An Efficient Approach for Detecting Malware Using API Call Mining

Research · January 2020

DOI: 10.13140/RG.2.2.31309.87528

4 authors, including:

Thanudas B.
Indian Institute of Space Science and Technology
13 PUBLICATIONS **40** CITATIONS
SEE PROFILE

S. Sreelal
Indian Space Research Organization
39 PUBLICATIONS **97** CITATIONS
SEE PROFILE

Sourav Maji
Indian Institute of Space Science and Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

International Journal of Advanced Science and Technology
Vol. 29, No. 4s, (2020), pp. 2254 – 2274

An Efficient Approach for Detecting Malware Using API Call Mining

B. Thanudas[≠] S. Sreelal[¥], V. Cyril Raj[≠], and Sourav Maji[ǂ]

[≠]*Dr. M. G. R. Educational and Research Institute, Chennai 600095, India*
[¥]*Vikram Sarabhai Space Centre, ISRO, Thiruvananthapuram 695022, India*
[ǂ] *U R Rao Satellite Centre, ISRO, Bengaluru 560017, India*

Abstract

Advanced malware remain as a major challenge in enforcing security of the enterprise networks. Since most of the commercial tools use static analysis for malware detection and prevention, they are unable to detect unknown malware and sophisticated malware such as polymorphic and metamorphic malware. Dynamic approaches that consider the real-time as well as run-time behaviour of the malware are very essential. We devised two methodologies of dynamic analysis by making use of the Application Programming Interface (API) call features to detect malware: *(i) Application Programming Interface Call Frequency Mining (API-CFM)* and *(ii) Application Programming Interface Call Transition Matrix Mining (API-CTMM)*. Our analysis shows that API-CFM and API-CTMM provide improved accuracy in malware detection. In our techniques, the API usage of a set of advanced malware and benign programs are learned/characterised using supervised classification algorithms: *random forest*, *adaboost*, *support vector machine*, and *naive bayes*. We used 94 API calls for the malware detection methodology. For the API call transition based mining techniques, we use a feature vector of dimension 94 × 94. We also engage the Principal Component Analysis (PCA) of the selected features to reduce the time complexity in malware detection. Our test results show that API-CFM technique gives an accuracy of 76.19% and API- CTMM technique gives an improved accuracy of 95.23%.

Index Terms—enterprise network security, malware, polymorphic, metamorphic, botnet, dynamic analysis, clustering, classification, Application Programming Interface (API), portable executable (PE), call frequency, call transition, data mining, machine learning.

I. INTRODUCTION

An enterprise network, not supported with the state-of-the-art security protection system, can fall victim to a variety of attacks that result into stealing of data or credentials, phishing, data manipulation, etc. The attacks can also inject trojan horses, spyware, self-propagating worms, or bots and can exploit vulnerabilities of the enterprise network. Some of the attacks can slow down the network speed and even potentially cause severe damage to the network systems and to the computer systems.

Malware is a malicious software designed to cause damage to a single computer, server, or enterprise computer network [1]. Very advanced, intelligent, data mining and machine learning methodologies are required for malware detection and prevention. In general, analytical methodologies of malware detection, are of two types: *i) Static*, and *ii) Dynamic*. Normally, in the static method, the binary executable file of the malware is disassembled into corresponding assembly code and analyzed for the presence of malicious code. Signature based analysis is an example of the static method. However, due to the uncertainty in firming up the signature, static method is not able to identify unknown malware as well as metamorphic and polymorphic malware programs [2] [3] [4] [5].

As the malware programs gain more capability and getting evolved as Highly Sophisticated Computer Malware (HSCM), their detection has become a very challenging process with the

ISSN: 2005-4238 IJAST

existing tools. HSCMs use a number of techniques to avoid detection. Such malware have ability to self-modify to avoid detection such as changing the process name over the run-time and mutating their process to behave similar to a genuine process [6]. Many malware have a defined behavioural pattern which help in their detection; however, sophisticated malware make the detection difficult.

In this paper, we propose a dynamic framework for analysing and classifying the Portable Executable (PE) files into malicious and benign files based on API call mining techniques. Our framework is based on the idea that any program, malicious or benign, in order to access or perform any operation on the operating system, needs to make certain sequence of calls to windows Application Programming Interface (API). For example, keyloggers [7] point to the memory address for the keywords typed. In order to do so, they make memory management calls to access the memory address. Then, the malware communicates with the attacker by making network calls to perform the data transfer operations. In fact, all application programs make API calls, in order to perform their operations. However, most malware are codes which are destined to execute a series of malicious operations in the victim machine. Therefore, a malicious program, for example, is likely to perform more network access or memory access calls compared to a benign program. In the case of rootkits [8], they need to trace the activity of memory processes, in order to steal important information stored in different memory locations. Therefore, a malware is likely to show increase in frequency of API calls compared to the benign program. We propose a new API call mining approach by tracing the calls made by malware PE files, creating log files of API call sequence, and extracting the features from the log files for differentiating malware and benign files.

We tested our proposed API call mining-based malware detection solution over a large number of malicious files, comprising of viruses, spyware, and trojans and also on large number of benign files that include windows system files and many windows portable tools. Our collection of malware used for analysis were gathered from various researchers and the ingenuity of the data was tested using virussign [9].

For the dynamic methodologies of analysis, the suspected malware PE is allowed to run in an isolated testbed that is well managed and controlled by the user. Provision for monitoring the functioning of the PE is enabled in the testbed. Further, the distinct run-time behaviour of the process is classified to the level of distinguishing whether it is a malware PE or benign PE. Therefore, dynamic analysis overcomes the drawback of static analysis. However, the dynamic behaviour of malware makes the detection a difficult task. In this paper, we propose two data mining techniques to detect malware using API calls.

We consider API calls important, as they provide abstract behaviour of malware executable. Malware programs are allowed to execute and features are extracted from the API call logs. Unlike benign application programs, malware programs make more network and memory accesses. Therefore, we use the frequency of API system calls made by the programs, and the associated characteristics, to classify whether it is benign or malware. For analysis, we consider 94 API calls and use Principal Component Analysis (PCA) to reduce feature vector space, thereby, reducing complexity of the detection process. Our detection process is enabled on all critical computer systems. These systems have a pre-installed module, which creates SQLite [10] database of all PEs with file name, file size, and date modified. The main concept behind this technique is that malware makes changes in the system files whenever it gets infected on our system. Our module scans these PE files in periodic intervals to detect any anomalies in size and date of modification. The module also brings out the new files found in the system. All the changed files and new files are fed into our detection engine to check malware infection.

In the paper, we considered API call as a dynamic feature for analysis. First, a virtual environment to monitor malicious programs was developed. All the executions were carried

ISSN: 2005-4238 IJAST

out for 5 minutes with absolute precaution. The virtual engine was kept isolated from the host machine. The help of API Monitoring tool, API Monitor V2 [11] was used to monitor the calls made by the malware process and the log files generated were recorded. Then, data mining techniques were used to analyse and build the description model.

Rest of the paper is organized as follows: Section II explains the prevailing API based detection techniques for malware. Section III elaborates on the techniques developed using API call frequency mining (API-CFM) for detecting malware on Microsoft Windows platform. Section IV describes the techniques developed using API call transition matrix mining (API-CTMM) to detect the malware in Windows. Section V provides the experiment results and performance evaluation of our method. Conclusions and future scope are given in Section VI.

II. RELATED WORK

Malware can be detected by using either static approach or dynamic approach. In the 90s, the main technique used in detection was signature based static approach. Static approach mainly deals with the off-line analysis of the executable binary code. The benefit we derive from the static analysis is the insight about the program without getting its source code, as part of the reverse engineering process [4]. One of the fundamental lacunae of this method is not being capable of detecting highly sophisticated malware. In this section, we discuss some techniques which are closely related to the work done in this paper.

