Behavioural detection with API call-grams to identify malicious PE files

Conference Paper · August 2012

DOI: 10.1145/2490428.2490440

4 authors:

Parvez Faruki
Government MCA College, Ahmedabad, Gujarat
21 PUBLICATIONS **245** CITATIONS
SEE PROFILE

Vijay Laxmi
Malaviya National Institute of Technology Jai…
185 PUBLICATIONS **919** CITATIONS
SEE PROFILE

Manoj Singh Gaur
157 PUBLICATIONS **804** CITATIONS
SEE PROFILE

P. Vinod
SCMS School of Engineering & Technology, E…
43 PUBLICATIONS **175** CITATIONS
SEE PROFILE

Behavioural Detection with *API Call-grams* to Identify Malicious PE Files

Parvez Faruki[*]
Malaviya National Institute of Technology, Jaipur, India.
parvezfaruki.kg@gmail.com

Vijay Laxmi[†]
Malaviya National Institute of Technology, Jaipur, India.
vlaxmi@mnit.ac.in

M. S. Gaur[‡]
Malaviya National Institute of Technology, Jaipur, India.
gaurms@mnit.ac.in

Vinod P.[§]
Malaviya National Institute of Technology, Jaipur, India.
vinodp@mnit.ac.in

ABSTRACT

Present day malware shows stealthy and dynamic capability to avail administrative rights and control the victim computer [10]. Malware writers depend on evasion techniques like code obfuscation, packing, compression, encryption or polymorphism to avoid detection by Anti-Virus (AV) scanners as AV primarily use signature based detection. According to the FireEye Threat report second half of 2011 [15], top 50 malware have generated 80% infections. Malware like Zues, Conficker, Koobface have become more stealthy by use of pay per install toolkits like Blackhole [15]. Pay per install toolkits make the samples dynamic in nature. This has led to exponential increase of unknown, zero-day malware [14].

To complement the signatured approach, a good behavioral scheme is imminent due to exponential increase in number of encoded malware samples. Behavioural analysis can detect unknown, encrypted, zero day malware, but these methods result in increased false alarm rate. We propose a behaviour model that represents abstraction of a binary by analyzing the Application Programming Interface (API) strings made by Windows Portable Executable (PE) [25] files. Our focus is based on extracting temporal snapshots of malware and benign executables known as *API Call-grams*, as API strings are primarily written for software development kits to generate sane code. Malcode writers misues the available functionality to keep the code compact and escape being detected by AV software.

[*]M.Tech. Scholar, Computer Engineering Department.

[†]Corresponding Author, Associate Professor and Head, Computer Engineering Department.

[‡]Professor, Computer Engineering Department.

[§]Ph.D Scholar, Computer Engineering Department.

secureIT '12 Amrita Institute, Chennai, India

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection–Invasive Software(Virus, Worms, Rootkits etc.); I.5 [**Computing Methods**]: Pattern Recognition

General Terms

Experimentation and Security

Keywords

Behavioural Analysis, Cuckoo Sandbox, Malware, *API Call-grams*, Data mining

1. INTRODUCTION

1.1 Overview

A malicious file also known as malware/malcode intends to harm the user by deleting important content from computer, compromise the integrity of system by surreptitiously hiding in the system, act as software robot being controlled by malware and unknowingly be a part of crimeware, ransomware syndicate by being a conduit in crime. Malcode is a commonly used term for harmful programs like Virus, Worms, Trojans, Backdoor, Spyware, Pay per Install programs, FakeAV, Rootkits [13] to name a few. Our focus of research is limited to Windows Portable Executables (PE) files as Microsoft Windows operating system binaries are prominent targets of malware writers [35].

AV programs performs policing act, identifies, protects the machine and mitigates harmful software and their malicious intent. AV programs primarily use signature based detection to identify a cognized malware by extracting a unique hex-pattern known as signature. Signature based detection schemes are built around a large database containing byte sequence which characterize individual files. Unknown, zero-day malware detection fails as the signature for the same is not available in the database of AV [1]. Easy availability of encryption kits, obfuscation tools and packing techniques create a serious problem for static signature generation approach. Static analysis approach is necessary but not sufficient in the changing scenario.

Behaviour model identifies certain functionalities that are prevalent and prominently used for harmful purpose. Obfuscation techniques do not change the abstraction of a pro-

gram. This includes but not limited to, stealing account information, passwords, credit card information for executing financial frauds. Behaviour identification mechanism observes the functionality by executing the file on a Virtual [8], Emulated or Bare-metal controlled environments [21] to identify the strains a sample leaves on the system after execution. The system being virtual, gets cleaned after collecting malware strain on the system once the execution is complete. In–Execution analysis and detection methods combined with pattern recognition techniques have gained ground in the era of obfuscation tools, as the outer layer/cover gets removed once the code is executed.

