An MCA Based Method for API Association Extraction for PE Malware Categorization

Mohamed Belaoued and Smaine Mazouzi

***Abstract*—In computer security, protecting systems against malwares has become the main concern of particulars and companies. Unfortunately, the existing anti-malware systems are so fare unable to provide an efficient protection. However, a new generation of powerful malware detection techniques has emerged. One of these techniques is that based on the static analysis of the called API functions by a program in order to detect any suspicious behavior. In this paper we provide a method to extract existing associations between the imported API functions by malware codes under Microsoft Windows environment. The main goal of this work is to be able to determine with a high degree of confidence what the most likely used Windows APIs and their associations by malware are. We have used for that purpose a well known and a powerful statistical method which is the Multiple Correspondence Analysis (MCA). We applied the MCA method on a set of APIs which were priori extracted from a large dataset of malware and clean portable executable (PE) files. According to our knowledge, this is the first work having used factorial analysis to determine API associations in malwares. We assume that this allows a more accurate behavior based malware detection.**

***Index Terms*—Malware, malware analysis, multiple correspondence analysis (MCA), Windows API.**

I. Introduction

Malware (e.g. Trojans, worms, viruses, etc.) are computer programs which are designed to accomplish malicious actions like stealing sensitive information (eg. user's IDs, credit card information, etc.), disturbing network traffic or hiring zombie machines in order to conduct network-based attacks (such as DDoS) that can have serious outcomes. The number of discovered malware is increasing every year, in its 1st quarterly report of 2014, McAfee Corporation has reported that more than 30 million new malware have been discovered during that period, which represents nearby 5 million more discovered malware compared to the 4th quarter of the last year [1]. That proliferation of malware has direct economical impacts on companies which are estimated to billions of dollars of loss every year [2]. Obviously, MS Windows is the most affected by malwares, seeing that it is the most used operating system. Indeed, during 2013 80% of computers run MS Windows operating system [3]. So, having an efficient anti-malware protection that will detect and avoid the threat before it executes its malicious content has become primordial or even vital.

Until now, there is no software that can efficiently protect computers against malware attacks [4]. Techniques used by anti-virus software are signature based; which consists of generating a short and unique string of bytes of the analyzed program called signature, and compare it with existing ones stored in a signatures' database. Such techniques suffer from two main drawbacks: First, they are inefficient against polymorphic malwares that are able to modify themselves [4]. Second, they need to priori know the malware by their signatures, so, they are unable to deal with unknown or new malwares, often developed after discovering a "Zero-day" exploit by hackers. Some efforts have been done in order to improve the signature-based detection systems; it's the case of SAVE [5] which was able to detect polymorphic malware. However, it failed to detect unknown malware.

In order to overcome the limits of the signature-based approaches, a new generation of non signature-based ones has emerged. One of the most efficient ones are the behavior based methods that use the API functions called by a program as a feature for malware detection [6]-[9]. They are based on the analysis of the set of API functions imported by an executable in order to predict its behavior (malicious or legitimate) once it will be run. The decision process consists of comparing the set of extracted APIs to a priori constructed model of suspicious ones. Such techniques are very effective and simple to implement, however, the list of malicious API functions which will be used as a model for the decision process should be selected in a proper way. Otherwise the effectiveness of the system would be affected. In this paper we will focus on that specific point, by proposing a statistical approach for identifying the API functions that are likely used by malware codes, for that purpose we will analyze API calls using a well known and a powerful statistical method which is the multiple correspondences analysis (MCA) [10]. Obtained results allow us to set what are the most likely used Windows APIs by malware PE and that by studying the different existing associations between multiple categorical qualitative data which represents the API functions called by a program.

This paper is organized as follow: Section II is a brief introduction to the most common malware types. In Section III the portable Executable file format is introduced to facilitate the comprehension of the remaining sections. Section IV will focus on the related works in the literature. In Section V we introduce the MCA method and all its related concepts, and then we will introduce the different phases of our analysis. In the remaining of the section we attempt to analyze the obtained results. Section VI will conclude our study and underlines it's perspectives.

II. Malware Types

In this section we will introduce some common malware

Manuscript received July 20, 2014; revised September 22, 2014.

The authors are with the Department of Computer Science, University of Skikda, Algeria (e-mail: belaoued.mohamed@gmail.com, mazouzi_smaine@yahoo.fr).

DOI: 10.7763/IJIEE.2015.V5.534

types which are Viruses, worms and Trojans.

A. Virus

A virus is a self replicate program with harmful intents, it replicates by merging itself into a host file (such as binary executable, files that contain macros, etc.) and once that file is run it'll run the malicious code, a virus can spread through different computers using several ways such as network, corrupted media, etc.

