JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

A Chi-Square-Based Decision for Real-Time Malware Detection Using PE-File Features

Mohamed Belaoued* and Smaine Mazouzi**

Abstract

The real-time detection of malware remains an open issue, since most of the existing approaches for malware categorization focus on improving the accuracy rather than the detection time. Therefore, finding a proper balance between these two characteristics is very important, especially for such sensitive systems. In this paper, we present a fast portable executable (PE) malware detection system, which is based on the analysis of the set of Application Programming Interfaces (APIs) called by a program and some technical PE features (TPFs). We used an efficient feature selection method, which first selects the most relevant APIs and TPFs using the chi-square (KHI²) measure, and then the Phi (φ) coefficient was used to classify the features in different subsets, based on their relevance. We evaluated our method using different classifiers trained on different combinations of feature subsets. We obtained very satisfying results with more than 98% accuracy. Our system is adequate for real-time detection since it is able to categorize a file (Malware or Benign) in 0.09 seconds.

1. Introduction

Computer systems are a real asset for organizations and corporations in today's technology-driven economy. Unfortunately, they are confronted with several kinds of cyber attacks. Among the most used attacks are those based on installing and running a malware in target machines. Malware (e.g., Trojans, viruses, worms, etc.) are computer programs, which are designed to accomplish unauthorized actions without the user's consent. Malware can be used to steal or damage data or to disrupt network traffic. Moreover, the compromised machines can be used as "zombies" in order to conduct targeted attacks, such as DDOS [1]. Recently, we are witnessing an unprecedented and a very concerning proliferation of malware. McAfee has reported that more than 30 million new malware were discovered during the first quarter of 2014; that number was doublethe amount for the the same period of 2013 [2,3]. This malicious software infects thousands of computers every day, and Microsoft Windows operating system (OS) remains the most affected, as it is the most used OS worldwide. During January 2015, about 80% of computers ran on MS Windows OS [4]. Under the latter operating system, malware are often present as legitimate executable files, and they can have any known file extension (e.g., exe, com, etc.).

Therefore, they are structured according to a common executable file format called PE (portable executable) [5] (see Section 2).

Efficiently protecting computers against malware attacks has become vital for companies, corporations, and individuals. Unfortunately, the existing commercial antivirus (AV) software is unable to provide the required level of protection. Techniques used by most AVs are signature-based. A signature is a short and unique string of bytes, which is recorded in the signature's database for each known malware so that future examples of it can be correctly classified with a small error rate. Such techniques suffer from two major drawbacks: First, they need to have a prior knowledge of malware by their signatures. Therefore, they are unable to detect unknown or newly launched (zero-day) ones [6,7]. Second, they are inefficient against metamorphic malware (i.e., variants of known malware) [6,8]. Recently, AV software has become more effective by using more sophisticated analysis techniques. However, there is still the problem of delay, which can vary from a few hours to several days, between the detection of a new malware and the updating of the viral databases by the AV firms [7,9,10]. During this period, the malware could cause considerable damages.

In the last decade, researchers employed more sophisticated malware detection methods, such as behavioral and heuristic analyses [11,12], as an alternative to signature-based ones. Behavioral analysis, which is also known as dynamic analysis, consists of observing the behavior of a program at runtime, by monitoring its execution in an isolated environment (Sandbox or virtual machine). During the monitoring process, the actions that the program accomplishes (such as library uses, kernel calls, network traffic, and registry updates) are recorded and used to generate features for classifying the program (malware or benign). The main advantage of dynamic analysis is that it can detect unknown and metamorphic malware [10]. However, new evasive methods that can detect the analysis environment and completely stop the execution of the malware code or delay the execution of its malicious content until the monitoring process is terminatedhave been introduced. Note that the monitoring process is run for a couple of minutes at most, which itself is inconvenient since it cannot observe the entire capabilities of a given program as not all execution paths can be explored [10]. Furthermore, such techniques are not suitable for real-time purposes.

On the other hand, heuristic-based analysis uses data mining and machine learning techniques to learn the behavior of the program [11]. Such techniques investigate different file features, such as Opcode instructions, structural information, and APIs. These sets of information are represented in different forms (e.g., control flow graphs, n-grams, etc.) and are used as features for the classification process, which is generally done using different classifiers, such as decision trees and Bayes algorithm. API calls were widely used for constructing anti-malware systems [6,8,10,13-16], since they can provide valuable information related to the possible behaviors that a given program could have [16]. Indeed, the operating system provides APIs to user applications allowing them to request its services. Also, some authors [12] have investigated structural information due to its robustness against code obfuscation techniques, such as packing.

The heuristic-based anti-malware systems are able to detect unknown and metamorphic malware [11,17]. They are also easy to implement compared to the behavioral ones. However, heuristic-based systems suffer from the inconvenience of their high false positive rate (i.e., benign programs that are wrongly detected as malware) [11]. In order to overcome this drawback, most of the existing systems combine a large number of different types of features [11]. This can lead to intensive computations and

increase the processing time. Due to that, most of the actual heuristic-based anti-malware systems are inadequate for real-time detection, which is a suitable characteristic, especially in a dynamic environment (i.e., a malware analysis environment). Therefore, being able to find an appropriate balance between accuracy and detection time is a real challenge when constructing anti-malware systems.

