A Survey of Malware Detection Techniques based on Machine Learning

Hoda El Merabet[1], Abderrahmane Hajraoui[2]
Department of Physics Faculty of Science
Abdelmalek Essaadi University
Tetouan, Morocco

***Abstract*—Diverse malware programs are set up daily focusing on attacking 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. The traditional signature-based static technique is primarily used by anti-malware programs in order to counter these malicious codes. Although this technique excels at blocking known malware, it can never intercept new ones. The dynamic technique, which is often based on running the executable on a virtual environment, may be introduced by a number of anti-malware programs. The major drawbacks of this technique are the long period of scanning and the high consumption of resources. Nowadays, recent programs may utilize a third technique. It is 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 provide a survey of available researches utilizing this latter technique to counter cyber-attacks. We explore the different training phases of machine learning classifiers for malware detection. The first phase is the extraction of features from the input files according to previously chosen feature types. The second phase is the rejection of less important features and the selection of the most important ones which better represent the data contained in the input files. The last phase is the injection of the selected features in a chosen machine learning classifier, so that it can learn to distinguish between benign and malicious files, and give accurate predictions when confronted to previously unseen files. The paper ends with a critical comparison between the studied approaches according to their performance in malware detection.**

Keywords*—Malware; anti-malware; machine learning; feature extraction; feature selection; random forest; SVM; neural networks; classification

I. Introduction

Every day, the AV-TEST1 institute registers over 250000 new malware. As Windows is one of the most universally used operating systems nowadays, it becomes an attractive target for malicious attacks. In this paper, we focus on researches built for the detection of malware designed for Windows operating systems.

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 has contributed significant value in detecting new malware, due to their generalization ability. Machine learning models are based on two stages: training and prediction, as illustrated in Fig. 1. The training stage relies on a training dataset that contain samples of benign and malicious files. It involves three phases. The first phase consists of extracting a large number of features from the different files in the training dataset. The second phase consists of rejecting non-pertinent features based on appropriate selection techniques. The third phase consists of using one or more classification models that will learn to distinguish between malicious and benign files. These models subsequently become able to give accurate predictions dealing with new executable files in the prediction stage. The choice of both appropriate input features and classification model leads to the improvement of prediction rates.

In this article, multiple kinds of input features used for malware detection will be reviewed. Different machine learning classification techniques deployed in the field of security 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 either be raw information contained in the files that will be examined, or the result of processing raw information. Both benign and malicious files are considered for the training of the chosen machine learning model.

So, which features are worthy to adopt? In this section, we will elucidate this issue by reviewing the most extracted features in machine learning researches for malware detection.

A. Signatures Extraction

Traditional commercial anti-malware programs basically rely on the signature-based static technique. This technique iteratively considers a known malware file, extracts code from its header, or calculates a numerical value from it, like a hash code for instance. The obtained attributes, called signatures, are stored in a database to check each new scanned file against them. Although 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 [1], used machine learning for malware detection. As baseline, they utilized the signatures extraction technique. Signatures were calculated as follows: they analyzed all the files available in the training set, then took the byte-sequences that were existent in the

[1]AV-TEST. *The Independent IT Security Institute*..

malicious files and missing in the benign ones. Later, these byte-sequences were concatenated together to construct a unique signature for each malicious file. The final data mining model [1] had a detection rate double than the detection rate of this signature-based scanner, while dealing with new binaries.

B. DLL Function Calls Extraction

Multiple researches rely on the extraction of the information related to the Microsoft Windows libraries DLLs and the API functions [1, 3, 5, 7, 13, 15, 20].

According to Schultz et al. [1], it is impossible to perfectly predict the behavior of a program without running it. However, it is possible to estimate what it can eventually do. They assumed that the information directing the behavior of the binary file is worthy to be extracted. They thereby 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
- The number of functions called from each DLL

These features were introduced using three different approaches. The first approach, illustrated in Fig. 2, considered 30 DLL libraries. The feature vector included 30 Boolean values indicating whether a file imports a DLL or not.

In the second approach, illustrated in Fig. 3, each feature was constructed as a conjunction of a DLL file name and an API function called from that DLL. The feature vector consisted of 2229 Boolean values.

