eax, dword[0f34]"
(2) "mov eax, dword[0f34]"; "add ecx, eax"
```

Fig. 8 '2-gram' assembly sequences [4]

Siddiqui et al., in their study for Trojans detection in 2008 [6], also disassembled the binary files. The disassembly was obtained using Data rescues' IDA Pro [7]. However, they defined a sequence as a succession of assembly instructions until the arrival to a conditional or unconditional branch instruction, and/or a limit function is obtained.

To illustrate this approach, Siddiqui et al. took the assembly code shown in Fig. 9. The sequences extracted from this piece of code are shown in Fig. 10.

```
mov   dword ptr [ebp-4], 4
lea   eax, [ebp-24h]
mov   [ebp-84h], eax
mov   dword ptr [ebp-8Ch], 4008h
mov   dword ptr [ebp-94h], 8
mov   dword ptr [ebp-9Ch], 3
push  10h
pop   eax
call  __vbaChkstk
lea   esi, [ebp-8Ch]
mov   edi, esp
movsd
movsd
movsd
movsd
push  10h
pop   eax
call  __vbaChkstk
```

Fig. 9 Portion of a disassembled Trojan [6]

ISSN: 2961-6611

(1) mov lea mov mov mov mov push pop call
(2) lea mov movsd movsd movsd movsd push pop call

Fig. 10 Assembly sequences extracted from the disassembled Trojan [6]

E. PE File Header Fields Extraction

The header of the PE file consists of several fields. These fields contain structural information of the executable file. This includes dynamic library references for binding, API import and export tables, different sections contained in the file, resource management data, thread local storage data (TLS) and different types of metadata. Multiple recent studies exploit this information for malware detection.

In their structural data mining study for malware detection in 2009 [8], Shafiq et al. were able to extract initially, 189 PE features, as represented on Table I.

TABLE I: LIST OF FEATURES EXTRACTED FROM THE PE FILE [8]

Feature Description	Type	Quantity
DLLs referred	binary	73
COFF file header	Integer	7
Optional header – standard fields	Integer	9
Optional header – Windows specific fields	Integer	22
Optional header – data directories	Integer	30
.text section – header fields	Integer	9
.data section – header fields	Integer	9
.rsrc section – header fields	Integer	9
Resource directory table & resources	Integer	21
Total		189

In regard to Kumar et al., they established a malware detection model to detect maliciousness of portable executable using integrated feature set in 2017 [14]. They used as input the values of PE header fields. The set of integrated features included 68 values, consisting of 28 raw features, 26 Boolean features (expressing the existence or absence of certain values) and 14 derived features. The derived features were constructed through the validation of raw values according to a set of rules. For instance, the raw value of the TimeDateStamp field is simply an integer indicating the number of seconds since 1969. According to them, using this raw value would not be a powerful feature. Thereby, the value of this field was compared to valid dates (from December 31, 1969 at 4:00 pm until the date of the experiment). The resulting Boolean output was taken as feature. Table II summarizes all the derived features considered and their raw counterparts.

TABLE II: RAW AND DERIVED FEATURES [14]

Feature	Raw Value	Derived Value	
		Type	Value
Entropy	Binary value	Integer	[-1,0-8]
Compilation Time	Integer	Boolean	[0,1]
Section Name	String	Integer	-
Packer Info	NA	Boolean	[0,1]
FileSize	Integer	Integer	-
FileInfo	String	Integer	[0,1]
ImageBase	Integer	Boolean	[0,1]
SectionAlignment	Integer	Boolean	[0,1]
FileAlignment	Integer	Boolean	[0,1]
SizeOfImage	Integer	Boolean	[0,1]

Among other studies that also used PE header data, we cite the study of Karthik Raman in 2012 entitled "Selecting features to classify malware" [9], and the study of Pablo et al. [11], published in 2016 and entitled "Towards an effective and efficient malware detection system".