In this work, we introduce an efficient heuristic-based PE-malware detection system, which has a real-time response and a low false alarm rate. The proposed system is composed of a feature extraction module, a pre-processing module, a feature selection module, and a decision module. The feature extraction module statically (i.e., without running the program) extracts the set of API calls from the Import Address Table (IAT), and the technical PE features (TPF) from the PE-optional header (see Section 2). The feature selection module is based on the KHI² test, which is a statistical method used for hypothesis testing [18]. We introduce the Phi (φ) measure for feature set reduction, which means that the decision will be taken according to the most significant features. The latter allows for a real-time response. The decision module is based on different data mining based classifiers that have been implemented in the Waikato Environment for Knowledge Analysis (WEKA) [19]. For experimentation purposes, we used two sets of files – a benign set and a malware set – that we obtained from well-known dedicateddatabases. To the best of our knowledge, our work is the first to use a combination of the KHI² measure and the φ coefficient as a feature reduction method and to use this combination with a hybridization of features (APIs + TPFs).

This paper is organized as follows: in Section 2, we introduce the PE file format and its structure, in order to facilitate comprehension about the rest of the sections. Section 3 is devoted to the best-known related works, published in the literature. In Section 4, we present the proposed malware detection system, and in Section 5 we present our experimental results. In Section 6, we introduce our analysis of the obtained results, and we compare the performance of our system with the existing ones. Section 7 concludes our work and underlines its perspectives.

2. PE File Format

PE is the common file format for binary executables and DLLs in Windows OS. A PE file is mainly composed of an MS-DOS header, PE header, and Section headers (section table), and a set of sections [5], as shown in Fig. 1.
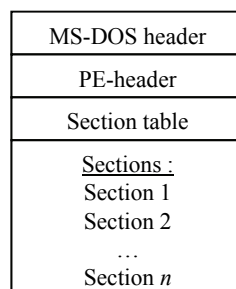
MS-DOS header
PE-header
Section table
Sections : Section 1 Section 2 … Section n

Fig. 1. Simplified PE-file structure.

- The MS-DOS header is located at the beginning of the PE-file and it is used to check whether it is a valid executable or not, when the file is run under DOS.

- The PE-header contains some important pieces of information about the PE-file. The PE-header is an IMAGE_NT_HEADERS data structure that contains three members, which are signature, file header, and the optional header (Fig. 2).

- The PE-optional headerisan IMAGE_OPTIONAL_HEADER structure, and it is composed of several fields, such as Magic, MajorLinkerVersion, and CheckSum [5]. One of the PE-optional header fields is the data directory, which is an array of 16 IMAGE_DATA_DIRECTORY structures. One of these structures is the IMAGE_DIRECTORY_ENTRY_IMPORT, which contains information about all the imports (DLLs/APIs). The import entry points to a vector of IMAGE_IMPORT_DESCRIPTOR structures. The field "OriginalFirstThunk" of the latter structure contains an RVA (Relative Virtual Address), which points to the IAT, as shown in Fig. 2. The IAT is an array of function pointers that contains elements of IMAGE_THUNK_DATA structures. Each structure corresponds to an imported API, and they contain the ordinal of a function or an RVA to an IMAGE_IMPORT_BY_NAME structure. The latter contains the names of the APIs that the code calls.
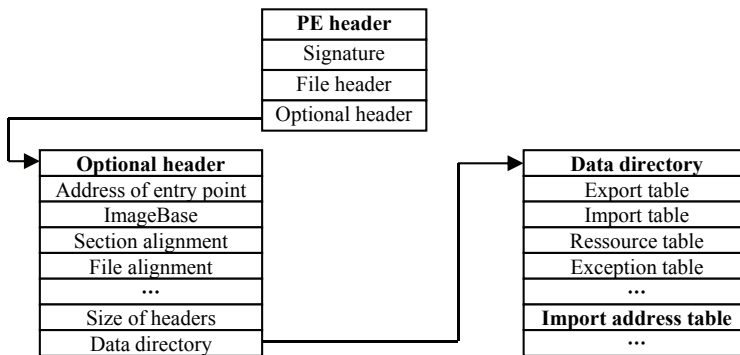

Fig. 2. Structure of the PE-optional header and location of the IAT.

3. Related Works

In this section, we provide a brief description of some often-cited works in the literature that have used APIs, PE features, or a KHI²-based decision to detect malware files.

Schultz et al. [16] introduced the first anti-malware system based on machine learning techniques. They investigated different information in the PE file, such as strings, API functions, and byte sequence. They used a classification method based on the naïve Bayes algorithm, and they obtained an overall accuracy of 97.11%. The method introduced in [13] uses naïve Bayes classification with API calls. The extracted APIs were used to construct models of suspicious behaviors, by grouping some APIs based on scenarios that a malware can accomplish, such as searching for a file to infect and writing malicious data into it. They obtained 93.7% accuracy.

Ye et al. [6] proposed CIMDS, which is an improvement of their previous malware detection system called IMDS (Intelligent Malware Detection System) [8]. As its predecessor, CIMDS is based on Object

Oriented Association (OOA) mining. They used the KHI² method for rule pruning (removing insignificant rules), rule ranking (from most significant to least significant), and rule selection (best *n*APIs). CIMDS was the first work that used post-processing techniques. The system achieved 67.5% accuracy and 88.16% detection rate, which still needs improvements. However, it has a very good detection time with 0.09 seconds per file. The authors in [14] also presented a malware detection system based on OOA mining and API calls. They proposed a feature selection method to reduce the number of obtained APIs by selecting the top 1,000 ones based on two criteria, which are document frequency and information gain. The system achieved 91.2% accuracy and a detection rate of 97.3%.