Obfuscation tools create a layer/cover that makes static analysis very difficult. Static analysis is also hindered by custom packing approaches. This leads to analysing the obfuscated code instead of original malware. Moreover, size of signature database increases exponentially as multiple variants can be generated from a single sample. Behaviour model defines certain functionalities as malicious, as a result false positives become an issue as normal programs also perform such jobs. Application Programming Interface (API) calls of Windows represent the abstract functionality of the programs. Our focus is to minimize the false positives by taking the sequence of API calls in order, or *API Call-grams* instead of looking for the occurance of an API in malware and saneware. Searching for the number of API calls made by malicious and sane programs lead to high false positives [6]. The novelty of our approach is use of API call sequence known as *API Call-grams* to detect unknown, real malicious samples.

1.2 Motivation

Huge increase in number of malware samples have led to look for complementary automated analysis techniques to reduce burden on signature based AV detection. Malicious programs misuse API strings as many of them are still undocumented due to proprietary constraints of Microsoft [12]. Malicious code developer tend to misuse system calls and keep the size of executables compact due to easy availability of functionality provided by the propriety vendor and evades detection by performing malicious actions, utilizing the functionality of API calls.

This paper is organized as follows. Section 2 discusses data–set preparation and brief overview of our proposed method, behavioural *API Call-gram* on a virtual machine based dynamic environment. In Section 2, we also discuss *API Call-gram* generation along with classification tools and techniques. Section 3 discusses experimental setup, results, along with classification approach with pattern recognition tool WEKA [33]. Comparison of our proposed method is discussed in section 4. Related work is covered in section 5. Section 6 concludes the paper with future scope.

2. PROPOSED METHODOLOGY

Here we discuss the outline of proposed work for *API Call-gram* along with data–set preparation as depicted in Figure 1.

- Collect benign and malware samples performing different functionality and varied size for maximum API coverage.
- Submit the executables to Cuckoo Sandbox [8] to generate API call trace under a controlled virtual environment.
- Preprocess the trace report and extract API calls for each samples from individual files.
- Generate *API Call-grams* for n=1,2,3,4 and so on.
- Generate prominent API grams used by benign samples. Likewise, generate prominent API grams used by malware samples.
- Generate Feature vectors for benign and malware samples based on common as well as discriminant API calls.
- Generate CSV files and normalize them for input to the classifiers supported by WEKA library.
- Generate *API Call-grams* model for malware and benign samples with WEKA.
- Apply classification algorithms Random Forest [7], J-48 Decision Tree [31], Voted Perceptron (VP) [16], Sequential Minimal Optimization (SMO) and Naïve–Bayes [31] supported by WEKA [33].

2.1 Dataset Preparation

The data set consists of different categories of programs (refer Tables 1 and 2) that includes varied category of API calls such as performing network activity, file operations, graphic functionality, registry operations, and memory related tasks. Windows API is a core functionality for easy interaction with the system and ensures fast and compact source code development. Misuse of known as well as undocumented API by the malware writers to fulfill the malafide intention needs proper analysis by *Call-gram* sequencing.

2.2 Extraction of API Calls

We discuss the process of API string extraction on an open source, automated, virtual machine based behavioural malware analysis environment `Cuckoo Sandbox` [8]. Cuckoo uses custom components written in python to analyse the behavior of a process by executing the file on an isolated Windows environment. Cuckoo analyses untrusted source to obtain information by executing the process in a caged real environment in form of a API trace report. The trace report is an abstract representation of the behaviour of the executable. On this basis we preprocess the trace report and extract API calls from each report file.

API calls are extracted from the trace report for sample monitored on virtual environment. The trace were generated with Cuckoo version 0.2 for 2510 benign and 2487 malware samples performing varied functionalities. Version 0.2 used custom developed hooking engine CHook [8].

The sandbox environment generates API trace report, network activity in `.pcap` file and screen shots at various intervals for analysis.

The benign program are collected from fresh installation of Windows XP SP3, system utilities and resource like cnet [9]. The downloaded benign samples were scanned with Kaspesky AV 2011 with latest database updates to acknowledge their clean behaviour before their use in the experiment. Malware dataset are gathered from well known malware repositories such as VX Heavens [36], OffensiveComputing [11], and Honey–pot contents of allied user agencies.