In [5], static approach was employed for the detection of malware, wherein the suspicious binary code was disassembled and the API calls control flow graphs were generated. The off-line generated control flow graphs were compared with the control flow graphs of known malware and the similarity index was computed. If the similarity index fell above the announced threshold, then the code under the observation was stamped as a malware.

The dynamic method is an on-line and real-time analysis wherein the suspicious malicious code is executed in a very controlled environment and its behaviour is monitored and logged. The dynamic approach is coupled with the more intelligent methods such as data mining and machine learning techniques.

The work in paper [12] highlighted dynamic methods of employing classification techniques and clustering different malware groups. They used Windows API calls extracted from the file samples. Intelligent malware detection system was developed by the process of clustering and classification, using cluster-oriented ensemble classifiers. The authors used classifiers of k nearest neighbour (kNN), Support Vector Machine, Decision Tree and Naive Bayes. The results showed that Support Vector Machine was better than the other classifiers, they considered.

In [2], authors proposed a framework consisting of three major parts. First part was a portable executable (PE) analyser for analysis of the executables in terms of the API calls invoked by the PEs, and this was done by running PE file in an isolated environment such as VirtualBox [13] or VMware [14]. The second part dealt with feature generation and feature selection. They sequenced two sets of API calls respectively for malware PEs and normal PEs based on the discriminative behaviour of the files in the context of malware PEs and normal PEs. Every executable was denoted by an array of API calls A_i, where A is the array and i is the index of i^{th} API call such that $A_i = 1$ when the API call was imported by PE, and $A_i = 0$ when the API call was not imported by the PE. The above features were inadequate to constitute a powerful feature set for malware detection; hence, the more frequently called API calls were formed as a set and added to the feature set, to improve the efficiency of detection. Thus, the binary file PE was mapped into an array of frequently called API sequence F_i where F is the array and i is the index such that $F_i = 1$, when the frequently called API set was found used by the PE file and $F_i = 0$, if found not used by the PE file. The final module was for classification, using the

ISSN: 2005-4238 IJAST

features as explained in second module. Their method yielded 99.7% for rate of detection and 98.3% for maintaining accuracy.

The work in [3] introduced a scalable method that used a combination of features namely i) Names of API calls, and ii) Names of API calls together with the arguments used. By running the PE, they formulated a feature set using the above techniques, for classifying further. The main concept behind the work is that PEs of similar nature use identical API call names with matching arguments. They executed the PEs in a virtual environment and monitored the process in terms of API calls, either till completion of the processes or till the timer period of 2 minutes. WINAPIOverride32 [15] tool was used to log the sequence of API calls, corresponding values initialised in the arguments, and resultant return values, if any. The final resultant features were a string list, apart from the feature vector of API names. The PE files for which the feature existed above a threshold value during the process were earmarked. The authors also incorporated an additional method to extract only outstanding features, in order to reduce computational time. For training and testing purposes, the entire data set was split into ten identical parts. Altogether, the classifier was executed ten times, wherein nine parts of data served for training and the balance one part served for testing.

In [16], authors proposed a novel approach in detecting malware. That is, instead of considering global frequency of API calls, they used sequence of API calls in order, as instruction N-grams, known as API call-grams. In this paper, first they took equal number of benign and malware files. The malware sample was checked for packet signature. They disassembled the code and control flow graph (CFG) was generated for each file. Then, they generated the Call-Graph from the CFG to record the API calls made by the executable into an output file. The API call was converted into feature vector, and was converted into API call-grams for n = 1, 2, 3. Then, 10-fold cross-validation algorithm was applied on the normalized comma-separated values (CSV) to classify the malicious and benign executable. By using 3 API call-gram, they achieved an accuracy of 98.1%. A notable observation was that the n-gram analysis did not succeed in detecting the mutating and polymorphic malware.

The work in [17] proposed a malware detection framework by sequence matching. In this framework, first the static characteristics of software was analyzed through windows PE analysis. Then, they attached the run-time analyzer to the software to perform dynamic analysis. The main concept used in this technique is that monitoring and analyzing all kinds of API function calls are not appropriate for detecting malicious behaviours, because there exist many API functions that are unrelated to malicious behaviours. Malicious behaviours of software were represented as forms of automata, and the behaviours were recognized by confirming that one of the behaviour automata was matched with the sequence of API function calls during the run-time. Therefore, after extracting a sequence of API function calls, during program execution, the sequence was analyzed to decide whether it was matched with malicious characteristics.

In [18], authors proposed a sequence matching technique for detecting malware. A program was used to trace the API call sequence in run-time. They observed that even malware in the same class can have quite different call sequence, and malware in different classes can have common call sequences. They considered a total number of 2,727 API calls which were categorized into 26 groups named from A to Z. They analysed the API calls to observe the specific sequence of calls maintained from the 26 groups. Then, they used longest common sub-sequences (LCSs) to extract the common API call sequence pattern among the malware, which can be considered as malware signature. In the analysis process, the extracted API call sequence of a newly inserted program was compared with the API signature in the database. If any LCS was matched with the signature, then the newly added program was classified as malware.

The work in [19] proposed a method in detecting zero day or unknown malware with high

ISSN: 2005-4238 IJAST

accuracy and efficiency based on signature of Windows API calls. The data mining framework employed in this work learned through analysing the behaviour of existing malicious and benign codes in large data-sets. The classifiers, such as Naive Bayes (NB) algorithm, k-Nearest Neighbour (kNN) algorithm, Sequential Minimal Optimization (SMO) algorithm and Neural Network (NN) algorithm were used for training and their performance was evaluated.

Our proposed technique of API call mining based malware detection varies from the existing approach. HSCMs make specific API calls for operating the OS kernel related functions concerned to *process*, *memory* and *network*. We used the new features, characterising the API call transition matrix of the dominant API call sequences. Also, in the API call frequency mining based approach, we added one more important feature *log file size* in the feature set that ideally characterises the run-time behaviour of HSCMs. In the multi-dimensional feature set of API call transition matrix mining based approach, we introduced the reduction in feature dimension using PCA which significantly reduces feature vector size, thereby reducing the time of training and detection processes without compromising the accuracy.

III. MALWARE DETECTION USING API CALL FREQUENCY MINING (API-CFM)

The malware needs to be executed in a controlled environment for monitoring the API call frequency. Virtual system is very ideal for execution and monitoring of the suspicious binary code. We established a virtual machine in such a way that the virtual engine is isolated from the native host machine. Using the tool API Monitor V2 [11], we monitored the API calls made by the malware/benign run-time processes corresponding to the executables and recorded the log files generated by the tool. Upon getting the log files, we used data mining techniques to analyse and create the API call frequency description set. A classification is designed based on the call frequency description, to categorize the Portable Executables (PEs) as either malicious or benign.

The major activity of the malware is the communication with the attacker for exchanging the stolen information of key logs, network changes, and the periodic memory accesses it has undertaken [20]. For this purpose, the malware has to make the additional and periodic API calls of the OS kernel. Our technique traces these API calls and apply data mining methodologies to detect malware among all the PEs, we execute. To reduce the overload and complexity, only the API calls concerned to the Dynamic Link Libraries (DLLs) catering the system functions such as *Process Management*, *Memory Management* and *Network Access and Management* are considered. Hence, we considered the most important six DLLs, viz. *user32.dll, kernel32.dll, ntdll.dll, advapi32.dll, ws2_32.dll* and *wininet.dll*. All API calls pertaining to those DLLs were monitored and their call details were logged. Table I lists the major functions provided by those six DLLs.

The three phases of API call mining based malware detection technique are: i) Data collection, ii) Feature analysis, and iii) Classification as illustrated in Fig. 1.