The authors in [17] combined the KHI² test andthehidden Markov model (HMM) for detecting malware using an Opcode sequence. They extracted Opcode from the analyzed files using the third party disassembler IDA Pro. They used the KHI² test to identify the set of instructions that are most likely to be used by the NGVCK malware generator to generate malware variants. They then learned the HMM using the set of obtained instructions in order to categorize the files. They tested their method on 200 malware and 40 benign programs. Their proposed system had an overall accuracy of 91%. The drawback of this method is the usage of a closed source disassembler (IDA Pro), which makes their system not fully automatic. Moreover, their analysis was limited to malware that is generated using NGVCK.

The method described in [15], extracts API calls and compares them to previously constructed models of malicious APIs stored in a signature's database. A similarity measure between the extracted model and the existing one was calculated by combining three different metrics, which are cosine similarity, extended Jaccard measure, and Pearson correlation. After that, they selected the most relevant features using a weighting scheme. The weight of an API was obtained by calculating the product of two metrics, which are term frequency and inversed document frequency. The classification phase was based on a Support Vector Machine (SVM) classifier and they obtained an overall accuracy of 91.5%.

The authors in [10] proposed a malware detection system that is based on analyzing API functions and their arguments. They used a dynamic feature extraction method using a virtual environment. They evaluated their method using different classifiers, and they obtained an overall accuracy of 98.1%. Using arguments requires program being executed. Consequently, this method suffers from the inconveniences of dynamic approaches, which were previously mentioned.

PE-Miner [12] was introduced as a real-time framework for PE-malware detection. This framework statically extracts PE-files structural features from the PE header, optional header, etc. The authors proposed a method based on information gainto select the most relevant features. They evaluated their framework using five different data mining classifiers. The proposed framework was able to achieve a detection rate ofmore than 99% and a false alarm rate of about 5%. The categorization process takes an average of 0.244 seconds per file.

4. Proposed System

Our proposed method for PE-malware categorization is based on the observation that there are some PE-file features that are specific to malware and others to benign programs. These features appear with different frequencies between these two categories of PE-files. Therefore, these features must be

extracted and sorted in a way that we consider to be only the most significant ones that will make a significant contribution in the benign-malware categorization process. Thus, the statistical KHI² test is used to estimate the significance (relevance) of these features. Thus, we will provide, at the same time, a real-time detection of PE-malware and a high specific categorization using small sets of features.

4.1 System Architecture

Our proposed system for malware detection is composed of four different modules, which are the feature extraction module, pre-processing module, feature selection module, and the decision module, as shown in Fig. 3.
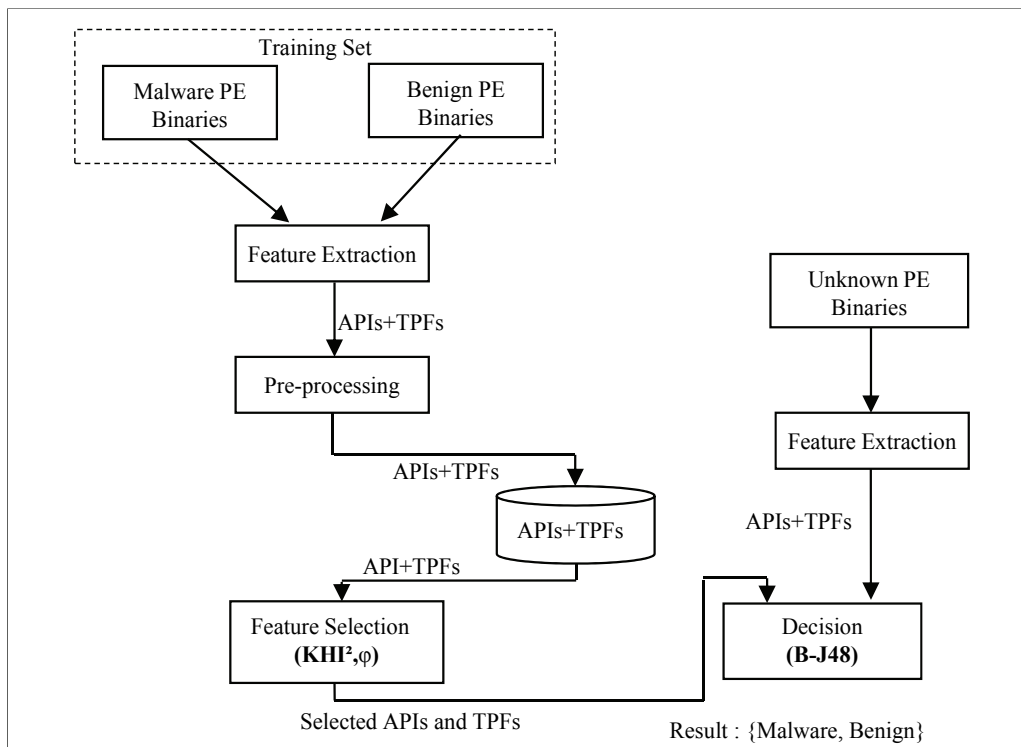
4.2 Feature Extraction

As mentioned previously, our system relies on the analysis of two different types of features, which are API calls and some TPFs. The TPFs are represented by the information stored in the PE-optional header fields. In order to extract these two types of features from a PE file and calculate their frequencies, we developed a module written in Python that uses a third-party Python module called Pefile, which is a multi-platform module for reading and working with PE files and extracting different information from them [20]. This extraction method is based on a static analysis of the IAT for API calls and the PE-optional header for the TPFs (see Section 2). Given below in Figs. 4 and 5, is an overview of some lines of Python code that we wrote to extract API calls and TPFs from multiple PE files contained in a folder named "Malware_Samples."