F. Machine Activity Metrics Extraction

Burnap et al. in their study "Malware classification using self-organizing feature maps and machine activity data" in 2017 [16], extracted some system-level activity metrics, by executing samples of malicious and benign executables in a Sandbox environment. These metrics are:

1. CPU User Use (percentage),
2. CPU System Use (percentage),
3. RAM use (count),
4. SWAP use (count),
5. received packets (count),
6. received bytes (count),
7. sent packets (count),
8. sent bytes (count) and
9. number of processes running (count)

G. Entropy Signals Extraction

The entropy measures the randomness in a given set of values [10]. The higher the entropy, the more random the data and thus the higher the content of information. For binary data, given that the values of a byte vary from 0 to 255, the formula used for the entropy is:

$$H = -\sum_{i=0}^{255} P_i \log_2(P_i)$$

Where Pi is the probability of i in the code.

Wojnowicz et al., in their work in 2016 entitled "Wavelet decomposition of software entropy reveals symptoms of malicious code" [17], relied merely on the entropy analysis. For each training file, several levels of resolution were chosen. For each level of resolution, the file was divided into chunks of code, and the entropy was calculated for each chunk of them, resulting in one discrete entropy signal per level of resolution. Subsequently, all the obtained signals were the features extracted in this first step.

III. FEATURE SELECTION TECHNIQUES

After the first feature extraction step, researchers mostly follow the second selection step. This step is essential for dimensionality reduction, getting rid of redundant data, reducing the learning and test times of the classifier, and thus improving the accuracy of new malware detection. The feature selection techniques, frequently used with machine learning for malware detection, will be described below.

A. Information Gain

The information gain is related to the entropy notion. It informs about the importance of a given attribute in the corresponding vector. We must therefore look for attributes with a high information gain.

ISSN: 2961-6611

After the application of the information gain to the list of n-grams and DLL calls, Masud et al. [4] reserved 500 binary n-grams, 500 assembly n-grams and 500 DLL function calls.

Masud et al. represented the equations of the entropy and the information gain as follows:

$$Entropy(S) = -\frac{p(s)}{n(s)+p(s)}\log_2\left(\frac{p(s)}{n(s)+p(s)}\right) - \frac{n(s)}{n(s)+p(s)}\log_2\left(\frac{n(s)}{n(s)+p(s)}\right)$$

$$Gain\,(S, A) \equiv Entropy\,(S) - \sum_{V \in Values\,(A)} \frac{|S_v|}{|S|} Entropy\,(S_v)$$

Where: S: training data
p(s): total number of positive instances
n(s): total number of negative instances
values (A) : set of all possible values for attribute A
|S| = p(s) + n(s)
Sv: subset of S where A= v
|Sv| = pv + nv
pv: total number of positive instances in Sv
nv: total number of negative instances in Sv

For the case of binary and assembly n-grams, an n-gram may be either present or absent. So each attribute A has only two possible values: v ε {0, 1}.

B. Redundant Feature Removal RFR

The redundant feature removal technique eliminates both the features that do not vary at all and the ones that show a significant variation. These features have an approximately uniform-random behavior. Using this technique, all the entities whose values are either constant or have a variance greater than a given threshold, will be deleted [8].

In the PE-miner study [8], Shafiq et al. employed this technique among others. Unfortunately, they did not specify the number of features obtained after its application.

C. Principal Component Analysis PCA

The principal component analysis is a procedure for reducing the number of variables and making the information less redundant. It uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of uncorrelated variables. It is at the same time a geometric approach (the variables are represented in a new space, according to maximum inertia directions) and a statistical approach (the research focuses on independent axes that best explain the variability or variance of the data) [13].

Siddiqui et al. [6] used this technique in their process of reducing the initial set of data. They started with 877 variables. After the application of PCA, they retained solely the variables that explained 95% of the full variance of the dataset. As a result, they obtained 146 variables.

D. Random Forest

The random forest technique is a prominent technique for classification and regression. It is a notable feature selection technique as well. For feature selection, it calculates the importance of an attribute by removing it from the model, then calculating the decrease in either accuracy or Gini index.

For the selection of features, Siddiqui et al. [6] used both PCA and random forest techniques, in two different approaches. Using random forest, they rejected the variables where the average decrease in accuracy was less than 10%. Thereby, they retained only 84 variables from 877.