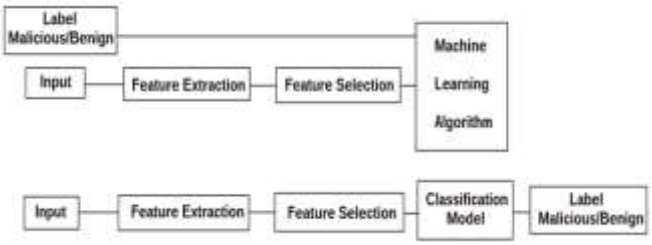

Fig. 1. Machine Learning Stages: Training and Prediction.

-advapi32 ∧ avicap32 ∧ ... ∧ winmm ∧ -wsock32

Fig. 2. First Feature Vector: Conjunction of DLL Names [1].

dvapi32.AdjustTokenPrivileges()
∧ advapi32.GetFileSecurityA() ∧ ...
∧ wsock32.recv() ∧ wsock32.send()

Fig. 3. Second Feature Vector: Conjunction of DLLs and Function Calls [1].

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

Fig. 4. Third Feature Vector: Conjunction of DLLS and the Number of Functions Called from Each DLL [1].

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

Fig. 5. Part of a Disassembled File (Only DLL Calls Taken into Account) [3].

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

Fig. 6. '2-Gram' DLL Sequences [3].

In the third approach, they took each file in the training set, and counted the number of API functions called per imported DLL. In this case, the vector of features included 30 integer values. Fig. 4 gives an example of a feature vector illustrating this approach.

Masud et al., in their study in 2007 [3], used the Windows P.E. Disassembler tool to disassemble the binaries. They extracted DLL function calls from the disassembled files, by omitting all other instructions. They subsequently built n-gram sequences. Each n-gram was defined as a sequence of n consecutive DLL calls, appearing in a disassembled file. A feature vector corresponding to an executable is a binary vector having one bit for each feature. The bit's value is 'one' to signify that the feature is present in the file, otherwise it is 'zero'.

The list of instructions shown in Fig. 5 represents a part of a disassembled file by omitting all the instructions from the code and reserving only DLL calls. Fig. 6 shows the two corresponding 2-gram DLL sequences.

C. Binary Sequences Extraction

In a first approach of this technique, one takes the hexadecimal code of each file contained in the training dataset. 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. [1] used the hexdump utility (Miller, 2000) to convert each executable into hexadecimal code and extract the different binary sequences. Fig. 7 shows an example of a hexadecimal code.

A second approach of this technique consists of converting binary sequences to n-grams [2, 14, 15]. Kolter and Maloof [2] 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 following 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 scanned executable.

Barker et al. similarly chose to extract byte sequences from the benign and malicious files in their study in 2017 [10]. Some bytes are intrinsically closer to each other, they might be seen to have a close interpretation. This interpretation is considered false a priori, since the meaning of the bytes depends on the context. For that reason, Barker et al. decided to avoid raw byte values. They used an embedding layer to map each byte to a feature vector of fixed and learned length, instead of considering raw byte values as features.

D. Assembly Sequences Extraction

Opcode sequences or assembly sequences are used by several researches to learn and detect malicious functionalities in the executable files [3, 4, 16, 19].

After disassembling the binaries, Masud et al. [3] extracted all the n-grams from the assembly instructions. In order to illustrate there approach, they took the sequence of assembly instructions represented by Fig. 8. For n=2, the extracted features from this sequence were the two 2-gram assembly sequences shown in Fig. 9.

Siddiqui et al. opted for disassembling the binary files to extract features in their study for Trojans detection in 2008 [4]. The disassembly was obtained using Data rescues' IDA Pro disassembler. 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. 10. The extracted features from this piece of code are shown in Fig. 11.

```
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. 7. Example of Hexadecimal Code [1].

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

Fig. 8. Assembly Instructions Sequence [3].

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

Fig. 9. '2-Gram' Assembly Sequences [3].

```
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. 10. Portion of a Disassembled Trojan [4].

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

Fig. 11. Assembly Sequences Extracted from the Disassembled Trojan [4].

E. PE File Header Fields Extraction

The portable executable (PE) format is a file format for executable files and object files under the Windows family of operating systems. It is a data structure that encapsulates the information necessary for the Windows operating system loader to manage the wrapped executable code.