Note that TPFs are represented by concatenating the field's name with its value (e.g., 'checksum0'), as shown in lines 8 and 9 of the above source code (Fig. 5).

```
1    import pefile
2    import os
3    API_list = []
4    PATH = "C:\Malware_Samples"
5    for FILE in os.listdir(PATH)
6      current_file =os.path.join(PATH, FILE)
7      pe= pefile.PE(current_file)
8      for entry in pe.DIRECTORY_ENTRY_IMPORT:
9        for API in entry.imports:
10         API_LIST.append(API.name)
11   ...
```

Fig. 4. Python source code for extracting API calls (for API in entry.imports) and storing them in APIs structure (API_LIST).

```
1    import pefile
2    import os
3    TPF_list = []
4    PATH = "C:\Malware_Samples"
5    for FILE in os.listdir(PATH)
6      current_file =os.path.join(PATH, FILE)
7      pe= pefile.PE(current_file)
8      TPF_LIST.append("CheckSum"+str(pe.OPTIONAL_HEADER.CheckSum))
9      TPF_LIST.append("LoaderFlags"+str(pe.OPTIONAL_HEADER.LoaderFlags))
10   ...
```

Fig. 5. Python source code for extracting TPFs (pe.OPTIONAL_HEADER…) and storing them in TPFs structure (TPF_LIST).

4.3 Pre-processing

After extracting APIs and TPFs using the previous module, the pre-processing module proceeds in removing duplicated APIs in the same PE file, and then it calculates their call frequencies in malware and benign files. The module also calculates the frequency of appearances of the obtained TPFs in malware and benign programs. At the end of this phase we obtained a table composed of three columns that, respectively, represent the feature's name, its frequency in malware PE, and its frequency in benign PE, and a number of rows, which is equal to the number of obtained features (APIs + TPFs) (see Subsection 5.2).

4.4 Feature Selection

The third module of our system is the feature selection module. It was also developed as a Python script, and it aims at selecting the most relevant features from the obtained list of APIs and TPFs. This module is based on a well-known statistical method, which is the KHI² hypothesis test [18]. This method is used to decide whether there is a significant association between two qualitative variables. This association is expressed by the distance D between an observed frequency O and an expected one E, which represents the case of perfect independence between the variables). Therefore, the correlation strength between two variables is proportional to the distance D. In our case, we studied the association between two variables: First, the variable "Feature" that has two modalities of "present" and "not

present." This variable represents the presence or absence of a feature (API or TPF) in a PE file. Second, the variable "PE-cat," which also has the two modalities of "malware" and "benign." This variable represents the two categories of PE files, for instance: "malware" and "benign." When conducting a KHI² test, we first started by defining the two hypotheses H_0 and H_1, where one will be accepted and the other rejected. H_0 (null hypothesis) represents the case of independence between the two variables. H_1 (alternative hypothesis) represents the case of dependency between the two variables. In our case, H_0 and H_1 are defined as follows:

- **H_0**: The presence or absence of a feature (API, TPF) is independent of the PE file's type (malware or benign).
- **H_1**: The presence or absence of a feature (API, TPF) is related to the PE file's type (malware or benign).

For each feature **F**, we had a contingency table, as shown in Table 1.

Table 1. Contingency table of a feature F

	Feature: present	**Feature: not present**	**Row total**
PE-cat: malware	M_1	M_2	M
PE-cat: benign	N_1	N_2	N
Column total	M_1+N_1	M_2+N_2	T

We describe below the different variables presented in Table 1.

- M and N are, respectively, the total number of malware and benign PE files.
- T is the total number of used PE files ($T=M+N$).
- M_1 is the number of malware PE files that contain F, and M_2 is the number of malware PE files that do not contain F, such as $M=M_1+M_2$.
- N_1 is the number of benign PE files that contain F, and N_2 is the number of benign PE files that do not contain F, such as $N= N_1+N_2$.

Based on the contingency table (Table 1), the KHI² score (D^2) is calculated using Eq. (1).

$$D^2 = \sum \frac{(O_{r,c}-E_{r,c})^2}{E_{r,c}} \tag{1}$$

where, $O_{r,c}$ is the observed frequency count at level r of the row variable and level c of the column variable. And $E_{r,c}$ is the expected frequency, which is defined by the following equation :

$$E_{r,c} = \frac{n_r \times n_c}{T} \tag{2}$$

where, n_r and n_c are, respectively, the sum of row r and the sum of column c. After calculating the KHI² values for the extracted features, we had to determine which of the two hypotheses H_0 or H_1 would be accepted or rejected for every feature. To do that, we compared the obtained KHI² values (D^2) of every feature to a threshold, which represents the theoretical KHI² value (χ^2), and H_0 was accepted (H_1 rejected) for every feature that had $D^2 \leq \chi^2$. Note that according to the KHI² hypothesis test, the features

for which H_0 is accepted are considered to be irrelevant and will be systematically removed. The χ^2 value is obtained by first calculating the degree of freedom (DF), and choosing a signification level α that represents the probability of rejecting a hypothesis even if is true. Considering DF and α, the χ^2 value is obtained from the KHI² distribution table [21]. DF is calculated as follows:

$$DF = (R - 1) \times (C - 1) \tag{3}$$

where R and C represent, respectively, the number of modalities of the first and the second variables. After removing all the features that are not correlated, which correspond to the case where H_0 is accepted, we calculated the φ coefficient for the remaining features. The φ coefficient [22] is a normalization of the KHI² score (D^2), which can only be applied to a 2×2 contingency tables (two variables with two modalities). It is used to measure the strength of the dependency between the two variables [23]. In our work, this coefficient was used to group features in subsets according to their correlation strength (relevance). φ is calculated as follows:

$$\varphi = \sqrt{\frac{D^2}{T}} \tag{4}$$

The value of φ ranges between 0 and 1, and the relevance of a feature (API or TPF) is proportional to that value. Therefore, we chose to use three subsets that would contain features that have φ≥0.25, φ≥0.5, and φ≥0.75, respectively. Our aim was to be able to identify the optimal number of features required to have the highest accuracy, which also helps to reduce the detection time.