B. Trojan Horses

A Trojan horse usually seems as a casual and a harmless program, but in fact it has a hidden malicious behavior. Once a Trojan is run it'll accomplish the harmless tasks that it was designed for, but it does also malicious actions in the background. Generally the purpose from propagating a Trojans is to perform unauthorized actions such as stealing information or disturbing the performance of computers and/or networks.

C. Worms

A worm is self replicating programs. Unlike virus, worms don't need any host file or user's intervention to spread, according to the way they spread worms can be classified into different categories such as network worms, P2P worms, E-mail worms, etc. worms can perform a various harmful tasks such as disturbing the network traffic, stealing information, etc.

Malware under MS Windows environment are often present as a normal executable files, and like any benign executable they can have any known executable extension file such as: .EXE, .COM, .CPL and are structured according to a standard format called PE for Portable Executable [11]. PE programs interact with the operating system using an Application Programming Interface (API). APIs are a set of functions that the operating system provides to user applications in order to request operating system's services, stored in dedicated executable files, typically DLL ones [12].

III. PE File Structure Overview

The PE (Portable Executable) is a file format for binary executables and DLLs used by the Windows operating systems [13]. A PE file is structured in layers (Fig. 1) and must have at least two sections, one for code and one for data.

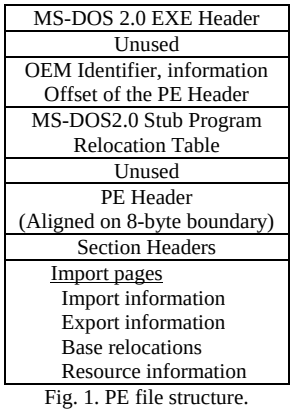

Fig. 1. PE file structure.

The DOS Header is used if the file is run from the DOS. So it can then check whether it is a valid executable or not. The PE header is an IMAGE_NT_HEADERS data structure. The IMAGE_NT_HEADERS structure contains three members which are: signature, File Header, and OptionalHeader. The Optional Header is an IMAGE_OPTIONAL_HEADER structure. One of the members of that structure is the DataDirectory which is an array of 16 IMAGE_DATA_DIRECTORY structures; each one of them is related to an important data structure. One of these structures is the IMAGE_DIRECTORY_ENTRY_IMPORT which contains information about all the imports (DLLs/APIs), the imports entry points to a vector of IMAGE_IMPORT_DESCRIPTOR structures, each structure has a size of 20 bytes and contains information about the DLLs used by the PE file to call APIs. The number of IMAGE_IMPORT_DESCRIPTOR structures is equal to the number of imported DLLs. The field OriginalFirstThunk of the IMAGE_IMPORT_DESCRIPTOR contains an RVA (Relative Virtual Address) which points to the IAT (Imports Address Table), the location of the IAT within the PE file is shown in Fig. 2.

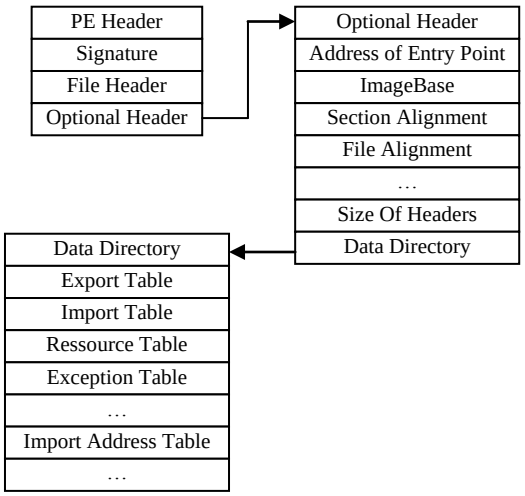

Fig. 2. Location of the import address table.

IAT is an array of functions' pointers that contains elements of IMAGE_THUNK_DATA structures, where every IMAGE_THUNK_DATA structure corresponds to an imported API, and they contain the ordinal of a function or an RVA to an IMAGE_IMPORT_BY_NAME structure. These are the imported functions (APIs) that the code calls, and the purpose of any importation-based malware analysis.

IV. Related Work

Many efforts have been done in order to overcome the limitations of the signature-based malware detection methods. Authors in [14] proposed a system for malware detection and classification which is able to detect polymorphic and zero-day malware by analyzing PE file structural information such as: COFF header, Optional header attributes, those techniques are very effective and have a high detection rate, however their main disadvantage is that the reliability of the used features can be discussed. if we take the example of the field CHECKSUM of the PE optional header, that field was used in by the authors as a characteristic for the detection process, that field represents a CRS checksum of the PE file, it is generally ignored and set to 0 by most of the compilers [15], and that is the mostly observed value in the malware

programs, however the hacker can easily update that field using different tools such as CheckSum Fixer [16] which will recalculate the real value of the CHECKSUM and update it in the PE file. So, in that case the field will contain a value rather than a null value.