Pablo et al. [11] took advantage of this technique as well. They combined it with another technique called Chi-Squared. They used the Chi-Squared method first, which allowed them to retain 68,800 features from a total number of 682,936 initial features that is 10% of the entire set. Then they applied the random forest technique. They chose the ranking made by accuracy decrease. The reduction passed here by successive stages. They went from 68,800 features, to 10,000 features, then to 5000, 1000, 300, 100, 30, 10, and finally to 9 features.

E. Calculation of Accuracy by Considering Each Attribute Separately

Karthik Raman in his feature selection study [9], considered the fields' values of the PE file header as features for the training of his classifier. He had the intuition that the different parts of the PE file header will be less correlated between them. Subsequently, the most important variables and the least correlated ones will be the variables generating the most important individual accuracy in each part of the header. The seven different parts of the header are: DataDirectory, OptionalHeader, Imports, Exports, Resources, Sections and FileHeader. Then, the seven fields generating the highest accuracy from each part were respectively: DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, VirtualSize2 and NumberOfSections.

F. Self-Organizing Feature Map SOFM

Self-organizing feature maps SOFMs form a class of neural networks. They can be used for either classification or dimensionality reduction. Burnap et al. [16] used SOFMs to reduce the features dimensionality. Once a sample is received, it runs on a virtual environment for 5 minutes. The chosen nine machine activity metrics are taken every second, producing 300 vectors of nine values for each sample, in the 5-minute time window. Then, SOFMs are used to transform each 9-dimensional vector to a 2-dimensional vector. Therefore, 300 vectors of x-y coordinates, are used as features for the training of the model.

G. Wavelet Transform

Various researches use the wavelet transforms for dimensionality reduction. Wojnowicz et al. [17] used this method to the entropy signals at different levels of resolution. For each level of resolution, each training file was divided into chunks of code, then the average entropy of each chunk was calculated, resulting in a discrete entropy signal. This signal was then multiplied by appropriate wavelet functions to get some wavelet coefficients. After that, the spectral energy was calculated as the sum of the wavelet coefficients squares. The spectral energies gathered from each level of resolution were used as input features for the machine learning classifier. For the highest level of resolution, the files

ISSN: 2961-6611

were divided into code chunks of 256 bytes each. For example, if a file size is 32 * 256 bytes, since $32 = 2^5$, the file will be decomposed 5 times; to 2^1, 2^2, 2^3, 2^4, and 2^5 pieces, giving 5 levels of resolution. It will subsequently generate 5 features which are the spectral energies E_1, E_2, E_3, E_4 and E_5. They used some other string features and entropy statistics for the training of their model.

IV. MACHINE LEARNING CLASSIFIERS

A. *Support Vector Machine SVM*

A support vector machine is a well-known supervised learning model for both linear and non-linear problems. For linear classification, the technique is based on finding the optimal hyperplane that separates the data into two categories. This hyperplane is the one that maximizes the margin between two parallel hyperplanes which separate the two classes of data. Its non-linear classification is obtained by applying a kernel function to map the original input space to a high-dimensional feature space, altering to a linear problem.

Masud et al. [4] used SVM as a single classification technique in their model. Siddiqui et al. [6] and Shafiq et al. [8] used SVM in addition to other classifiers, each at a time in order to make comparisons between the obtained results. As for Pablo et al. [11], they chose to make comparisons between a combination of several models, to keep the combination of SVM and neural networks in their model.

B. *Random Forest*

A random forest is a technique based on a set of classification trees. Each tree is divided at each node taking into account random features. Each tree gets its classification. Therefore, the model selects the most chosen class among all trees. Siddiqui et al. [6] built their model with 100 classification trees. The number of variables tested at each division was ranged from 6 to 43, depending on the number of selected variables in the dataset. They formed several combinations presenting several experiments, such as:

- Random forest for classification using all the 877 extracted features;
- Random forest for classification using 146 features retained by PCA feature selection technique;
- Random forest for classification using 84 features retained by random forest feature selection technique.

The best results were obtained using random forests for both feature selection and classification.

Karthik Raman [9] opted for the use of random forest as the only technique for file classification.