4.5 Decision

After generating the different features' subsets from the previous module, we had to identify which of the APIs or TPFs or combinations of them provided the best results (i.e., high accuracy and low detection time). Therefore, we first evaluated our system using TPFs subsets. We then used APIs subsets, and, finally, we evaluated it by trying every possible combination of the TPFs-APIs subsets. We used different classification algorithms available in WEKA, which are the decision tree (J48) [19], boosted decision tree (B-J48) using the AdaBoostM1 algorithm [19], Rotation Forest (Rot-F) [24], and Random Forest (Ran-F) [25]. Our decision module took the features' subsets generated from the feature selection module and the set of extracted ones from the analyzed fileas input. Both are represented as WEKA data files (.arff file) [19], which are automatically generated using a Python script.

5. Experimentation

5.1 Dataset

For experimentation purposes, we collected a dataset composed of 552 PE files (338 malware and 214 benign programs), of 80% were used as the training set and 20% as the test set. The infected PE files were downloaded from Vxheavens.com and contained 12 different malware categories, as shown in Table 2.

Table 2. Malware dataset

No.	Malware type	Counts
1	Backdoor	27
2	Email-Worm	19
3	Exploit	28
4	Hacktool	22
5	Net-Worm	16
6	P2P-Worm	17
7	Trojan	59
8	Trojan-Downloader	24
9	Trojan-Dropper	32
10	Trojan-Spy	18
11	Virus	42
12	Worm	34
	Total	338

The benign PE files include some utility software that was downloaded from Softpedia.com, and also some Windows system files that were collected from a clean installation of Windows XP. In our work, we only considered non-packed (non-compressed) programs. Packing is a method that is legitimately used by software developers to protect their programs from reverse engineering, and malware creators use it to hide the malicious code from being detected by AV software [26]. Therefore, we analyzed our dataset using well-known Packers detection tools, such as PEiD, and ProtectionID. These tools are able to detect a large variety of packers, including popular ones like UPX, ASPack, and PECompact. Note that packed binaries are the only category that was excluded from our samples. We also scanned all the files using more than 40 different AVs from the website VirusTotal.com. More than 30 AVs identified the infected PE files that were used as malware, and none of the benign PE files were identified as malware.

5.2 Results

In this subsection, we present the results obtained from the feature extraction phase to the decision phase. After the feature extraction and pre-processing phases, we obtained the results presented in Tables 3 and 4.

Table 3. Overview of the obtained TPFs

No.	Optional-header field	Value	Frequency	
			Malware (271)	Benign (172)
1	BaseOfCode	4096	271 (100)	172 (100)
2	BaseOfData	102400	4 (1)	1 (1)
3	BaseOfData	106496	3 (1)	1 (1)
4	BaseOfData	110592	1 (1)	0 (0)
5	BaseOfData	118784	2 (1)	0 (0)
…	…	…	…	…
586	SizeOfUninitializedData	802816	1 (1)	0 (0)
587	SizeOfUninitializedData	8192	1 (1)	0 (0)
588	SizeOfUninitializedData	95744	1 (1)	0 (0)
589	Subsystem	2	226 (83)	106 (62)
590	Subsystem	3	45 (17)	66(38)

Data are presented as number (%).

Table 4. Overview of the obtained APIs

No.	API name	Frequency	
		Malware (271)	Benign (172)
1	Abort	4 (1)	1 (1)
2	Accept	21 (8)	0 (0)
3	Ace_Cleanup_Destroyer	1 (1)	0 (0)
4	ActivateKeyboardLayout	63 (23)	4 (2)
5	AddAccessAllowedAce	1 (1)	18 (10)
…	…	…	…
1636	xml_setcharacterdatahandler	1 (1)	0 (0)
1637	xml_setelementhandler	1 (1)	0 (0)
1638	xml_setuserdata	1 (1)	0 (0)
1639	zwquerysysteminformation	1 (1)	0 (0)
1640	zwunmapviewofsection	1 (1)	0 (0)

Data are presented as number (%).

We calculated the KHI² values for the obtained APIs and TPFs and removed the irrelevant ones that had KHI² ≤ 3.84 (3.84 is the χ^2 value for DF=1 and alpha=0.05). Thus, H_0 was rejected and H_1 was accepted for 681 APIs and 50 TPFs, and H_0 was accepted and H_1 was rejected for 959 APIs and 540 TPFs. The results are presented in Tables 5 and 6.