Recently many researchers use behavior-based malware's detection methods that use the set of imported APIs in order to indentify a malicious behavior in PE programs. IMDS [9] is one these malware detection systems that are based on a behavior method. IMDS used data mining methods and an OOA mining algorithm in order to generate APIs class association's rule which will be used as pattern by the detection module, authors have experimented their method on a large number of executables and have obtained a satisfying results. However, it was impossible to access the used dataset, due to the industrial nature of the research [17].

Authors in [7] proposed a malware detection system which is based on a model of suspicious behavior which is built by extracting and grouping API functions according to some intuitive scenarios of malicious actions that a malware can accomplish, such as searching for files to infect, writing malicious data into files, etc. the proposed decision process is based on a Bayes algorithm, they obtained a detection rate of 93.71 % and very low false positive and false negative rates. The API selection phase was based on grouping APIs in twelve different scenarios, and we think that is too restrictive because there are many other actions that a malware can accomplish, so the mentioned behaviors didn't reason about a sufficient number of potential malicious behaviors.

In a more recent work [6], authors have proposed a behavior-based malware detection method using a graph matching algorithm, the proposed system is composed of three different phases first the API calls are extraction was done dynamically by monitoring the execution of the programs under a controlled and secured environment, then a data dependant API call graph is constructed in order to be compared those stored in the malware's call graphs database, the decision process is made by calculating the similarities between the analyzed file call graph and the call graphs in the database using the LCS algorithm, they experimented their method on a set of 85 PE files (75 malware programs and 10 benign programs), and they have obtained a detection rate of 98.6% and with a false positive rate of 0%.

There is a growing use of approaches based on the analysis of API functions as features for malware detection and classification, and as mentioned before, it is very necessary to be able to distinguish associations of APIs typically used by malware, from those used by benign applications because the effectiveness of such detection systems relies entirely on that.

V. Proposed Approach

The proposed approach for identifying APIs associations that are likely used by malware codes proceeds is three main steps which are: API extraction, API pretreatment, and decision.

The first step consists of extracting the list of API calls from every PE program of the used dataset. It is done by statically analyzing the IAT. The result of this first step is a full list of all imported API functions.

The second step consists mainly of making the obtained list more practical to analyze by removing duplicated APIs and grouping and sorting the similar ones. The call frequency of every imported API function will be also calculated.

The last step is the decision making, which is basically a triage process of the list of APIs obtained in the previous phase. At the end of that step we will obtain different groups of APIs including those that are likely used by malwares. The decision step is based on a well known and a powerful statistical method which is the Multiple Correspondence Analysis (MCA).

A. Overview of the MCA

The multiple correspondence analysis (MCA) is a statistical multivariable analysis technique that is used to bring out the relationships (dependence or independence) between categorical variables in a given dataset.

The first step to do when conducting an MCA is to identify the variables that will be analyzed, which is done by establishing a questionnaire which is a list of features (questions) that describe the individuals. For a given dataset, the MCA maps each individual and answer into a set of coordinates and distributes them on a Cartesian coordinate system according to sorted principal axes. These axes will condense the information contained in the dataset so that the majority of the information is explained by only few of them. These axes will bring out the variables that better describes the set of individuals.

Usually, the interpretation of the MCA starts by analyzing the obtained plots in order to describe the principal axes by identifying what they represent. The next step is to identify which are the variables that most contribute to the inertia of the axes. The plot will also illustrate several groups of correlated variables.

B. Data analysis, Results and Discussion

1) Data collection and preprocessing

Our population consists of 425 PE files. 110 of them are malware PE files (Worms) and 315 are clean ones. The clean files were downloaded from Softpedia.com and collected from Windows system files of a clean installation of windows XP. The malware sub-population was obtained from the websites Vxheavens.com.Malware authors generally pack their malware in order to obfuscate them and made them more difficult to detect [18]. We made sure that our population does not contain any packed program. All the executable files were selected after being analyzed by four packers' detectors tools which are: PEStudio V7.935, PEiD V0.95, ExeInfoPe V0.0.3.4 and ProtectionID V0.6.4.0. The cited tools are able to detect a large variety of packers including the most popular ones like: UPX, PECompact, ASPack, etc. All the samples were also scanned by more than 40 different antivirus (AV) software from the website VirusTotal.com. Every used infected PE file was identified as a malware by more than 30 AV.

From our population we've selected randomly 150 clean PE programs and 50 malicious ones. We have limited our study to one type of malware which are worms. We assume that analyzing one specific malware category would be more beneficial. Like with any statistical study, limiting the studied population to a specific category of individuals allow to have more representative results.