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, source management data, thread local storage data (TLS), and different types of metadata. Recent studies exploit the values of the PE file headers in order to train machine learning models and detect new malware [5, 6, 8, 11, 13, 19].

In their data mining study in 2009 [5], Shafiq et al. were able to extract initially, 189 PE features, as represented on Table 1.

In regard to Kumar et al., they opted for the use of an integrated feature set in 2017 [11]. They used the values of PE header fields as inputs for their model. 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 that they specified. For instance, the raw value of the Time Date Stamp 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 a feature. Table 2 summarizes all the derived features considered and their raw counterparts.

TABLE I. LIST OF FEATURES EXTRACTED FROM THE PE FILE [9]

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

TABLE II. RAW AND DERIVED FEATURES [11]

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]

F. Machine Activity Metrics Extraction

Several researches use performance metrics to reveal the process behavior [12, 18]. These metrics can be obtained by executing the samples in a sandbox or in a virtual environment.

Burnap et al. extracted some system-level activity metrics in their study in 2017 [12], by executing samples of malicious and benign executables in a sandbox environment. These metrics are:

- CPU User Use (percentage)
- CPU System Use (percentage)
- RAM use (count)
- SWAP use (count)
- received packets (count)
- received bytes (count)
- sent packets (count)
- sent bytes (count)
- number of processes running (count)

These features are continuous values. They allow the ability to be more flexible with the classification of samples than discrete features such as DLL calls and PE header fields. At the same time they are more difficult to obfuscate through cyber-attacks, according to Burnap et al.

Abdelsalam et al. [18] executed the samples in a virtual machine. Then, they collected 28 features from the following eight categories:

- Status
- CPU information
- Context switches
- IO counters
- Memory information
- Threads
- File descriptors
- Network information

G. Entropy Signals Extraction

The entropy measures the randomness in a given set of values. 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 represented by (1):

$$H = -\sum_{i=0}^{255} Pi \log_2(Pi) \quad (1)$$

Where, Pi is the probability of i in the code.

Various researches represent the content of the executable file as an entropy stream, where each value denotes the entropy of a small piece of code in a specific location in the file [9, 21].

Wojnowicz et al., in their work in 2016 [9], relied merely on the entropy analysis. For each training file, several levels of detail or resolution were chosen. For each level of resolution, the file was divided into chunks of code, and the entropy was calculated for each chunk, resulting in one discrete entropy signal per level of resolution. For instance, for the level of resolution 2, the file will be divided into $2^2 = 4$ chunks, so 4 entropy values will be generated, producing a signal of 4 discrete values. Subsequently, all the obtained signals are considered as the features extracted in this first step.

III. FEATURE SELECTION TECHNIQUES

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

A. Information Gain

Information gain has been widely used for feature selection [3, 14, 20]. Its notion is related to the entropy notion. It informs about the importance of a given attribute in the corresponding vector. One therefore must look for attributes with a high information gain.

After the application of the information gain to the list of n-grams and DLL calls, Masud et al. [3] 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 by (2) and (3) respectively.

$$\text{Entropy(S)} = -\frac{p(s)}{n(s)+p(s)} log_2\left(\frac{p(s)}{n(s)+p(s)}\right) - \frac{n(s)}{n(s)+p(s)} log_2\left(\frac{n(s)}{n(s)+p(s)}\right) \quad (2)$$

$$\text{Gain(S, A)} = \text{Entropy(S)} - \sum_{v \epsilon Values(A)} \frac{|Sv|}{|S|} Entropy(Sv) \quad (3)$$

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: $v \varepsilon \{0,1\}$.

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 [5].

In the PE-miner study [5], 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).

Siddiqui et al. [4] 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 data set. As a result, they obtained 146 variables, a considerable reduction of the number of features.

After using the information gain as a first feature selection technique, Zhang et al. [20] utilized the PCA as a second feature selection technique in their research in 2018. They finally retained 50 features to train their classification model. They didn't specify the initial number of features.

D. Random Forest

The random forest technique is a prominent technique for classification and regression. Nevertheless, 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. These two metrics are used to evaluate the classification models and to explain their performance. The chosen attributes are the ones that imply a significant decrease in the chosen evaluation metric when removed.