DLL Name	Functionality
advapi32.dll	Advanced-API services calls related to registry and security

ISSN: 2005-4238 IJAST

kernel32.dll	Kernel level calls for memory access and resource management
ntdll.dll	NT system function calls
user32.dll	User level calls for managing messages, timers, and communications
wininet.dll	Calls for Internet protocols, like FTP and HTTP
ws2_32.dll	Calls for windows socket programming for managing connections

Table I: DLLs and their functions

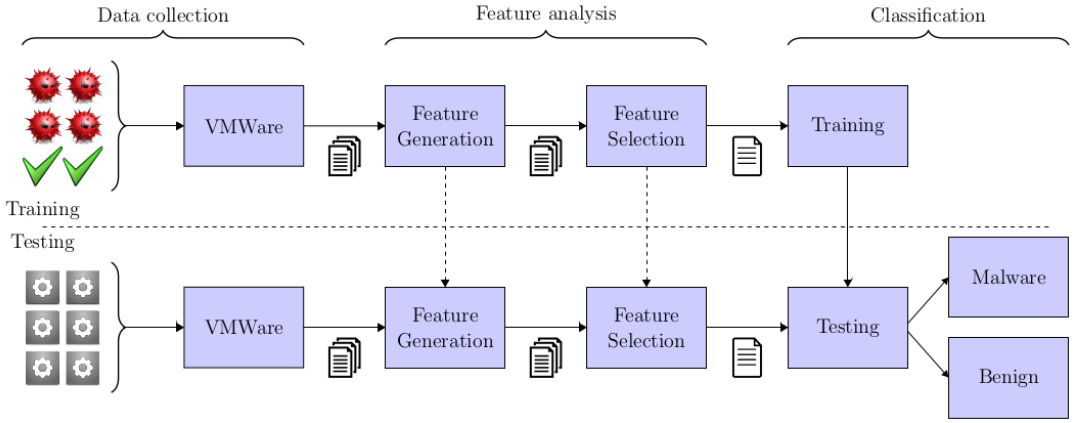

Fig. 1: Phases and functional flows of API call mining based Malware Detection

A. Data Collection

For analysis, malicious programs are downloaded from various domains of researchers' and the sophistication of the malware are verified by scanning the malware files using Virustotal [21] and observing that Virustotal could not detect these files as malware. The malware programs considered include advanced viruses, trojans, worms, and backdoors. For the benign programs, we took the executable programs from system32 module of Microsoft Windows operating system. In order to monitor the behaviour of collected programs, the malware and benign PEs are executed in a virtual environment. In parallel, the API monitoring tool is executed to capture the log files related to API calls.

B. Feature Analysis

In order to derive the characteristics of malware programs, it is required to extract and analyze the required features from the log file created. The process of Feature analysis is carried out in two steps namely: (i) Feature Generation, and (ii) Feature Selection.

1) Feature Generation: The API call log was given as input to the feature generation part of the data mining module. Once the log files are generated, the files that got less than 1000 API calls and files of size less than 1 MB were omitted. Such low sized log files are due to the unsuccessful execution of the corresponding PE files. Some of the executable file formats were not recognised due to the lack of necessary system files required to execute the binaries. Further, some of the files were identified malicious by our virtual environment and were

ISSN: 2005-4238 IJAST

rejected for execution. The features considered for our detection technique were API call frequency and log file size. These features were considered because malware programs are likely to make more API calls to windows kernel in order to do their operations. These repeated periodical calls to perform the above such illegal operations increase the API call frequency made by malware. Also, the greater number of API calls made by the malware make the size of API log file higher than that of any benign software and hence log file size is considered as a feature. Malware read data/information from the memory pointer addresses more than that of benign software, because malware want to steal critical information. Hence, memory pointer address related API calls are also included as features. All these features are combined to get high accuracy in the detection rate. Hence, from the log files, we extracted the frequency of a list of 144 API function calls and recorded the log file size in a feature vector space using a Python script [22]. The feature vector space formats the feature information extracted into a vector which can be easily read by the data mining algorithms. In order to reduce dimensionality and to avoid redundancy, we considered the important features and finally we selected 94 API function calls frequency and log file size, counting to a total of 95 features. Details of feature selected are discussed in the next sub section.

2) Feature Selection: We carried out feature selection based on past behavioural study of malware [23]. Most of the malware, after attacking the victim machine, perform a sequence of tasks according to the contents of its immediate payload. First, they make access to the system directory, bypass the administrator permission to read the desired files, and make changes to the file by writing to the file. Second, the malware make memory access to find the local and global memory location of the file. Then, in order to perform operation on the victim machine, the malware tend to change the system's registry. A set of API functions used for typical malware operations is provided in Table II.

Requirement of malware	**API calls**
Get directory details of the system	GetWindowsDirectory, GetSystemDirectory
For infecting files	FindFirstFile, FindNextFile, FindCLose
For mapping files	CreateFileNapping, MapViewofFile, UnmapViewofFile
Writing onto files	CreateFile, OpenFile, WriteFile, CloseHandle
For changing attributes of files	GetFileAttributes, SetFileAttributes
For altering file-time	GetFileTime, SetFileTime
Allocating Global-Memory	GlobalAlloc, GlobalFree
Allocating Virtual-Memory	VirtualAlloc, VirtualFree
Loading into Register	RegOpenKey, RegCreateKey, RegSetValue, RegCloseKey

Table II: A typical sequence of calls made by the malware

Mutation is used by a malware to change its source image from time to time to evade detection [24]. The API call responsible for the mutation by a malware is ReleaseMutex. Similarly, GetTickCount is a function call which a malware PE tends to call more often than

CharLower	CharNext	CloseHandle	CompareString
CopyFile	CreateFile	CreateFileMapping	DecodePointer

ISSN: 2005-4238 IJAST

EncodePointer	EnterCriticalSection	EventActivityIdControl	EventWriteTransfer
ExitProcess	FindClose	FindFirstFile	FindNextFile
GetCommandLine	GetCurrentProcessId	GetCurrentThreadId	GetFileAttributes
GetFileTime	GetLastError	GetModuleHandle	GetProcAddress
GetStringType	GetSystemDirectory	GetSystemTimeAsFileTime	GetTickCount
GetWindowsDirectory	GlobalAlloc	GlobalFree	HeapAlloc
HeapSetInformation	HeapSize	InterlockedIncrement	IsProcessorFeaturePresent
LCMapString	LdrGetProcedureAddres	LdrLockLoaderLock	LeaveCriticalSection
LoadLibrary	LocalAlloc	LocalFree	Lstrlen
Malloc	MapViewofFile	Memcpy	Memmove
Memset	MultiByteToWideChar	NtAllocateVirtualMemory	NtClose
NtOpenKey	NtProtectVirtualMemory	NtQueryKey	NtQueryValueKey
OpenFile	OpenFileMapping	RegCloseKey	RegCreateKey
RegOpenKey	RegQueryValue	RegSetValue	ReleaseMutex
RtlAcquirePebLock	RtlAllocateHeap	RtlAnsiStringToUnicodeString	RtlEnterCriticalSection
RtlFreeHeap	RtlFreeUnicodeString	RtlInitString	RtlInitUnicodeStringEx
RtlLeaveCriticalSection	RtlNtStatusToDosError	RtlReleasePebLock	RtlRunOnceExecuteOnce
RtlValidateHeap	SetFileAttributes	SetFileTime	SetLastError
SetUnhandledExceptionFilter	StrCmpNICA	TerminateProcess	TlsGetValue
TlsSetValue	UnMapViewofFile	UnhandledExceptionFilter	UrlCanonicalize
VirtualAlloc	VirtualFree	WaitForSingleObject	Wcsrchr
WideCharToMultiByte	WriteFile		

Table III: 94 API calls used for our detection technique

benign programs in order to change the file time. Surveying through the literature in [25], we selected 94 important API calls that are shown in Table III as the features necessary to obtain high accuracy. Combined with log file size, we identified 95 features. Only the nonredundant important API function calls responsible for three functionalities: *process*, *memory* and

ISSN: 2005-4238 IJAST

network management are considered.