2) APIs extraction and pre-processing

To be able to extract API functions and calculate their call frequencies, we developed a module written in Python. The extraction method used by our module is based on a static analysis of the IAT (Import Address Table). We used a third party Python module called Pefile, which is a multi-platform module to read and work with PE files and extract different information from them [19]. After extracting the APIs from malware PE files, we have obtained 5639 imported APIs. So our module proceeds first in removing duplicated APIs in the same PE file and then it will group them by their API names and calculates their call frequencies. Finally, the module will check for the corresponding call frequency in benign programs for every imported API function by malware. At the end of the pre-processing we have got 801 entries of imported API functions by infected PE. The Table I shows an overview of the obtained list.

TABLE I: OVERVIEW OF THE OBTAINED LIST OF API CALLS

N °	API Name	Call Frequencies	
		Worm(50)	Benign(150)
1	ExitProcess	48(96%)	74(49%)
2	GetModuleFileNameA	46(92%)	64(42%)
3	CloseHandle	46(92%)	114(76%)
4	GetModuleHandleA	44(88%)	117(78%)
5	WriteFile	43(86%)	100(66%)
6	GetProcAddress	42(84%)	108(72%)
7	GetStartupInfoA	41(82%)	79(53%)
8	CreateFileA	40(80%)	58(39%)
9	GetCommandLineA	38(76%)	56(37%)
10	RtlUnwind	38(76%)	68(45%)
11	RegCloseKey	38(76%)	101(67%)
12	VirtualAlloc	38(76%)	64(42%)
13	ReadFile	37(74%)	80(53%)
14	VirtualFree	37(74%)	63(42%)
15	GetLastError	37(74%)	115(77%)
…	…	…	…
797	DrawTextExA	1(2%)	9(6%)
798	CopyAcceleratorTableA	1(2%)	8(5%)
799	WnetOpenEnumW	1(2%)	0(0%)
800	CopyRect	1(2%)	27(18%)
801	ShGetFolderPathA	1(2%)	0(0%)

The API functions listed in Table I represent an overview of the imported APIs by Worms, with their corresponding call frequencies (in worms and in benign files). By analyzing the obtained call frequencies we can see that there are some APIs that have a higher call frequency in malware PE and other APIs that have a higher frequency in benign PE such as GetLastError and also APIs that are imported with close frequencies between both malware and benign programs like the API RegCloseKey. So in that case the main problem is how to be able to identify and separate between the malicious and the non malicious APIs.

In this work and starting from the list in Table I we will indentify the most likely used APIs by malware (worms), and their associations. We use Multiple Correspondence analysis (MCA) in order to identify different API associations (groups) including those that represent the most likely used APIs by worms. We will consider the APIs that have been imported by at least 30% of malware PE, assuming that the most likely used APIs by malware would be imported by a higher number of malware PE. So, as a result we have a final list of 89 APIs and they will be the subject of our study. For practicality purposes we will abbreviate the names of the APIs, the complete list of APIs with their abbreviations can be found in the Table VI of the appendix at the end of this paper.

3) Identifying MCA variables

The set of individuals is composed of 200 elements which represents the whole PE dataset (50 malware PE and 150 Benign PE). We used 90 variables, 89 are variables to explain which represents the list of APIs listed in the Table VI of the appendix where every variable has two modalities 'Y' and 'N' (y: yes, n: no). 'Y' means that the corresponding API function was imported by the PE file, and 'N' means that the API function was not imported. The last variable which is PE is considered as an explanatory variable and has two modalities 'Worm', and 'Ben' (for benign) which represent the type of the PE file. All the data has to be represented in a general form which is the gross data table (a matrix of Individuals x Variables), what the next section will describe.

4) Generating the gross data table

The gross data table (Table II) is composed of 90 columns which represent the variables (PE + API functions) and 200 lines which represent the individuals (Malware and benign PE files).

TABLE II: OVERVIEW OF THE GROSS DATA TABLE

N °	PE	V1	V2	V3	…	V87	V88	V89
1	worm	n	n	y	…	n	n	n
2	worm	n	y	y	…	n	y	n
3	worm	n	y	y	…	y	n	n
…	…	…	…	…	…	…	…	…
198	Ben	n	n	n	…	n	n	n
199	Ben	n	n	n	…	n	n	n
200	Ben	n	n	n	…	n	n	n

The gross data table is generated automatically from the list of the APIs presented in the appendix, using a python script that will check for those APIs into every PE file of the malware and the benign sub sets.

In order to perform the MCA method, we have used STATISTICA v7. STATISTICA is a comprehensive and integrated system for statistical data analysis, charting, managing databases, and custom application development, offering a wide range of basic and advanced procedures in science, data mining, the business and industrial applications [20].