Table 5. Overview of the obtained KHI² and φ values for the selected TPFs

No.	TPFs	KHI²	φ
1	CheckSum0	375.21	0.92
2	MajorImageVersion0	370.57	0.91
3	DllCharacteristics0	355.91	0.9
4	MajorOperatingSystemVersion5	346.02	0.88
5	MinorOperatingSystemVersion0	341.92	0.88
…	…	…	…
46	BaseOfData49152	4.03	0.1
47	SizeOfImage28672	3.95	0.09
48	MinorLinkerVersion55	3.86	0.09
49	SizeOfImage163840	3.86	0.09
50	SizeOfInitializedData28672	3.86	0.09

Table 6. Overview of the obtained KHI² and φ values for the selected APIs

No.	APIs	KHI²	φ
1	__p__commode	254.59	0.76
2	__setusermatherr	254.59	0.76
3	_exit	254.59	0.76
4	_xcptfilter	254.59	0.76
5	_controlfp	245.57	0.74
…	…	…	…
677	Getenhmetafiledescriptiona	3.86	0.09
678	GetSockName	3.86	0.09
679	Ischaralphaa	3.86	0.09
680	Ntohl	3.86	0.09
681	GetCurrentThread	3.85	0.09

As mentioned previously, we obtained a final list of 681 APIs and 50 TPFs with their corresponding KHI² scores and φ values, as shown in Tables 5 and 6. We divided these features into different groups (subsets) according to their φ values. At the end of the feature selection phase, we obtained three subsets for APIs, which are A1, A2, and A3, and three subsets for TPFs, which are H1, H2, and H3. The latter subsets correspond, respectively, to the three φ values, which are φ≥0.75, φ≥0.5, and φ≥0.25. We obtained 5, 31, and 297 APIs in A1, A2, and A3, respectively. We also obtained 11, 14, and 22 TPFs in H1, H2, and H3. We used a fourth subset for both APIs (A4) and TPFs (H4) that contained all of the extracted features (1,640 APIs and 590 TPFs). The purposed of using these two additional subsets was to see whether the proposed feature selection method improved our system's performance or not.

In the next subsection, we evaluate our system's performance by using different classifiers with the obtained features' subsets and see which subset or combination of subsets generates the best results.

5.3 Evaluation

Our experiments were conducted on a Windows 7 OS, I3-2350M 2.30 GHZ CPU, and 4GB of RAM computer. The feature extraction, pre-processing, and selection modules were implemented in Python 27. The decision module was implemented in WEKA 3.7.

The performance of a malware detection system is generally evaluated according to three different metrics, as discussed below.

- Detection rate (DR): This represents the percentage of malware detectedamong all malware of the given test set, and it is calculated using Eq. (5):

$$DR = \frac{Number\ of\ detected\ malware}{Total\ number\ of\ malware} \times 100\% \tag{5}$$

- False alarm rate (FA): This is the percentage of benign files wrongly classified as malware among all the benign files of the given test set, and it is calculated using Eq. (6):

$$FA = \frac{Number\ of\ benign\ files\ classified\ as\ malware}{Total\ number\ of\ benign\ files} \times 100\% \tag{6}$$

- **Accuracy (AC):** This represents the rate of files that were correctly classified in their class, and it is calculated using Eq. (7):

$$AC = \frac{Number\ of\ correctly\ classified\ files}{Total\ number\ of\ files} \times 100\% \tag{7}$$

Since we want to achieve a real-time detection of malware, we must take into consideration the detection time (DT) as a fourth metric to evaluate our system's performance. DT represents the average time required for categorizing a given PE file from our test set, and it is expressed in seconds per file. The obtained results are presented in Tables 7–9.

Table 7. Experimental results using TPFs subsets

Subset	φ	TPF counts	Classifier	DR (%)	FA (%)	AC (%)	DT (s)
H1	≥0.75	**11**	J48	100	9.52	96.33	0.074
			B-J48	100	9.52	96.33	0.076
			Ran-F	100	9.52	96.33	0.074
			Rot-F	98.51	7.14	96.33	0.074
H2	≥0.5	**14**	J48	98.51	9.52	95.41	0.074
			B-J48	97.01	7.14	95.41	0.074
			Ran-F	100	9.52	96.33	0.074
			Rot-F	**100**	**7.14**	**97.25**	**0.077**
H3	≥0.25	**22**	J48	98.51	9.52	95.41	0.074
			B-J48	95.52	9.52	93.58	0.074
			Ran-F	98.51	7.14	96.33	0.074
			Rot-F	98.51	9.52	95.41	0.076
H4	-	**590**	J48	98.51	9.52	95.41	0.081
			B-J48	85.07	4.76	88.99	0.086
			Ran-F	89.55	4.76	91.74	0.081
			Rot-F	97.01	7.14	95.41	0.116

We can see from the results presented in the table above that our system has the highest AC with the subset H2 (97.25%), with an improvement of 1.84% compared with the obtained AC using the subset H4 (no feature selection). In addition, the average detection time was also reduced (-0.004s).

Table 8. Experimental results using APIs subsets

Subset	φ	API counts	Classifier	DR (%)	FA (%)	AC (%)	DT (s)
A1	≥0.75	**5**	J48	100	28.57	88.99	0.076
			B-J48	100	28.57	88.99	0.076
			Ran-F	100	28.57	88.99	0.076
			Rot-F	100	28.57	88.99	0.076
A2	≥0.5	**31**	J48	95.52	11.90	92.66	0.078
			B-J48	97.01	7.14	95.41	0.078
			Ran-F	98.51	14.29	93.58	0.078
			Rot-F	**98.51**	**7.14**	**96.33**	**0.081**
A3	≥0.25	**297**	J48	94.03	11.90	91.74	0.087
			B-J48	97.01	9.52	94.50	0.087
			Ran-F	98.51	14.29	93.58	0.086
			Rot-F	97.01	9.52	94.50	0.110
A4	-	**1640**	J48	95.52	9.52	93.58	0.138
			B-J48	97.01	9.52	94.50	0.147
			Ran-F	100	11.90	95.41	0.136
			Rot-F	97.01	9.52	94.50	0.343

The results presented in Table 8 show that our system is more accurate with the subset A2 and Rot-F classifier. Compared with the subset A4 there was an improvement of 1.27%. The detection time was reduced by 40% from 0.136s (A4+Ran-F) to 0.081s (A2+Rot-F). By comparing the results in Table 7 and Table 8, we can see that TPFs provided a better accuracy (97.28%) compared with APIs (96.20%), and a better detection time (-0.004s). Next, we combined both APIs and TPFs in order to increase accuracy. The results are presented in Table 9 and include only the best classifier scores.