C. Clustering

Our malware files did not have any legitimate labels of the category of malware it belonged to, because all the PE binaries provided by sources were not labelled. Clustering comes handy whenever researchers find it difficult to find behavioural similarities between files in feature vector space which do not share labels. Clustering is an unsupervised algorithm widely used in machine learning [12]. Algorithm 1 discusses about the multi-dimensional clustering methodology *K*-Means Clustering, used to cluster the similar family malware into unique cluster, using the distance based clustering of the feature space of the files. Parameter *K* was chosen to describe the number of clusters.

1: Procedure CLUSTERING (X, V)
2: Considering N malware files in dataset with D dimensionality
3: Let X = $x_1, x_2, x_3, \ldots, x_D$ be the set of data points and
V = $v_{i1}, v_{i2}, v_{i3}, \ldots, v_{iD}$ be the set of centres
4: Choose K value as the number of clusters
5: i = 0
6: loop for cluster i = 1 : K do
$d_i = \sqrt{(v_{i1} - x_1)^2 + (v_{i2} - x_2)^2 + \cdots + (v_{iD} - x_D)^2}$
7: d = min (d_i)
8: if d = d_i then cluster the file in i^{th} cluster
9: for cluster i = 1 : K do
Recalculate the cluster centres
10: Mean centre for i^{th} cluster $c_i = \sqrt{(v_{i1})^2 + (v_{i2})^2 + \cdots + (v_{iD})^2}$
11: Update the cluster centres $v_i = \frac{1}{c_i} \sum_{i=1}^{D} x_i$
12: Repeat the process for the next malware file

Algorithm 1: Clustering Algorithm for the PE files

Time Complexity of the clustering Algorithm 1 is O (N × T × D × K), where N is the number of files, T is the number of iterations, K is the number of clusters and D is the dimension of the feature space. Over iterations we identified K as 4.

1) Identifying Malware Clusters: To make our clustering algorithm intelligent, we devised a classification algorithm to detect malware on the basis of the highest feature weight of a file and then formulated a threshold with respect to the highest weight. Our algorithm focussed on the idea that malware make higher frequency of API calls, compared to benign programs. Hence, malware are likely to have higher log file size and higher number of memory address calls. Also, there is a high difference between the average feature weight of a malware cluster to the average feature weight of a benign cluster. Our threshold was identified as *0.4 × highest feature weight* value. All clusters lying above the threshold were marked malicious and those lying below were marked benign. The Malware Classification algorithm is illustrated in Algorithm 2.

1: Procedure CLUSTER CLASSIFICATION (X, V)
2: Considering N malware files in data set of dimensionality D
3: Compute the distance of each file from the origin

ISSN: 2005-4238 IJAST

4: Coordinates for file I = (x_{i1}, y_{i1}, z_{i1}, …, zD_{i1})
5: For i = 1 : N do

$$| d_i | = \sqrt{(x_{i1})^2 + (y_{i1})^2 + \cdots + (zD_{i1})^2}$$

6: Compute the maximum feature weight d = max (di)
7: Compute the threshold = 0.4 × d
8: for i = 1 : K do
for a = 1 to $cluster_i$ do
average weight of cluster i $c_i = \frac{\sum_{a=1}^{size(cluster_i)} {d_a}^2}{size(cluster_i)}$
9: if c_i > threshold then cluster i is malware cluster
else cluster i is a benign cluster

Algorithm 2: Malware Cluster Classification Algorithm

The time complexity of the cluster classification Algorithm 2 is O (N × K).

D. Detection of Malware file

From the clustering of the training dataset, we could observe the fact that the malware of similar behaviour were clustered in a particular cluster. All such clusters were marked malicious because of their higher feature weights. For a suspicious malware new file, Euclidean distance of its feature are computed cluster wise. That is, cluster 1 is considered first and Euclidean distance is calculated between each feature of the suspicious file to the feature's corresponding cluster centre axis projection on the feature plane. As we have 4 clusters, this process is completed for the remaining 3 clusters for the suspicious file. Now, for the new file under malware detection, the distance of the features to our cluster centres are computed and we take the minimum distance. The cluster to which the file is having the minimum distance implies that the file is to be considered as part of that cluster. If the file is classified in a malicious cluster, the file is marked malicious, else it is marked benign. The algorithm to classify a new file into a malware or benign program is illustrated in Algorithm 3.

1: Procedure CLASSIFICATION_AND_DETECTION (X, V)
2: Compute the distance of the feature file from origin
3: Let the features of the file be (x_1, x_2, x_3, …, x_D), and
D is the dimensionality of features
4: Cluster centre of $cluster_i$ is (c_{i1}, c_{i2}, c_{i3}, …, c_{iD})
5: for cluster i = 1 : K do
Distance $| d_i | = \sqrt{(x_1 - c_{i1})^2 + (x_2 - c_{i2})^2 + \cdots + (x_D - c_{iD})^2}$
6: d = min (d_i)
7: if d = d_j then file belongs to cluster j

Algorithm 3: Classification and Detection Algorithm

Time Complexity of the Algorithm 3 is O (N).

IV. MALWARE DETECTION USING API CALL TRANSITION MATRIX MINING (API-CTMM)

For performing the illegal operations, the malware has to invoke the API calls in a particular

ISSN: 2005-4238 IJAST

sequence or order. Therefore, we propose a new method which uses API Call Transition Matrix (ACTM) to analyze the sequence of API calls to capture the finer behavioural features of the malware and build a suitable model. ACTM is a matrix of dimension n×n, where n is the number of APIs, and an element v_{ij} of the matrix denotes the number of times j^{th} API is invoked since the last call to the i^{th} API. Similar to API-CFM, we divide the entire process of malware detection using API-CTMM into the three phases: (i) Data collection, (ii) Feature analysis, and (iii) Classification as shown in Fig. 1.

A. Data Collection

The data collection process is same as that we followed in API-CFM, i.e., the malware PEs are downloaded from researchers' domain and the benign programs are identified from the system32 module.

B. Feature Analysis

The procedure of feature analysis consists of two steps:

1) Feature Generation: By monitoring the 94 API calls identified, an API-CTMM of dimension 94 × 94 is created. Further, the matrix is transformed to a linear one of dimension 1×94^2. The dimension is decreased by removing the columns for which the maximum value is less than a threshold T_{max} (we set threshold $T_{max} = 2$).

First step in this process is to create API call transition matrix for all the files which are to be used for training and testing. Once the matrix data set is created, multiple classifiers are trained and the associated programs are tested.

2) Feature Selection: Similar to API-CFM, we select the features for classification using the same methods and also we set up three cases: (i) we considered 94 important API calls from six selected DLLs as shown in Table III, (ii) we carried out Principal Component Analysis (PCA) for reduction of feature set, and (iii) we selected the most important features from PCA.

By the PCA process, we are effectively applying orthogonal transformation to transform the data of correlated variables into the data of linearly uncorrelated variables. Therefore, use of PCA results in reducing higher dimensional feature data to lower dimension ones with minimum information loss. In addition, PCA is used to compute the variation and brings out strong patterns in a data-set, thereby, making the data easier to explore and visualize.

The algorithmic steps of PCA analysis are:

1: Perform mean centre the data by subtracting mean of the column.
2: Calculate the covariance matrix of the dimensions.
3: Compute the eigen vectors of the covariance matrix.
4: Perform sorting of eigen vectors in the descending order of eigen values.
5: Present the data that is to be tested on to the eigen vectors in order.