Using STATISTICA, we import the text file containing the generated gross data table, after that the software will perform the analysis. The obtained results and their interpretation will be presented in the remaining of the current section.

5) Results interpretation

As mentioned previously, the MCA method aims to reduce and condensate the information in few axes (dimensions). So, the first step to do when we start the interpretation of the results is to determine how many axes reflect the majority of the information, and that is generally done by analyzing the contribution to inertia of the different dimensions. Table III

shows the different eigenvalues of the analysis for the five first dimensions. As we can see, the first two dimensions have respectively 36.65% and 15.87 % of inertia which represents 52.52% of the total inertia, which can be considered as a good value for result interpretation. We can also note that there is a significant jump of eigenvalues just after the second dimension. So, we will restrict our analysis to the first two dimensions which will be represented graphically by the axis 1(horizontal axis) and the axis 2 (vertical axis).

TABLE III: EIGENVALUES OF THE ANALYSIS

Dim	Singular	Eigen-Val	% Inertia	%Cumul	Chi ²
1	0.61	0.37	36.65	36.65	97888.31
2	0.40	0.16	15.87	52.53	42396.45
3	0.24	0.06	5.53	58.06	14775.40
4	0.20	0.04	3.82	61.88	10197.10
5	0.17	0.03	3.02	64.89	8056.30

Now we introduce what each of the two axes represents by analyzing the obtained plot which represents the mapping of the variables modalities as shown in Fig. 3.

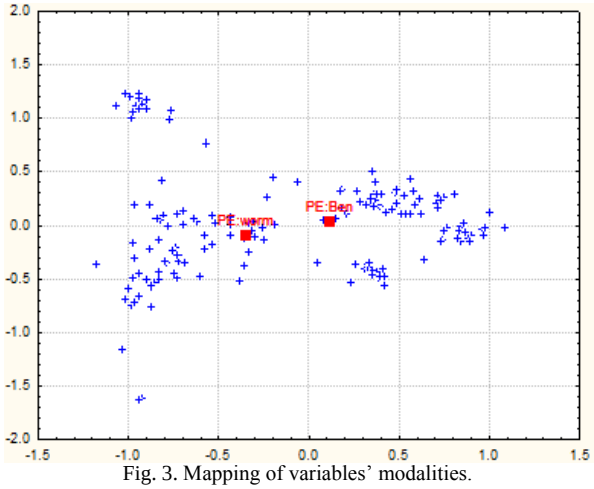

Fig. 3. Mapping of variables' modalities.

From the above plot, we can clearly see that the axis 1 separates between the two modalities of the variable 'PE' (marked by a red square). So we have in the negative side of the axis the modality 'worm' and in the positive side of the axis the 'Ben' modality. We can conclude that the axis 1 separates between APIs that are likely used by worms and those that are likely used by clean programs.

TABLE IV: CONTRIBUTION TO THE INERTIA OF THE AXIS 2

Modality	Coordin.1	Coordin.2	Inertia Axis2
V76:y	-0.92293	-1.61795	0.031499
V83:y	-0.93459	-1.63779	0.030378
V70:y	-0.93511	1.19000	0.028065
V73:y	-0.89892	1.17435	0.027820
V69:y	-0.98696	1.19476	0.026775
V64:y	-0.93763	1.22767	0.026670
V74:y	-1.01468	1.22550	0.026045
V66:y	-1.00535	1.20722	0.025789
V63:y	-0.93484	1.08178	0.025678
V71:y	-0.89742	1.09042	0.025669

Unlike for the axis 1 the plot in Fig. 3 doesn't allow us to graphically identify what represents the axis 2. So, we will try to determine what it represents by analyzing the APIs that most contribute to the inertia of that axis. Table IV shows the first 10 APIs that have the highest contribution to the inertia of the axis 2.

From the 10 variables listed in Table IV we have two variables 'V76' and 'V83' with the modality '**Y'** in the negative side of the axis 2. These two variables have the highest contribution to the inertia and they represent respectively the APIs: GetKeyboardType and SysFreeString. In the positive side of the axis 2 we have 8 variables with the modality '**Y'** which represent respectively the APIs: FreeEnvironmentst, GetEnvironmentStrig, Lcmapstringw, FreeEnvironments, GetStringTypea, LcmapStringA, SetHandleCount, and HeapCreate.

So we can conclude that the axis 2 separates two sets of API functions that are not specific to malware/benign feature. It concerns some predefined associations of APIs, written by programmers of generated by compilers.

After showing what represents each axis, we will try to identify the different groups of associated APIs. So, APIs that are projected close to each axis are correlated which means that correlated according to what represents each part of the axis. First we will attempt to identify these associations visually according to the plot in Fig. 3. The Obtained groups are shown in Fig. 4. It consists in 4 different groups of API functions (G1 to G4).