C. *Neural Network*

A neural network or an artificial neural network ANN is an algorithm emulating the connections between neurons in the human brain. It is constructed from interconnected nodes. Those nodes are represented in an input layer, hidden layers for the processing of input information, and an output layer for the answer. An ANN is based on a number of parameters, which are updated following a learning rule, like Gradient Descent and Backpropagation. ANNs revealed their effectiveness as a strong classification technique in many areas, especially while dealing with a huge amount of data.

Pablo et al. [11] used neural networks for classification in combination with support vector machines and random forests. After multiple tests of the three techniques, they opted for the following procedure. They started with a pretreatment of the nine best features they obtained. After that, they transformed all the original data by applying the SVM kernels to each feature, which means transforming the feature vector into another easily separable space. They subsequently used simultaneously three sets of data to constitute the input layer of the neural network classifier. These three sets are:

- The initial set of data, built of nine features,
- The set of features transformed by SVM kernels and
- The results of the SVM classifier applied to the initial set of nine features.

Their model gave an increase in both accuracy and speed of training. The training passed actually from few hours to few minutes.

Barker et al. [12] chose neural networks as well. After the input layer, they introduced an embedding layer, followed by convolutional layers, recurrent neural networks and finally a fully connected layer. Convolutional neural networks are widely used in image processing because of their ability to learn the existence of a feature regardless of its position. Barker et al. found that the MS-DOS header is the only component of the PE file that has a fixed position. Each of PE header, code, resources, etc. can be placed anywhere. To better capture such a high-level localization invariance, they chose to use a convolutional neural network architecture.

V. CLASSIFICATION OF THE STUDIED RESEARCHES

There are several indicators to measure the performance of a given classifier. For the classification of the different studied researches in this paper, we are interested in the accuracy rate of each one of them. Accuracy is defined as the number of malicious files classified as malicious plus the number of benign files classified as benign, divided by the total number of the files. The results are shown in Table III.

We should mention that for [16], the precision rate was used instead of the accuracy. It is the number of malicious files classified as malicious, divided by the number of all the files classified as malicious. There are other metrics to measure the performance like the false positive rate, the true positive rate, the false negative rate, the true negative rate, etc.

Several researches investigated in this paper use the k-fold cross-validation. This technique partitions the existing data set into iterative learning and test subsets, but does not use a new subset for the final test of the model. This can lead to an overfitting of the model to the training data, subsequently it would fail to generalize perfectly to previously unseen data.

Therefore, an important factor is to check whether a classification model could generalize from previously seen data in the model training, to the new data exclusively used for the last test phase.

ISSN: 2961-6611

TABLE III CLASSIFICATION OF THE OBTAINED RESULTS

Ref	Features	Feature Selection	Machine Learning Classifier	Accuracy
[16]	300 vectors of 9 machine activity metrics taken in a 5-minute time window.	SOFMs. 300 vectors of x-y coordinates obtained	Random forest	86,70%
[14]	68 raw and derived values from the PE header fields	None	Random forest	89,23%
[12]	Byte (mapped) sequences taken from the file bodies	None	CNN + RNN + ANN	90,90%
[6]	877 assembly n-grams from the file bodies	Random forest: 84 features retained	Random forest	94,00%
[16]	300 vectors of 9 machine activity metrics taken in a 5-minute time window.	SOFMs. 300 vectors of x-y coordinates obtained	Logistic regression	94,60%
[4]	Binary n-grams, assembly n-grams and DLL function calls	Information gain: 1500 features retained	SVM	96,30%
[11]	682936 features: PE info, DLLs and other static and dynamic information from VirusTotal site [15]	Chi-squared, random forests. 9 features adopted	SVM and ANN	98,40%
[17]	Most common strings observed in file corpus + entropy statistics + file entropy signals	Wavelet transform of the entropy signals	Logistic regression	98,90%

Burnap et al. [16] performed a first experiment using 10-fold cross validation. In a second experiment, they used a new unseen set of data for the final test. By comparing the two experiments, we remark that the results of the random forest model decreased by more than 12% from the first experiment to the second one, whereas those of the ANN model decreased by 2.45% only. That demonstrates that a model based on an ANN provides stability between training and test datasets.

Pablo et al. [11] also made such a comparison. In the first experiment they obtained an accuracy of 99.60%, and after the application of their model to new malicious files, whose date of appearance was located after the date of the files used for training, the new accuracy was 98.40%. The results of the ANN model decreased here by just 1.20%.