As can be seen from the obtained results, we were able to improve the accuracy of our system, compared with TPFs only (+0.67%) and APIs only (+1.84%), and that by combining H2 and A3 subsets using B-J48. Furthermore, we were able to keep a good detection time with an average of 0.090 s. We were also able to achieve the same accuracy with the combination of H3 and A3 subsets using the same classifier, but with a detection time of 0.092 s. Therefore, we prefer to consider the first combination (H2+A3). From the results previously presented, we can conclude that the proposed feature selection method made a very important contribution in increasing the system's accuracy and in reducing the detection time.

Table 9. Experimental results (best classifier) using combinations of API-TPF subsets

Subset	Classifier	DR (%)	FA (%)	AC (%)	DT (s)
H1+A1	B-J48	100	9.52	96.33	0.080
H1+A2	J48	100	7.14	97.25	0.080
H1+A3	Ran-F	100	9.52	96.33	0.089
H1+A4	Rot-F	98.51	7.14	96.33	0.333
H2+A1	B-J48	97.01	7.14	95.41	0.079
H2+A2	B-J48	95.52	7.14	94.50	0.080
H2+A3	**B-J48**	**100**	**4.76**	**98.17**	**0.090**
H2+A4	B-J48	98.51	4.76	97.25	0.145
H3+A1	Rot-F	100.00	7.14	97.25	0.081
H3+A2	B-J48	95.52	7.14	94.50	0.080
H3+A3	**B-J48**	**100**	**4.76**	**98.17**	**0.092**
H3+A4	B-J48	98.51	4.76	97.25	0.157
H4+A1	Rot-F	98.51	7.14	96.33	0.148
H4+A2	J48	98.51	7.14	96.33	0.103
H4+A3	B-J48	97.01	9.52	94.50	0.124
H4+A4	B-J48	98.51	4.76	97.25	0.187

6. Comparison and Discussion

In this section, we evaluate the efficiency of our system by comparing our obtained results to those of the previously cited systems, as presented in Table 10 and Fig. 6.

Table 10. Comparison of our results with previously cited methods having used the same evaluation metrics

Method	Features used	DR (%)	AC (%)
Our method	API + TPFs	100	98.17
CIMDS [6]	API	88.16	67.50
Ding et al. [14]	API	97.30	91.20
Salehi et al. [10]	API + Arguments	99.20	98.40
Schultz et al. [16]	Strings	97.43	97.11
Wang et al. [13]	API	94.40	93.71

From the results presented in the above table, is can be seen that our system outperforms four of the five presented systems in terms of accuracy, with an improvement that varies from 1.06% to 30.67%. The method proposed by Salehi et al. [10] had a better accuracy than our method with an improvement rate of 0.23%. However, our system has the best detection rate compared to all of these systems.
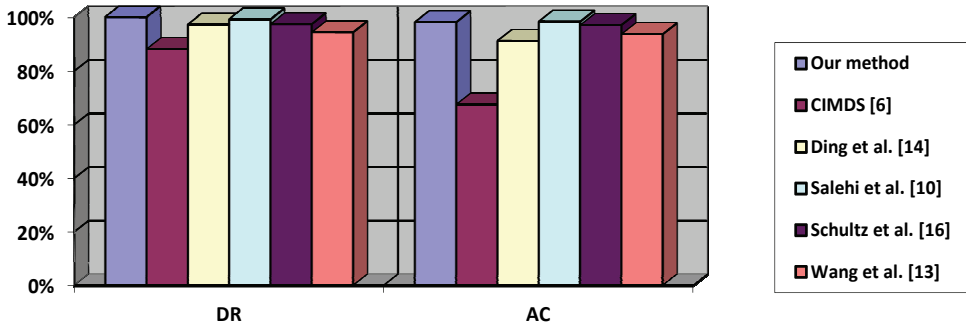
Fig. 6. Histogram comparing our system's AC and DR with those of the previously cited ones.

Considering the detection time (categorization time), we can conclude that our system is adequate for the real-time detection of malware, since it only requires 0.090s for the whole categorization process. Our system has the same detection time as CIMDS [6], and it is almost three times faster than PE-Miner [12], as the latter takes 0.244s to categorize a file.

Our proposed feature extraction and pre-processing method took 0.040s, 0.037s, and 0.041s for APIs, TPFs, and APIs + TPFs, respectively. These results are very satisfying compared with the method used in [10], which needs to monitor the analyzed program for two minutes in order to extract the API calls and their arguments, which is almost 3,000 times the amount of time required for our proposed method.

The time cost for generating the .arff file depends on the number of features used. For instance, it took 0.037s with A1+H1, 0.049s with H3+A3, and 0.142s for H4+A4. Note that the arff file generation process is related to the usage of the WEKA environment. Therefore, its time cost was not taken into consideration when the classifier was directly implemented and integrated in our system.