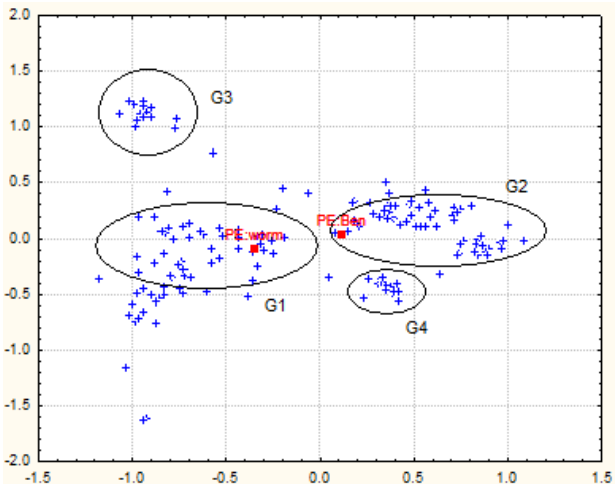

Fig. 4. Groups of associated APIs.

The group G1, which is located on the negative side of the axis 1 (as in Fig. 5) contains 42 APIs; all of them have the modality 'Y'. So, we can conclude that these APIs are those likely used by worms.

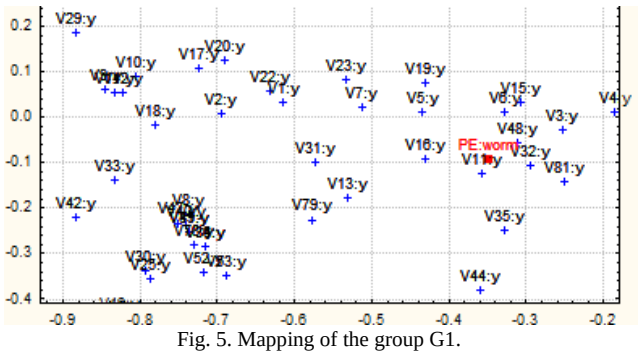

Fig. 5. Mapping of the group G1.

These APIs are important for our analysis as they contribute to 44% of the total inertia (both axes). However, it is possible to reduce the number of these APIs by identifying the most representative ones. That is done generally by analyzing the inter-correlation expressed by the value of the

Cos ²of the angle between the modality and the axis. In our case we will set a threshold for Cos ²set to 0.4 so, we will take into account only APIs that have Cos ²value greater than 0.4. For the first group G1 of 41 APIs, we have obtained 20 APIs that have Cos ²≥0.4. These APIs are considered as the most likely used APIs by worms. The Table V shows the selected APIs.

TABLE V: LIST OF SELECTED APIS FROM THE GROUP G1

N °	API Name	N °	API Name
1	CreateFileA	11	GetCommandLineA
2	SetFilePointer	12	GetModuleFilenameA
3	WriteFile	13	ExitProcess
4	VirtualAlloc	14	GetVersionExA
5	VirtualFree	15	GetStdHandle
6	Rtlunwind	16	LoadLibraryA
7	EnterCriticalSection	17	GetAcp
8	InitializeCriticalSection	18	TlsSetValue
9	LeaveCriticalSection	19	MultybyteTowideChar
10	DeleteCriticalSection	20	WideChartoMultibyte

The second group of APIs (G2) contains 59 APIs that are associated to the modality 'Ben' of the variable PE as they are plotted in the positive side of the axis as shown in Fig. 6.

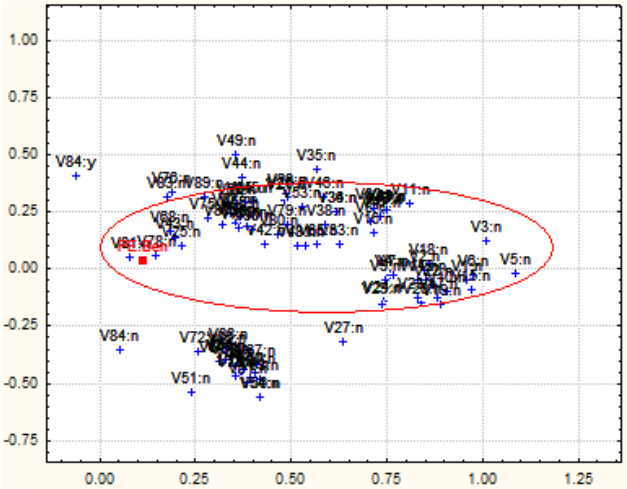

Fig. 6. Mapping of the group 2.