Prominent API calls	Description
memcpy	Last occurrence of WC in the C wide string ws is returned as a pointer
wcsrchr	memory is allocated

ISSN: 2005-4238 IJAST

	based on the size from the specified heap
HeapAlloc	allocates amount of requested memory
EnterCriticalSection	ownership of the critical section object - being waited
LeaveCriticalSection	critical section object ownership is getting released
memmove	block copy of no. bytes from the source location to the destination location.
GetProcAddress	From the dynamic link library gets the address of exported function or variable
LdrGetProcedureAddress	address of corresponding function is obtained
RtlFreeHeap	Memory freeing from the allocated heap-RtlAllocateHeap
RtlAllocateHeap	pointer of the allocated memory block is returned

Table IV: API calls identified by PCA

As part of features reduction without compromising the detection accuracy, PCA analysis identified dominant 10 API calls (as shown in Table IV) from the feature set of 94 API calls.

C. Classification

For classifying the malware programs, we used four supervised learning methods and algorithms: (i) *Random Forest classifier*, (ii) *Naive Bayes classifier*, (iii) *Support Vector Machine classifier*, and (iv) *Adaboost classifier*.

1) Random Forest Classifier: Random Forest classifier makes use of the machine learning technique and carry out regression and classification functions. Also, it combines a set of weak models and brings out a strong model. As the initial step, multiple trees are created with the prior sample knowledge from the training set. Upon the completion of training, the input vector corresponding to an object, which needs to be classified, is fed into each tree in the forest. The forest chooses the classification using the result from each of the trees, i.e., the classification with highest number of tree votes.

2) Naive Bayes Classifier: Naive Bayes classifier works based on Bayes theorem and maximum posteriori hypothesis. Bayes theorem computes the probability of occurrence of an event from the probability of another event which has already occurred. In terms of our requirement and method, Naïve Bayes classifier assumes the presence of a particular feature in a class which is unrelated to the presence of any other feature. Also, among all the data mining methods, Naive Bayes algorithm is easy to implement.

3) Support Vector Machine (SVM): Support Vector Machine classifier is defined by a separating hyper-plane. For a set of training data, the classifier produces an optimal hyper-plane that categorizes the samples need which are to be tested. This classifier makes provision to represent the samples as points in space, and allows to map so that the samples of the separate groups are divided using a bigger gap. SVM allows to map new samples into the same space and computes the prediction of the category with respect to the side of the gap, they exist.

4) Adaboost Classifier: Adaptive Boosting (Adaboost) classifier combines the weak learners into a weighted sum that represents the final output of the boosted classifier. Adaboost is also

ISSN: 2005-4238 IJAST

adaptive in nature so that the future weak learners are tweaked in support of those instances misclassified by earlier classifiers.

V. PERFORMANCE ANALYSIS

As mentioned earlier, all the malware considered for our experiment were gathered from various websites and were verified for their undetectability by normal use anti-virus software. All the executions were carried out in MS-Windows virtual environment with Windows Defender [26] running as a process. Among the 198 malware executions, only a minimal number of 5 malware were detected by Windows Defender. Hence, our technique plays a major role in detecting new sophisticated malware.

A. Malware Detection using API-CFM

As highlighted in Section III, all the algorithms were implemented in the testbed for classification using the features of frequency of API calls, and log file size. The details of experiments conducted and results obtained are explained below.

1) Clustering Visualisation and Analysis: To generate clusters based on the similar behavioural pattern of malicious programs, our multi-dimensional clustering Algorithm 1 is used. Our dataset is visualised as shown in Fig. 2, using the cluster classification algorithm described in Algorithm 2.

The X-axis in Fig. 2 shows the cluster number, with a total of 4 clusters. The Y-axis shows the increasing direction of cluster centres. We observe that the first and second clusters have higher cluster centre magnitude relative to the other clusters. This implies that the files in these clusters (Cluster 1 and Cluster 2) are having more features than the other clusters. Hence, any particular file clustered in Cluster 1 or Cluster 2 is likely to have high frequency of API calls related to *process*, *memory* and *network*, and having a larger *log file size*. Hence, we classify Cluster 1 and Cluster 2 as malicious clusters, because the average feature weights of these two clusters lie above the threshold value of *0.4 × highest feature weight*. Cluster 3 and Cluster 4 have cluster centres that are less than the threshold value of *0.4 × highest feature weight*. Thus, these two clusters are classified as benign clusters.

As mentioned earlier, the time complexity of our clustering algorithm is of O(N×T×D×K). With our training dataset of 100 files and each file having feature vectors of dimensionality 95 and the number of clusters as 4, our clustering of malware clusters and identifying the pattern between clusters took 154.5591 minutes, that was experimentally determined using the time function of Python library.

ISSN: 2005-4238 IJAST

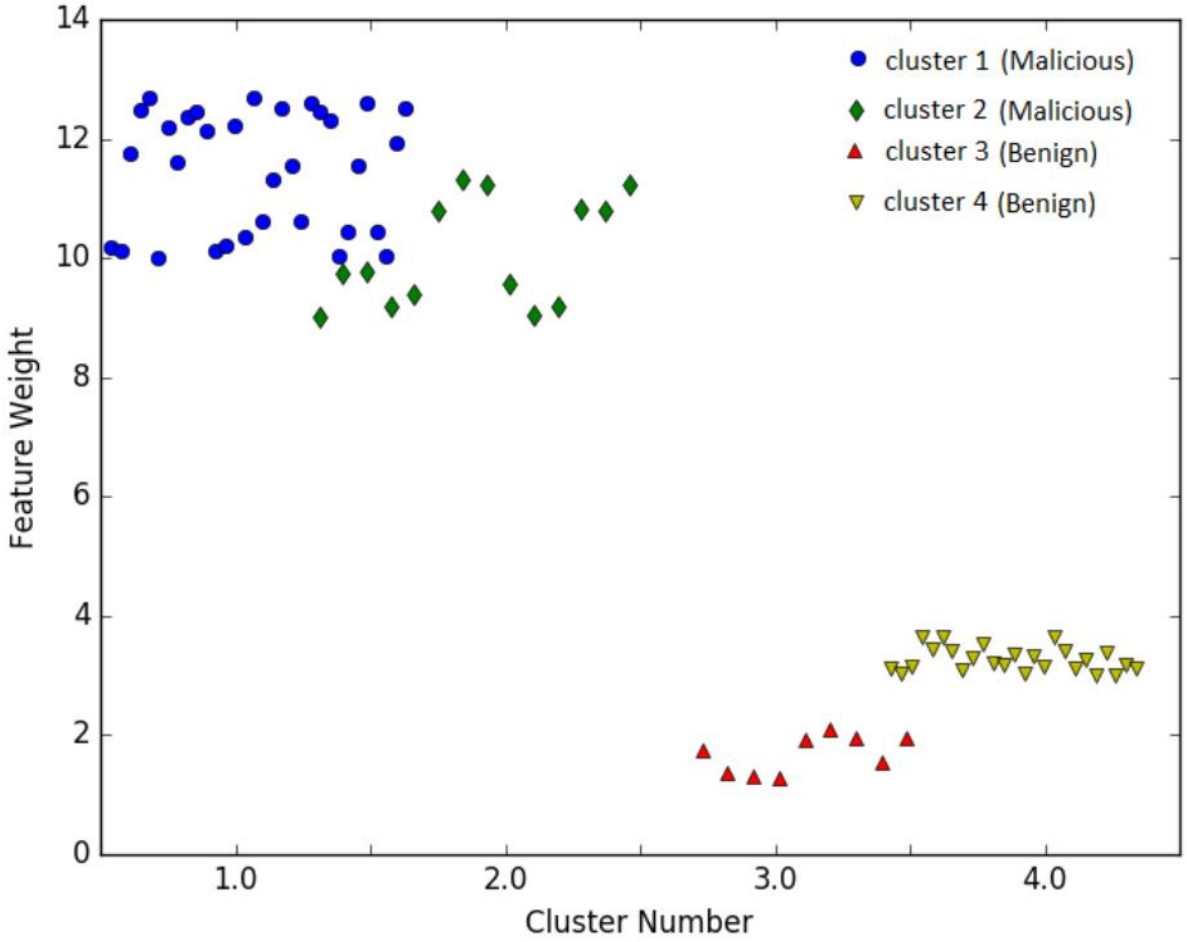

Fig. 2: Visualization of Clustering of Malware and Benign programs

2) Feature Dimensionality and Visualisation: Our technique of API call frequency mining classifies the malware files on the basis of higher cluster weight compared to other files. These identified malicious files were cross checked by virussign [9]. Fig. 3 shows a biplot giving a detailed view of the files and how the files share the dependency of features.