For the selection of features, Siddiqui et al. [4] used both PCA and random forest techniques, in two different approaches. Using the 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. [12] 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, which is 10% of the entire set. Then, they applied the random forest technique. They chose the ranking made by accuracy decrease. The reduction passed 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 uniquely.

E. Calculation of Accuracy by Considering Each Attribute Separately

Karthik Raman, in his feature selection study [6], considered the fields' values of the PE file header as features for the training of his classifier. He was convinced 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 PE file header are: Data Directory, Optional Header, Imports, Exports, Resources, Sections, and File Header. The study of Karthik Raman revealed that the seven fields generating the highest accuracy from each part are respectively: Debug Size, Image Version, IatRVA, ExportSize, ResourceSize, VirtualSize2, and NumberOfSections. He retained solely these seven features to train his machine learning algorithm.

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. [12] used SOFMs to reduce the features dimensionality. Once a sample is received, it runs through a virtual environment for 5 minutes. The chosen nine machine activity metrics, mentioned in section II.F, 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 [9, 21]. Wojnowicz et al. [9] applied the wavelet transform 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 values called 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 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 subsequently generates 5 features which are the spectral energies E_1, E_2, E_3, E_4 and E_5. In addition to these spectral energies, Wojnowicz et al. integrated additional 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 hyper plane that separates the data into two categories. This hyper plane is the one that maximizes the margin between two parallel hyper planes which separate the two classes of data, in our case benign and malicious files. Its non-linear classification is obtained by applying a kernel function, which is a mapping function, to map the original input space to a high-dimensional feature space, creating a linear problem.

Masud et al. [3] used SVM as a single classification technique in their model. Siddiqui et al. [4], Shafiq et al. [5], and Ninyesiga and Ngubiri [15] used SVM in addition to other classifiers, one at a time in order to make comparisons between the obtained results. Li et al. [13] used SVM and neural networks with different types of features to conclude the best combination between features and classification model. Their best results were obtained using SVM with features derived from the filename, path, static properties of the file, and imported functions. As for Pablo et al. [8], they chose to make comparisons between combinations of several models. They ultimately kept the combination of SVM and neural networks in their model, which was the combination that obtained the highest accuracy rate.

B. Random Forest

A random forest is an ensemble learning method used for classification and regression. It is constructed from a collection of decision trees. Each tree determines the class label of an unlabeled instance and then gets its classification. Each tree is divided at each node taking into account random features. Therefore, the model selects the most chosen class among all trees. The larger the number of trees, the more accurate is the result. Siddiqui et al. [4] 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 data set. They formed several combinations presenting several experiments, such as:

- Random forest for classification using all the 877 initially 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. Bai et al. [7] also obtained the best results using random forests as classification model, they compared it to other decision tree models.

C. Neural Network

A neural network or an artificial neural network (ANN) is a brain-inspired system intended to replicate the way that we humans learn. It is constructed from interconnected nodes. These nodes are represented in three forms of layers. An input layer consisting of input features, hidden layers that process the input information and transform it to something that the output layer can use, and an output layer which is responsible for giving the answer. An ANN is based on a number of parameters, which are updated iteratively in the learning phase. At each iteration, the ANN makes a prediction, then the error between its prediction and the correct answer is calculated based on a chosen cost function, after that the parameters are adjusted according to a learning rule like Gradient Descent and Backpropagation. The iterations are repeated until obtaining a minimum error [22]. ANNs revealed their effectiveness as a strong classification technique in many areas, especially in dealing with a huge amount of data.

The classification model of Pablo et al. [8] was constituted from neural networks 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 their selected nine features. After that, they transformed all the original data by applying SVM kernels, which are mapping functions, to each feature. That is to say transform the feature vector space into another space easily separable. Then, they simultaneously used 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
- 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 time lessened from few hours to few minutes.

Barker et al. [10] chose neural networks as well. After the input layer, they introduced an embedding layer, followed by convolutional layers, recurrent neural networks RNNs, and finally a fully connected layer. Convolutional neural networks CNNs 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. The other parts like the PE header, the code and the resources can be placed anywhere. To better capture such a high-level localization invariance, they chose to use a convolutional neural network architecture.

Multiple other researchers utilized convolutional neural networks in their works. Abdelsalam et al. [18] and Yan et al. [19] chose to represent each sample as an image (2D matrix)

which will be the input to a convolutional neural network. They obtained great results.