All these variables have the modality 'N' which means that they represent APIs that were not imported by the benign programs and that might mean that these APIs are generally avoided by software developers. As it was done for the APIs of the group G1 we will select the most representative ones by selecting the APIs that have a value of Cos ²greater than 0.4. In that case we have obtained 26 APIs, 19 of them represent the same APIs of the group G1 but with the modality 'N'. The remaining 7 APIs are: Comparestringa, Getcpinfo, Getlocaleinfoa, Messageboxa, RaiseException, Regopenkeyexa and Setendoffile. Note that these APIs have the modality 'N'.

The two other groups (G3 and G4) are not specific to malwares nor to benign programs. These APIs might represent some preferences of development of Win32 applications. It is the case of the APIs of the group G3 that contains APIs with the modality 'Y' which are: Getstringtypew, Sethandlecount, Freeenvironments, Lcmapstringa, Getenvironmentstrings, Getoemcp, Lcmapstringw, Freeenvironmentst, Heapcreate…etc.

In our study we will focus on API associations that are specific to malware programs (worms). In the next sub-section we will try to predict some possible behaviors according to the obtained API list from the Table V.

6) *Results discussion*

According to the results obtained in the previous section (APIs of the group G1) there are some possible and predictable actions that worms can accomplish such as:

- **Creation of a new process:** when it is launched a malware creates at least one resident process. It is necessary to control the infected computer, and accomplish the tasks for which the malware was created for. To do so, it is necessary to use some APIs like: GetCommandLineA, to specify which process file will be created and creates a copy of the malware code. CreateFileA, WriteFileA are used to create a file and write the malicious data into. GetModuleFileName can be used to create an instance of the malicious process.
- **Duplication:** worms can self duplicates (creating copies of themselves) in order to infect more computers and it is a typical case of mass mailers. For that purpose worms can use file-based APIs, which we have obtained in our results. For instance: CreateFileA, WriteFile and SetFilePointer.
- **Internal Tasks of the malicious code:** the malicious process must handle some data, which can be internal or external. For this, it uses some less common memory APIs like: VirtualAlloc, VirtualFree. The latter APIs, enable to share the memory between other running processes. This situation is typical in malware behavior, where one of their goals is to cross exchange data with other running applications in order to get non authorized user data. Malware can use critical section objects in order to protect the malicious code from being used by another thread and that might justify the usage of the following APIs: EnterCriticalSection, InitializeCriticalSection, LeaveCriticalSection, DeleteCriticalSection.

VI. CONCLUSION AND FUTURE WORKS

In this work we have introduced a statistical study based on the MCA method to determine what are the most maliciously used windows APIs by PE malwares. The obtained results were very satisfying and we were able to identify what are the several APIs associations that are mostly used by malware and we classified them according to predicted behavior of the malware. Such a work is very useful in building anti-malware tools, especially behavioral based ones.

In future works, we will try in a first part to generalize our study by using different types of malwares and try to determine the most used APIs clustered by malware's type. In a second part, we project to build our own malware detection systems based on the obtained results.

APPENDIX

TABLE VI: LIST OF SELECTED APIS

abv	API Name	abv	API Name
V1	ExitProcess	V46	RaiseException
V2	GetModuleFilenameA	V47	deletecriticalsection
V3	CloseHandle	V48	gettickcount
V4	GetmoduleHandlea	V49	LocalAlloc

V5	WriteFile	V50	GetCurrentProcess
V6	GetProcAddress	V51	TerminateProcess
V7	GetstartupInfoa	V52	GetFileSize
V8	CreateFileA	V53	FindClose
V9	GetCommandLineA	V54	HeapAlloc
V10	Rtlunwind	V55	GetLocalTime
V11	RegCloseKey	V56	HeapFree
V12	Virtualalloc	V57	Getfileattributesa
V13	Readfile	V58	Getenvironmentstrings
V14	Virtualfree	V59	Regcreatekeyexa
V15	Getlasterror	V60	lstrlena
V16	Sleep	V61	Heaprealloc
V17	Getstdhandle	V62	Getstringtypew
V18	Setfilepointer	V63	Sethandlecount
V19	Unhandledexceptionfilter	V64	Freeenvironments
V20	Loadlibrarya	V65	Getlocaleinfoa
V21	Regqueryvalueexa	V66	Lcmapstringa
V22	Widechartomultibyte	V67	GetOemCp
V23	Multibytetowidechar	V68	Setfileattributesa
V24	Getcpinfo	V69	Lcmapstringw
V25	Copyfilea	V70	Freeenvironmentst
V26	RegOpenKeyExa	V71	Heapcreate
V27	GetFileType	V72	Heapdestroy
V28	MessageBoxA	V73	Getenvironmentstri
V29	Getacp	V74	Getstringtypea
V30	Regsetvalueexa	V75	Findwindowa
V31	Createthread	V76	GetKeyboardType
V32	Getcurrentthreadid	V77	VirtualQuery
V33	GetVersionExA	V78	CreateMutexA
V34	TlsGetValue	V79	WaitForSingleObject
V35	FreeLibrary	V80	CompareStringA
V36	TlsSetValue	V81	GetSystemTime
V37	DeleteFileA	V82	DispatchMessageA
V38	SetEndOfFile	V83	SysFreeString
V39	LeaveCriticalSection	V84	GetThreadLocale
V40	InitializeCriticalSection	V85	FlushFileBuffers
V41	EnterCriticalSection	V86	CreateWindowExA
V42	Getversion	V87	SetStdHandle
V43	GetSystemDirectoryA	V88	CharNextA
V44	LocalFree	V89	PostMessageA
V45	FindFirstFileA		