The files sharing similar frequency of API calls and similar features are observed clustered around the respective features. Since the feature space is a high dimensional space, most of the dimensions depicted by the majority of files are observed overlapping. The high clustered Blue region are the clustered malware files that are marked as *virussign* compared to the other scattered benign files with their corresponding names. It may be observed that a malware labelled as *virussign* is clustered among the benign files, due to its premature termination of the execution for want of additional system resources. It is again observed that the malware files are clustered in the direction of increasing feature components. This shows that the malware files have higher feature weights compared to benign programs. Hence, the biplot supports our argument that malware files show high frequency of API function calls compared to benign files.

ISSN: 2005-4238 IJAST

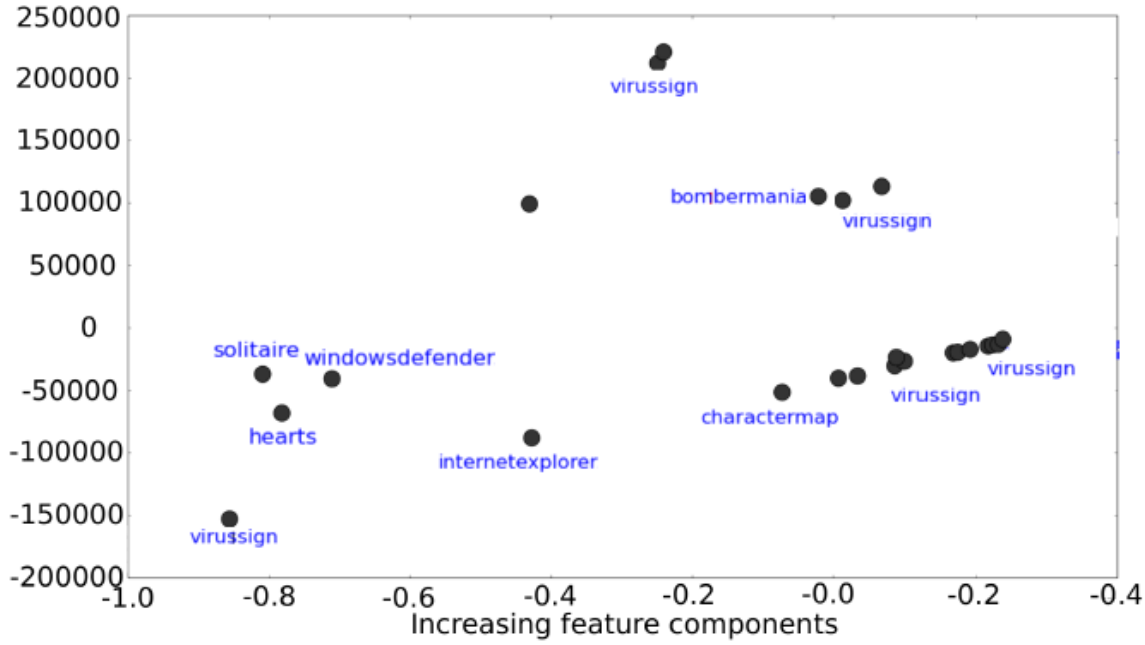

Fig. 3: Malware Files in the Direction of Increased Features

3) Visualisation of Frequency Analysis: To support our argument that malware PE makes more frequent API calls compared to benign PEs, frequency analysis of malware and benign program was carried out. In Fig. 4, the Black bar diagram shows the API call frequency analysed for a famous rootkit named tekdefense [27]. Similar to any other rootkit, the tekdefense malware also makes a lot of memory management API calls in order to steal information from volatile memory of the victim computer system. The frequency analysis shows that the frequency of memory management API calls is very high.

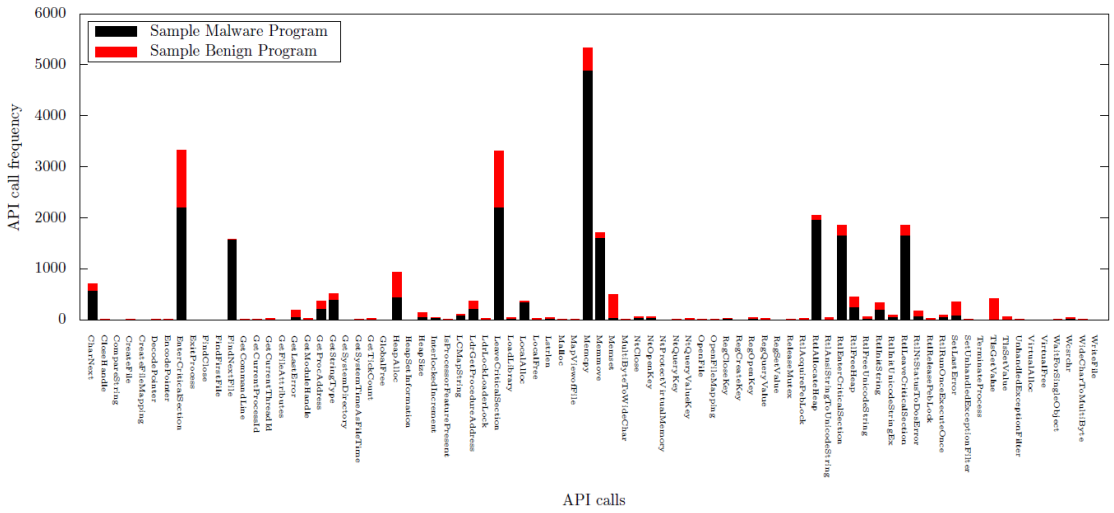

Fig. 4: API call frequency behaviour of sample benign and malware programs.

On the other hand, the Red bar diagram of Fig. 4, depicting the frequency analysis for a benign program hearts Windows game shows that the API calls made are very low in frequency count compared to the malware program. If we see the frequency labels of the two graphs by comparing the Y-axis values, we see that some of the API calls of the malware are more than

ISSN: 2005-4238 IJAST

ten times that of the benign program.

4) Malware Testing Implementation: Similar to the generation of the training dataset, the testing dataset consists of features collected from monitoring of API calls of the files. A set of new malware downloaded from virustotal [21] was used for the data set. The set was mixed with an identical number of benign programs collected randomly from various system32 modules of Microsoft Windows operating system. One by one, the programs were executed and the run-time calls of the API were monitored, for 5 minutes using the API monitoring tool, as mentioned earlier. The frequency of the same API function calls and the log file size for each file were extracted and the output was organized in arrays of features called feature vector space.

Each of the testing datasets corresponding to each file to be tested was directed to a common directory named log testing. The cluster centres calculated during the training of the clusters by the earlier mentioned process of clustering were also directed to this directory to be read by our testing distance metric illustrated in Algorithm 3. The cluster centres calculated were axis projections of multi-dimension space where each dimension was represented by a particular feature. For our 95 features, we obtained a collection of 95 cluster centres corresponding to axis projection of each feature dimension. Hence, our testing logic is made simple. Our testing file also consisted of 95 dimension features represented in a feature vector to be read easily by our algorithm. We calculated the Euclidean distance from each file to every cluster's mean centre. The file was classified to the cluster with whose centre it had the least distance. If the file is classified into a malware cluster, the file is marked with 1 and if it is classified into a benign cluster, it is marked with 0.

5) Accuracy of the technique: After classifying the malware and benign files from our data set, we calculated the various accuracy estimations. Measures of accuracy are True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN).