The time cost for the classification process depends on the number of used features and the classification method. For instance, with H1+A1 subsets and the B-J48 classifier, the classification process took 0.001s, with H2+A3 using the same classifier it took 0.004s, and using the Rot-F classifier it took 0.119s.

7. Conclusion and Future Works

In this work, we have presented a fast PE-malware detection system that is based on the analysis of API calls and TPFs. The KHI² test was used as the feature selection method and was combined with the φ coefficient, which was used to select the optimal number of features. Different classification algorithms were used to evaluate our system. The results show that our system is more accurate when the APIs and TPFs are combined using a B-J48 classifier, as we achieved an overall accuracy of 98.27%. Our system is automatic and can be used for the real-time detection of malware. As for future work, we will try to increase the accuracy of our system by combining other types of features and at the same time, we will try to keep the same rapidity. We alsointend to study the different API associations that are specific to the different malware categories, which will allow for the multiclass categorization of malware.

References

[1] C. V. Zhou, C. Leckie, and S. Karunasekera, "A survey of coordinated attacks and collaborative intrusion detection," *Computers & Security*, vol. 29, no. 1, pp.124-140, 2010.

[2] McAfee threat report: first quarter 2013 [Online]. Available: http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q1-2013.pdf.

[3] McAfee Labs threats report: first quarter 2014 [Online]. Available: http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q1-2014.pdf.

[4] W3Schools, "OS platform statistics," [Online]. http://www.w3schools.com/browsers/browsers_os.asp.

[5] M. Pietrek, "Peering inside the PE: a tour of the win32 portable executable file format," *Microsoft Systems Journal*, vol. 9, no. 3, pp. 15-38, 1994.

[6] Y. Ye, T. Li, Q. Jiang, and Y. Wang, "CIMDS: adapting postprocessing techniques of associative classification for malware detection," *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, vol. 40, no. 3, pp. 298-307, 2010.

[7] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16-29, 2009.

[8] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, no. 4, pp. 323-334, 2008.

[9] "Anti-malware vendors slow to respond," *Computer Fraud & Security*, vol. 2010, no. 6, pp. 1-2, 2010.

[10] Z. Salehi, A. Sami, and M. Ghiasi, "Using feature generation from API calls for malware detection," *Computer Fraud & Security*, vol. 2014, no. 9, pp. 9-18, 2014.

[11] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *Proceedings of the 5th Conference on Information and knowledge Technology (IKT)*, Shiraz, Iran, 2013, pp. 113-120.

[12] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: mining structural information to detect malicious executables in realtime," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, Saint-Malo, France, 2009, pp. 121-141.

[13] C. Wang, J. Pang, R. Zhao, and X. Liu, "Using API sequence and Bayes algorithm to detect suspicious behavior," in *Proceedings of the International Conference on Communication Software and Network*, Macau, China, 2009, pp. 544-548.

[14] Y. Ding, X. Yuan, K. Tang, X. Xiao, and Y. Zhang, "A fast malware detection algorithm based on objective-oriented association mining," *Computers & Security*, vol. 39, pp. 315-324, 2013.

[15] M. K. Shankarpani, K. Kancherla, R. Movva, and S. Mukkamala, "Computational intelligent techniques and similarity measures for malware classification," in *Computational Intelligence for Privacy and Security*. Heidelberg: Springer, 2012, pp. 215-236.

[16] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the IEEE Symposium onSecurity and Privacy*, Oakland, CA, 2001, pp. 38-49.

[17] H. Toderici and M. Stamp, "Chi-squared distance and metamorphic virus detection," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 1, pp. 1-14, 2013.

[18] P. Fornasini, "The chi square test," in *The Uncertainty in Physical Measurements*. New York: Springer, 2008, pp. 187-198.

[19] H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Burlington, MA: Morgan Kaufmann Publishers, 2011.

[20] pefile [Online], http://code.google.com/p/pefile/.

[21] S. Kokoska and C. Nevison, *Statistical Tables and Formulae*. New York: Springer, 1989.

[22] B. Chedzoy, "Phi-coefficient," in *Encyclopedia of Statistical Sciences*, 2nd ed. Hoboken, NJ: Wiley, 2006.

[23] D. P. Farrington and R. Loeber, "Relative improvement over chance (RIOC) and phi as measures of predictive efficiency and strength of association in 2×2 tables," *Journal of Quantitative Criminology*, vol. 5, no. 3, pp. 201-213, 1989.

[24] J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619-1630, 2006.

[25] L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[26] A. Singh and A. Lakhotia, "Game-theoretic design of an information exchange model for detecting packed malware," in *Proceedings of the 6th International Conference on Malicious and Unwanted Software (MALWARE)*, Fajardo, Puerto Rico, 2011, pp. 1-7.

Mohamed Belaoued http://orcid.org/0000-0002-9412-1959

He received his B.A. and M.A. degrees both in computer science from the University of Skikda, Algeria in 2009 and 2011, respectively. He is now and since 2011 a PhD student within the department of computer science of the same university. His main research interests include computer security, distributed systems, and data mining.

Smaine Mazouzi

He is an associate professor at 20 Août 1955 University of Skikda. He received his M.S. and Ph.D. degrees in Computer Science from University of Constantine, respectively, in 1996 and 2008. His fields of interest are pattern recognition, machine vision, and computer security. His current research concerns using distributed and complex systems modeled as multi-agent systems in image understanding and intrusion detection.