REFERENCES

[1] The McAfee Global Threat Intelligence. (2014). Threats Report: First Quarter 2014. [Online]. Available: http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q1-2014.pdf

[2] MacAfee Centre for strategic and international studies. (2013). The Economic Impact of cybercrime and cyber espionage. [Online]. Available: http://www.mcafee.com/sg/resources/reports/rp-economic-impact-cybercrime.pdf.

[3] OS Platform Statistics. W3School. [Online]. Available: http://www.w3schools.com/browsers/ browsers_os.asp

[4] M. Sikorski and A. Honig, *Practical Malware Analysis, the Hands on Guide to Dissecting Malicious Software*, No Starch Press, 2012. ch. 1, p. 10.

[5] A. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static Analyzer for Vicious Executables (SAVE)," in *Proc. 20th Annual Computer Security Applications Conference*, 2004, pp. 326-334.

[6] A. Elhadi, M. Maarof, and B. Barry, "Improving the detection of malware behaviour using simplified data dependent API call graph," *International Journal of Security and Its Applications*, vol. 7, no. 5, pp. 29-42, 2013.

[7] C. Wang, J. Pang, R. Zhao, and X. Lui, "Using API sequence and bayes algorithme to detecte suspicious behavior," in *Proc. International Conference on Communication Software and Network*, 2009, pp. 544-548.

[8] K. Han, I. Kim, and E. G. Im, "Malware classification methods using API sequence characteristics," in *Proc. International Conference on IT Convergence and Security*, 2011, pp. 613-626.

[9] Y. Ye, D. Wang, T. Li, and D. Ye, "An intelligent PE-malware detection system base on association mining," *J. Comput Virol*, vol. 4, pp. 323-334, 2008.

[10] H. Abidi and D. Valentin, "Multiple correspondence analysis," *Encyclopedia of Measurement and Statistics*, California, 2007, p. 13.

[11] Goppit, "Portable executable file format – A reverse engineer view," *Code Breakers Magazine*, vol. 1, no. 2, p. 4, 2006.

[12] E. Eilam, *Reversing: Secrets of Reverse Engineering*, Wiley Publishing Inc, 2005, ch. 3, p. 88.

[13] Microsoft. (Feb 6, 2013). Microsoft Portable Executable and Common Object File Format Specification. R8.3. [Online]. Available: http://msdn.microsoft.com

[14] S. Shah, H. Jani, S. Shetty, and K. Bhowmick, "Virus detection using artificial neural networks," *International Journal of Computer Applications* (0975-8887), vol. 84, no. 5, pp. 17-23, December 2013.

[15] M. Pietrek. (1994). Microsoft developer network, peering inside the PE —A tour of the Win32 portable executable file format. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms809762.aspx

[16] CheckSum Fixer. [Online]. Available: http://www.woodmann.com/collaborative/tools/index.php/CheckSum_Fixer

[17] A. Sami *et al*., "Malware detection based on mining API calls," in *Proc. 2010 ACM Symposium on Applied Computing, SAC'10*, March 22-26, 2010, pp. 1020-1025.

[18] R. Arora, A. Singh, H. Pareek, and U. R. Edara, "AHeuristics-based static analysis approach for detecting packed PE binaries," *International Journal of Security and Its Applications*, vol. 7, no. 5, pp. 257-268, 2013.

[19] Google code, pefile. [Online]. Available: http://code.google.com/p/pefile/

[20] StatSoft. (2014). Statistica Feartures Overview. [Online]. Available: http://www.statsoft.com/Products/STATISTICA-Features.

Mohamed Belaoued received his B.A. degree and M.A. degree both in computer science from the University of Skikda Algeria in 2009 and 2011 respectively. He is now a PhD student advised by Dr. Smaine Mazouzi in the Department of Computer Science of the same university. His main research interests include reverse engineering, malware analysis and data mining.

Smaine Mazouzi is an associate professor in University of Skikda. He received his MS 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.