Accuracy is the probability that a malware file is correctly identified. The accuracy is estimated by the following equation:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Table V shows the accuracy estimated for our malware detection technique.

Malware identified as Malware	**Benign identified as Benign**	**Benign identified as Malware**	**Malware identified as Benign**	
True Positive	True Negative	False Positive	False Negative	**Accuracy**
83.33 %	70.6 %	29.4%	16.7%	77 %

Table V: Accuracy Estimation

The accuracy bar chart depicted in Fig. 5 shows the percentage of the malware and benign PEs detected from the actual total number of malware and benign files in the testing dataset.

ISSN: 2005-4238 IJAST

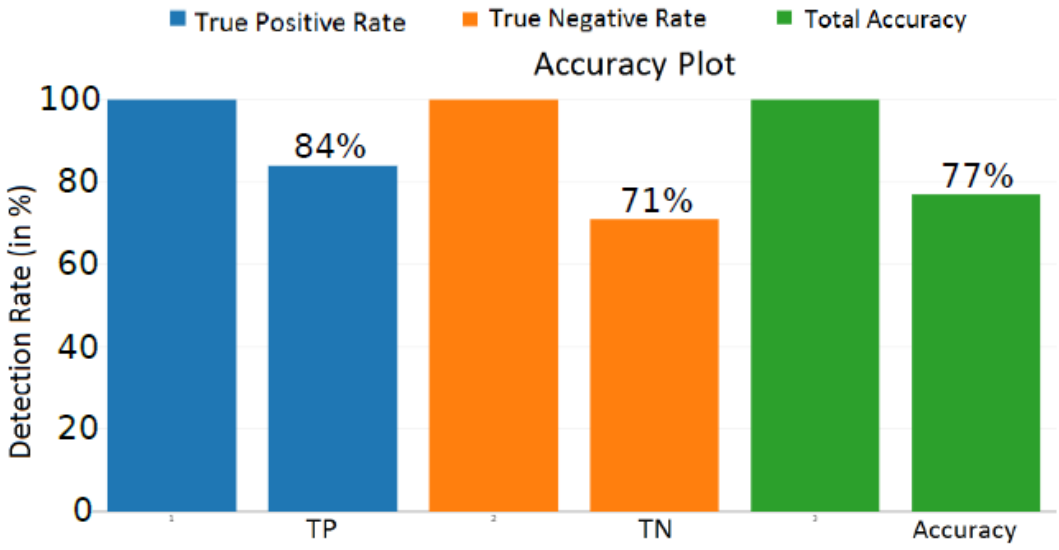

Fig. 5: Accuracy Percentage of Malware and Benign Files Detected

The accuracy estimated took into consideration of the True Positive and True Negative estimations of Equation 1. We obtained an accuracy of 77%. During our file execution in the virtual machine, a number of malware files obtained from Internet sources did not match our computer architecture and hence midway during their monitoring, the programs crashed and was not monitored for the whole period of 5 minutes. Hence, these files, although being malware, carried lower API call data and frequency values. Therefore, they were not classified as malware. As a result, the False Negative Rate increased and the accuracy measurement was affected. The accuracy increases, if we use larger dataset in our experiment. With the collection of more malware PE files compatible with our system architecture, our test bed output would contain larger sized call log sequence files which significantly improves the accuracy. Also, the false detection rate decreases, if we add more training datasets.

B. Malware Detection using API-CTMM

The features used for API Call Transition Matrix Mining (API-CTMM) are the 94 API calls' transitions. The additional feature of log file size considered for API-CFM is discarded for API-CTMM. As explained earlier, the reason for removing the log file size feature is that huge size variation is observed between the log file size of malware and benign file and PCA algorithm brought this as the dominant feature, keeping API call transition in the lower profile. By removing the log file size feature, PCA brought 10 dominant features from the considered 94 API call transition matrix.

The dataset identified for training and testing of API Call Transition Matrix Mining is identical to that used for API Call Frequency Mining, except that the last parameter namely *log file size* is not considered. The data mining algorithms used for API-CTMM are also different from that used for API-CFM. The supervised learning algorithms: Random Forest (RF) classifier, Naive Bayes (NB) classifier, Support Vector Machine (SVM) classifier, and Adaboost classifier are used in the API Call Transition Matrix Mining whereas K-Means Clustering and Classification algorithms were used in the API Call Frequency Mining.

Another added procedure in API-CTMM is that the introduction of Principal Component Analysis (PCA) for reducing the dimensionality of the feature set. PCA brought the dimension of the feature set from 94 to 10. We also considered one test case by taking only the most

ISSN: 2005-4238 IJAST

dominant 10 API Call Transitions, identified by PCA. Thus, we have three cases for the testing or performance analysis of our API Call Transition Matrix Mining technique.

We present the performance results of the API-CTMM technique by conducting the experiments with the following three cases:

Case 1: With 94 API call transition features
Case 2: Features selected with PCA (PCA selected 10 features out of the 94 features)
Case 3: With the 10 API call transition features selected by PCA

All the four classifiers Random Forest classifier, Naive Bayes classifier, Support Vector Machine classifier, and Adaboost classifier were trained and tested with the data for the three cases considered. For training the classifier, as mentioned in the previous section, the malware were downloaded from the websites of researchers and scanned the malware using virustotal [21] and observed that the malware files were not detected as virus. The benign programs were copied from the Windows32 directory of MS-Windows and trained our classifiers.

For conducting the testing, the malware were downloaded from virussign [9] and different programs were taken from MS-Windows as benign program files. For testing our malware detection algorithm, we used 16 malware and 5 benign programs.

Case 1: With 94 API call transition features

Experiments were conducted for the detection of malware and benign program files using the data set consisting of all the 94 features. Table VI shows the summary of the results along with the accuracy of detection and the time taken for testing. Out of the 16 malware, the data mining algorithms: Random Forest (RF) classifier, Naive Bayes (NB) classifier, Support Vector Machine (SVM) classifier, and Adaboost classifier detected 14, 16, 16 and 15 of them as malware and detected the remaining malware 2, 0, 0, and 1 as benign program. Out of 5 benign programs the classifiers detected 2, 4, 1, and 1 as benign programs and the rest 3, 1. 4, and 4 as malware. The accuracy of detection is computed using the parameters, True Positive, True Negative, False Positive and False Negative of Equation 1. The detection accuracy of the data mining algorithms: Random Forest classifier, Naive Bayes classifier, Support Vector Machine classifier, and Adaboost classifier were respectively 76.1%, 95.23%, 80.95, and 76.19%. The time taken for detection by the classifiers were 9.17 seconds, 8.54 seconds, 7.82 seconds, and 8.78 seconds respectively. Naive Bayes classifier resulted better accuracy out of the four classifiers, we considered with full data set.

Actual Class		**Predicted Class**		**Accuracy**	**Time to train and test**
		Malware	**Benign**		
Random Forest	Malware	14	2	76.19%	9.17 secs
	Benign	3	2		
Naive Bayes	Malware	16	0	95.23%	8.54 secs
	Benign	1	4		
Support Vector Machine	Malware	16	0	80.95%	7.82 secs
	Benign	4	1		
Adaboost	Malware	15	1	76.19%	8.78 secs
	Benign	4	1		

Table VI: Results of malware detection using API-CTMM by considering 94 API calls as features

ISSN: 2005-4238 IJAST

Case 2: Features selected with PCA (10 features selected by PCA)

PCA Analysis is carried out to reduce the dimension of the data set from 1×94^2. The reduced dimension is arrived at 1×10^2. We conducted the testing of our method API-CTMM with the data set of reduced-dimension. The summary of the detection result is shown in Table VII.