Boydell et al. [17] used a generic image scaling algorithm, where the raw malware byte code is interpreted as a one dimensional 'image' and is scaled to a fixed target size. According to them, their approach is simpler than converting a malware binary file to a 2D image before doing classification since one doesn't have to make the decision about the height and the width of the image. The raw static byte code is used as input to a convolutional neural network followed by a recurrent neural network. However their work was intended to identify the malware class from nine classes and not to decide if a file is benign or malicious.

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, this paper was 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 files. Table 3 shows our results taking into account the most important researches.

We should mention that Burnap et al. [12] didn't calculate the accuracy in their experiments. They used instead the precision rate. It is the number of malicious files classified as malicious, divided by the number of all the files classified as malicious.

Several researches investigated in this paper use the k-fold cross-validation. It is a resampling procedure used to evaluate machine learning models on a limited data sample. The technique partitions the existing dataset into iterative learning and test subsets. It is applied to estimate the efficiency of a machine learning model on unseen data. However, since it does not use a completely new subset for the final test of the model, this can lead to an overfitting of the training data, the model subsequently would fail to perfectly generalize 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 new data exclusively used for the last test phase.

Burnap et al. [12] performed their 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 shows that a model based on an ANN provides more stability between training and test datasets.

Pablo et al. [8] 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 in this case by just 1.20%.

As for Abdelsalam et al. [18], their best model considered performance metrics collected over a time interval as inputs to a convolutional neural network. They obtained an accuracy of 97% with the validation dataset, this result dropped to 90% while testing with the new test dataset. Here we see that the accuracy rate of the results of the convolutional neural network model decreased by 7%.

TABLE III. Classification of the Obtained Results

Ref	Features	Feature Selection	Machine Learning Classifier	Accuracy
[12]	300 vectors of 9 machine activity metrics (taken each second in a 5-minutes time window).	SOFMs. 300 vectors of x-y coordinates obtained	Random forest	86.70%
[18]	128 vectors of 28 performance metrics (taken each 10 seconds in a 30-minutes time window)	None	CNN	90%
[10]	Byte (mapped) sequences taken from the file bodies	None	CNN + RNN + ANN	90.90%
[4]	877 assembly n-grams from the file bodies	Random forest: 84 features retained	Random forest	94.00%
[12]	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%
[3]	Binary n-grams, assembly n-grams and DLL function calls	Information gain: 1500 features retained	SVM	96.30%
[8]	682936 features: PE info, DLLs and other static and dynamic information from VirusTotal website	Chi-squared, random forests. 9 features adopted	SVM and ANN	98.40%
[9]	Most common strings observed in file corpus + entropy statistics + file entropy signals	Wavelet transform of the entropy signals	Logistic regression	98.90%

The references [8], [9], [10], [12] and [18] are the only malware detection researches 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 3, the results of [3] and [4] were taken into account for comparison reasons, knowing that it is very likely that their accuracy rates will decrease while using unseen data for the final performance tests.

VI. Conclusions

The transformation of the input dataset into another easily exploitable space brings a great gain in both data processing time and performance measures. This is illustrated in this paper through the use of either SOFMs, or the wavelet transform of the entropy signal, or the kernel functions defined by SVMs.

The use of random forest for feature selection provides a significant benefit in reducing both the size of the dataset and the processing time, and in increasing the accuracy rate as well.

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 and deep learning 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. Deep neural networks have a major aptitude of generalization and are very powerful as shown in the best results of Table 3. It can be a very good way to design a smart anti-malware program.

The metrics taken from system-level activities could very well train the classification models. The introduction of such features into anti-malware programs might be slightly difficult, because of the high execution time and the significant consumption of resources. Besides, the use of deep neural networks has an intense computational requirement. However, faster and more powerful processors are showing up continuously allowing the application of the above-mentioned techniques in more ease.

References

[1] G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. “Data mining methods for detection of new malicious executables”. IEEE, Symposium on Security and Privacy, pp. 38–49, USA S&P 2001.

[2] J. Z. Kolter and A. M. Maloof, “Learning to detect and classify malicious executables in the wild”. Journal of Machine Learning Research 7, vol. 6, pp. 2721-2744, 2006.