The references [11], [12], [16] and [17] are the only ones in this paper that used unseen data for the final tests, yet they are the ones that have frequently achieved the best results.

In Table III, the results of [4], [6] and [14] were taken into account for comparison reasons, knowing that it is very likely that they will decrease by using unseen data.

VI. CONCLUSIONS

The transformation of the input dataset into another easily exploitable space, brings a high gain in both data processing time and obtained results. This was achieved in the researches of this paper through the use of either the kernel functions defined by SVMs, or SOFMs, or the wavelet transform of the entropy signal which has shown its potential utility.

The use of random forest for feature selection provides a significant benefit in reducing both the size of the dataset and the processing time.

The logistic regression model, in its relative simplicity, has shown its efficacy in the researches that used it in this paper. Several reasons could be indicators for an exceeding success by using neural networks instead of logistic regression. Neural networks are based, in several cases, on the sigmoid function as activation function between the layers of the network. This function is the same used in logistic regression. Neural networks have a major aptitude of generalization. Deep neural networks are generally more powerful.

The metrics taken from system-level activities could very well train the logistic regression classification model. However, it is difficult to introduce such features into anti-malware programs because of their high execution time.

REFERENCES

[1] AV-TEST. *The Independent IT Security Institute*. [Online]. Available: https://www.av-test.org/en/statistics/malware/.

[2] G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. *Data Mining Methods for Detection of New Malicious Executables*. Proceedings 2001 IEEE, Symposium on Security and Privacy. S&P 2001, pp. 38–49, USA.

[3] J. Zico Kolter and Marcus A. Maloof, *Learning to Detect and Classify Malicious Executables in the Wild*. Published on 'Journal of Machine Learning Research 7' (2006) 2721-2744.

[4] Mohammad M. Masud, Latifur Khan and Bhavani Thuraisingham, *A hybrid model to detect malicious executables*. Proceedings of IEEE. Conference of Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007.

[5] Windows P.E. Disassembler. [Online]. Available: http://www.heaventools.com/PE_Explorer_disassembler.htm

[6] Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee, *Detecting Trojans Using Data Mining Techniques*. Springer-Verlag Berlin Heidelberg 2008.

[7] IDA Pro Disassembler. [Online]. Available: http://www.datarescue.com/idabase/index.htm

[8] M. Zubair Shafiq, S. Momina Tabish, Fauzan Mirza and Muddassar Farooq. *Pe-miner: Mining Structural Information to Detect Malicious Executables in Realtime*. Raid 2009: Recent Advances in Intrusion Detection pp 121-141.

[9] Karthik Raman, *Selecting Features to Classify Malware*. Adobe Systems Incorporated, 2012

[10] [Online]. Available: http://www.forensickb.com/2013/03/file-entropy-explained.html

[11] Chia Tien Dan Lo, Ordonez Pablo and Cepeda Mora Carlos. *Towards an Effective and Efficient Malware Detection System*. Published on December 2016. Proceedings of IEEE International Conference on Big Data

[12] Edward Raff , Jon Barker , Jared Sylvester , Robert Brandon, Bryan Catanzaro and Charles Nicholas. *Malware Detection by Eating a Whole EXE*. Published on Research Gate, October 2017.

[13] Principle Component Analysis. [Online]. Available: https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales

[14] Kumar, A., Kuppusamy, K.S., Aghila, G., *A Learning Model to Detect Maliciousness of Portable Executable Using Integrated Feature Set*, Journal of King Saud University - Computer and Information Sciences (2017).

[15] VirusTotal site. [Online]. Available: https://www.virustotal.com

[16] Pete Burnap, Richard French, Frederick Turner and Kevin Jones. *Malware Classification Using Self-Organizing Feature Maps and Machine Activity Data*. Published on Elsevier Ltd 2017.

[17] Michael Wojnowicz, Glenn Chisholm, Matt Wolff and Xuan Zhao. *Wavelet Decomposition of Software Entropy Reveals Symptoms of Malicious Code*. Published on Elsevier Ltd 2016.

ISSN: 2961-6611