Actual Class		**Predicted Class**		**Accuracy**	**Time to train and test**
		Malware	**Benign**		
Random Forest	Malware	15	1	85.75%	0.0471 secs
	Benign	2	3		
Naive Bayes	Malware	14	2	90.48%	0.0069 secs
	Benign	0	5		
Support Vector Machine	Malware	16	0	80.95%	0.0077 secs
	Benign	4	1		
Adaboost	Malware	14	2	70.95%	0.0200 secs
	Benign	2	3		

Table VII: Results of malware detection using API-CTMM by considering PCA reduction on 94 API calls

In this case also, Naïve Bayes classifier yielded better results in terms of accuracy for malware detection. It is also observed that the time duration for testing is very less. For example, Naïve Bayes classifier took 8.54 seconds for the malware detection using all the 94 features, where as it took only 0.0069 second for malware detection using the PCA reduction method. The PCA based testing is 1200 times faster than that of using all the 94 features.

Case 3: With the 10 API call transition features selected by PCA

In this case, we took only the dominant API calls selected by PCA method. In similar lines of other two cases, we prepared the test data set and tested this case for all the classifiers, using our technique API-CTMM. The summary of the results for our detection method using the four classifiers is shown in Table VIII.

Actual Class		**Predicted Class**		**Accuracy**	**Time to train and test**
		Malware	**Benign**		
Random Forest	Malware	16	0	85.75%	0.0342secs
	Benign	4	1		
Naive Bayes	Malware	16	0	95.23%	0.0132secs
	Benign	2	3		
Support Vector Machine	Malware	16	0	80.95%	0.0162secs
	Benign	4	1		
Adaboost	Malware	14	2	90.48%	0.0209secs
	Benign	2	3		

Table VIII: Results of malware detection using API-CTMM by considering dominant 10 API calls as features

ISSN: 2005-4238 IJAST

In this case also, Naïve Bayes classifier emerged as the better classifier than the other three. The detection accuracy is same as that of considering all the 94 API call features. The time duration to test is very less for Case 3 test data with respect to Case1 test data.

VI. CONCLUSIONS AND FUTURE SCOPE

The security of an enterprise is of prime importance when it is connected to the Internet. In this paper, we proposed two techniques, (i) API call frequency mining (API-CFM) and (ii) API call transition matrix mining (API-CTMM), for detecting malicious programs in Enterprise networks. Our results show that API-CFM provides an accuracy of 76.19%. On the other hand, the inclusion of API call transitions in API-CTMM provides an accuracy of 95.23%, which is comparable with the existing detection methods. In addition, the reduction in feature dimension using PCA significantly reduces the time of malware detection. It is also observed that when number training samples is higher than the testing samples, the accuracy of detection improves.

Our approach brings a more real-time approach to detect malware. We used novel features including the API call transition matrix and log file size which were not used in any other literature. Our multi-dimensional detection algorithm of classification and detection brings the novelty to even a higher level.

The potential future work in this area include: 1). In order to improve the accuracy, our methods may be used with a training set of significant size. 2). Further study may be conducted for analyzing the relationship between API call transition matrix and the detection accuracy. 3). Our detection module can be extended to large enterprise networks as a case study to gather additional results on performance.

The framework developed is capable of adding more machine learning intelligent algorithms to increase the accuracy of detection.

REFERENCES

[1] "modpos," http://www.isightpartners.com/2018/11/modpos/.
[2] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malwaredetection based on mining api calls," in Proceedings of the 2010 ACM symposium onapplied computing. ACM, 2010, pp. 1020–1025.
[3] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on api functioncalls and their arguments," in The 16th CSI International Symposium on ArtificialIntelligence and Signal Processing (AISP 2012). IEEE, 2012, pp. 563–568.
[4] S. Burji, K. J. Liszka, and C.-C. Chan, "Malware analysis using reverse engineering anddata mining tools," in 2010 International Conference on System Science and Engineering.IEEE, 2010, pp. 619–624.
[5] K. Iwamoto and K. Wasaki, "Malware classification based on extracted api sequencesusing static analysis," in Proceedings of the Asian Internet Engineering Conference. ACM,2012, pp. 31–38.
[6] "Praveen Sundar, P.V., Ranjith, D., Vinoth Kumar, V. et al. Low power area efficient adaptive FIR filter for hearing aids using distributed arithmetic architecture. Int J Speech Technol (2020). https://doi.org/10.1007/s10772-020-09686-yevade-security-detection/, accessed: 19-11-18.
[7] T. Holz, M. Engelberth, and F. Freiling, "Learning more about the underground economy:A case-study of keyloggers and dropzones," in European Symposium on Research inComputer Security. Springer, 2009, pp. 1–18.
[8] G. Hoglund and J. Butler, Rootkits: subverting the Windows kernel. Addison-WesleyProfessional, 2006.

ISSN: 2005-4238 IJAST

[9] Umamaheswaran, S., Lakshmanan, R., Vinothkumar, V. et al. New and robust composite micro structure descriptor (CMSD) for CBIR. International Journal of Speech Technology (2019), doi:10.1007/s10772-019-09663-0,.
[10] Karthikeyan, T., Sekaran, K., Ranjith, D., Vinoth kumar, V., Balajee, J.M. (2019) "Personalized Content Extraction and Text Classification Using Effective Web Scraping Techniques", International Journal of Web Portals (IJWP), 11(2), pp.41-52
[11] "apimonitor," https://www.rohitab.com/apimonitor, accessed: 19-10-18.
[12] S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye, "Cluster-oriented ensemble classifiers for intelligent malware detection," in Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015). IEEE, 2015, pp. 189–196.
[13] A. V. Romero, VirtualBox 3.1: Beginner's Guide. Packt Publishing Ltd, 2010.
[14] I. VMware and R. Calculator, "Vmware," 2016.
[15] J. Potier, "Winapioverride32," 2012.
[16] P. Faruki, V. Laxmi, M. S. Gaur, and P. Vinod, "Mining control flow graph as api call-grams to detect portable executable malware," in Proceedings of the Fifth International Conference on Security of Information and Networks. ACM, 2012, pp. 130–137.
[17] Vinoth Kumar, V., Arvind, K.S., Umamaheswaran, S., Suganya, K.S (2019), "Hierarchal Trust Certificate Distribution using Distributed CA in MANET", International Journal of Innovative Technology and Exploring Engineering, 8(10), pp. 2521-2524.
[18] Maithili, K , Vinothkumar, V, Latha, P (2018). "Analyzing the security mechanisms to prevent unauthorized access in cloud and network security" Journal of Computational and Theoretical Nanoscience, Vol.15, pp.2059-2063.
[19] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of api call signatures," in Proceedings of the Ninth Australasian Data Mining Conference-Volume 121. Australian Computer Society, Inc., 2011, pp. 171–182.
[20] V.Vinoth Kumar, Ramamoorthy S (2017), "A Novel method of gateway selection to improve throughput performance in MANET", Journal of Advanced Research in Dynamical and Control Systems,9(Special Issue 16), pp. 420-432.
[21] V. Total, "Virustotal-free online virus, malware and url scanner," Online: https://www. virustotal. com/en, 2012.
[22] Dhilip Kumar V, Vinoth Kumar V, Kandar D (2018), "Data Transmission Between Dedicated Short-Range Communication and WiMAX for Efficient Vehicular Communication" Journal of Computational and Theoretical Nanoscience, Vol.15, No.8, pp.2649-2654.
[23] Kouser, R.R., Manikandan, T., Kumar, V.V (2018), "Heart disease prediction system using artificial neural network, radial basis function and case based reasoning" Journal of Computational and Theoretical Nanoscience, 15, pp. 2810-2817
[24] Shalini A, Jayasuruthi L, Vinoth Kumar V, "Voice Recognition Robot Control using Android Device" Journal of Computational and Theoretical Nanoscience, 15(6-7), pp. 2197-2201
[25] Jayasuruthi L,Shalini A,Vinoth Kumar V.,(2018) " Application of rough set theory in data mining market analysis using rough sets data explorer" Journal of Computational and Theoretical Nanoscience, 15(6-7), pp. 2126-2130

ISSN: 2005-4238 IJAST