[3] M. M. Masud, L. Khan, and B. Thuraisingham, “A hybrid model to detect malicious executables”. IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007, pp. 1443-1448.

[4] M. Siddiqui, M. C. Wang, and J. Lee, “Detecting trojans using data mining techniques”. Hussain D.M.A., Rajput A.Q.K., Chowdhry B.S., Gee Q. (eds) Wireless Networks, Information Processing and Systems. IMTIC 2008. Communications in Computer and Information Science, vol 20. Springer, Berlin, Heidelberg.

[5] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. “Pe-miner: Mining structural information to detect malicious executables in realtime”. Kirda E., Jha S., Balzarotti D. (eds) Recent Advances in Intrusion Detection. RAID 2009. Lecture Notes in Computer Science, vol 5758. Springer, Berlin, Heidelberg.

[6] K. Raman, “Selecting features to classify malware”. Adobe Incorporated Systems, 2012

[7] J Bai, J. Wang, and G. Zou. “A malware detection scheme based on mining format information”, The Scientific World Journal; June 2014.

[8] C. T. D. Lo, O. Pablo, and C. M. Carlos. “Towards an effective and efficient malware detection system”. IEEE International Conference on Big Data, Washington, DC, USA, December 2016.

[9] M. Wojnowicz, G. Chisholm, M. Wolff, and X. Zhao. “Wavelet decomposition of software entropy reveals symptoms of malicious code”. Journal of Innovation in Digital Ecosystems, vol. 3, issue 2, December 2016, pp. 130-140.

[10] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. “Malware detection by eating a whole EXE”. AAAI Workshops, 2018.

[11] A. Kumar, K.S. Kuppusamy, and G. Aghila, “A learning model to detect maliciousness of portable executable using integrated feature set”, Journal of King Saud University Computer and Information Sciences, 2017, ISSN: 1319-1578.

[12] P. Burnap, R. French, F. Turner, and K. Jones. “Malware classification using self-organizing feature maps and machine activity data”. Computers and Security 2018, vol. 73, pp. 399-410.

[13] L. Bo, R. Kevin, G. Chris, and V. Yevgeniy. “Large-scale identification of malicious singleton files”. 7TH ACM Conference on Data and Application Security and Privacy, pp. 227-238 Scottsdale, Arizona, USA, March 22 - 24, 2017.

[14] M. Yousefi-Azar, L. Hamey, V. Varadharajan, and S. Cheng. ”Malytics: A malware detection scheme”, arXiv: 1803.03465 [cs.CR], March 2018.

[15] A. Ninyesiga and J. Ngubiri. “Behavioral malware detection by data mining”, LAP LAMBERT Accademic Publishing; October 2018.

[16] M. Sewak, S. Sahay, and H. Rathore. “An investigation of a deep learning based malware detection system”. 13th International Conference on Availability, Reliability and Security. Article N. 26, Hamburg, Germany, August 27-30, 2018.

[17] Q. Le, O. Boydell, B. M. Namee, and M. Scanlon, “Deep learning at the shallow end: Malware classification for non-domain experts”. Digital Investigation, vol. 26, supplement, pp. S118-S126. 18th Annual DFRWS USA July 2018.

[18] M. Abdelsalam, R. Krishnan, Y. Huang, and R. Sandhu. “Malware detection in cloud infrastructures using convolutional neural networks”. 11th IEEE International Conference on Cloud Computing (CLOUD), San Francisco, CA, July 2-7, 2018.

[19] J. Yan , Y. Qi , and Q. Rao. “Detecting malware with an ensemble method based on deep neural network”. Security and Communication Networks, vol. 2018, Article ID 7247095, 16 pages, 2018.

[20] J. Zhang, K. Zhang, Z. Qin, H. Yin, and Q. Wu. “Sensitive system calls based packed malware variants detection using principal component initialized multiLayers neural networks”. Cybersecurity (2018) 1: 10. https://doi.org/10.1186/s42400-018-0010-y.

[21] D. Gibert, C. Mateu, J. Planes, and R. Vicens. “Classification of malware by using structural entropy on convolutional neural networks”. 30th AAAI Conference on Innovative Applications of Artificial Intelligence (IAAI-2018).

[22] Michael A. Nielson, “Neural networks and deep learning”, Determination Press, 2015.