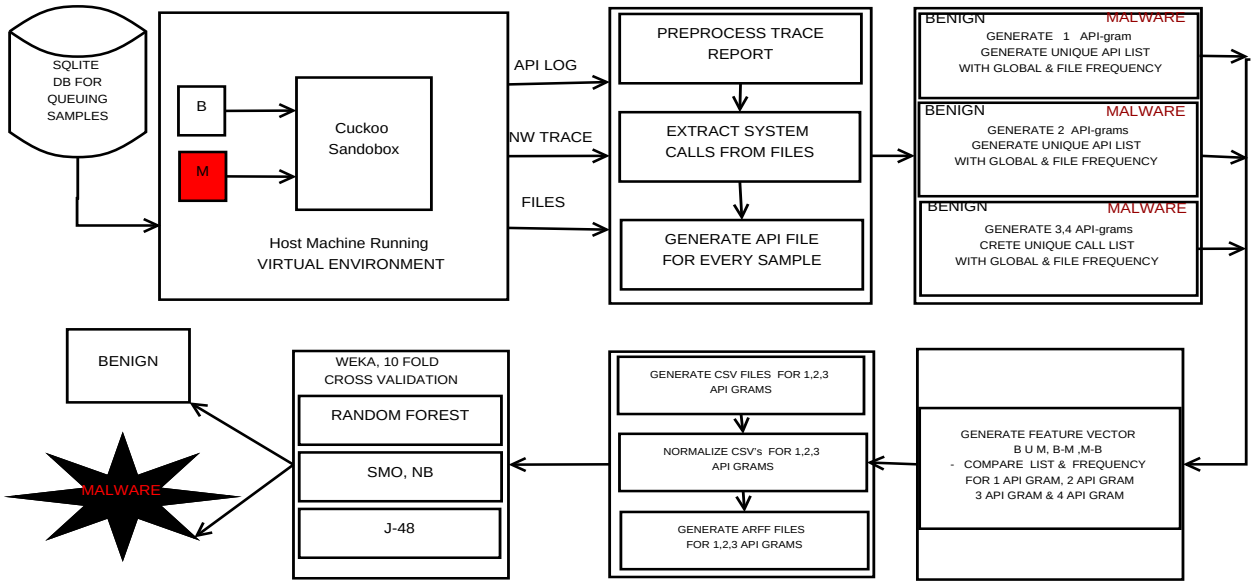

Figure 1: `API Call-gram` approach for Malware Detection.

Table 1: Benign (B) Data set

Category	Number	Size(min.)	Size(max)
System-software	1230	800 KB.	150 MB.
Application-software	400	4 MB.	150 MB.
Utility-software	567	1 MB.	48 MB.
Downloaders	50	1 MB.	150 MB.
Media-Players	100	3 MB.	100 MB.
Device Drivers	200	1 MB.	230 MB.
Virtualization s/w	3	130 MB.	250 MB.
Anti-Virus	10	200 MB.	400 MB.
Network-Activity	35	2 MB.	43 MB.

The samples were submitted to Cuckoo for execution on virtual machine based controlled environment. Traces of executed samples are fetched from the analysis folder on Ubuntu host machine. An automated script stores the executable samples, benign and malware respectively into the SQLite database interacting with Cuckoo Sandox for execution in the controlled environment. These samples were monitored for a maximum duration of 300 seconds.

Table 2: Malware (M) Data set

Category	Number	Size(min)	Size(max)
Virus	430	1.3 MB.	20 MB.
Worm	632	800 KB.	90 MB.
Trojan	130	2.3MB.	120 MB.
Backdoor	243	1 MB.	60 MB.
RootKits	25	1 MB.	4 MB.
Peer-Peer Bot	10	2 MB.	12 MB.
New Malware	479	1.3 MB	7 MB.
FakeAV	47	15 MB.	23 MB.
Hoax	107	2 MB.	17 MB.
User agencies	670	2 MB.	17 MB.

The generated log report is pre–precessed to extract the APIs calls for each file into a individual files. These individual files are input to generate a single, two, three and four *API Call-grams*. Unique *API Call-grams* with their global frequency and occurance across the benign and malware samples is generated respectively. Their occurance in individual files is generated for unigram, bigram, trigram etc. for benign and malware PE files.

2.3 *API Call-grams*

An imbalanced data set has large number of files for a particular class may be biased towards the API calls of that particular class. Moreover, we also need to perform hypothesis testing like t-test when one particular class has very few samples compared to another. Motivated by the work of authors in [26] we have considered nearly equal number of benign and malware files in our experiments.

Table 3: Number of API attributes in call grams

Call-gram	B	M	B ∩ M	B \ M	M \ B	B ∪ M
1 API-gram	36	35	35	1	0	36
2 API gram	493	576	420	73	156	649
3 API gram	1052	1173	664	388	509	1511
4 API-gram	6303	8084	1602	4699	6482	12783

First, the API sequence are extracted with a script from the call log generated by Cuckoo sandbox. The preprocessing step extracts the API calls ignoring the parameters of each call. We generate a single *API Call-gram* along with a unique single API call list that consists of global count of API and the presence in number of files in both benign and malware. Subsequently we generate API string two, three, four grams and so on. The example shown below explains the process of generating *API Call-grams*.

Suppose File 1 has a set of API LoadLibrary(A1), GetProcAddress(A2),CreateFile(A3),VirtualAlloc(A4). We generate *API Call-grams* and their unique API list for benign and malware sets.

- 1-API gram- A1, A2, A3, A4

- 2 API gram- A1A2,A2A3,A3A4
- 3-API gram- A1A2A3,A2A3A4
- 4-API gram- A1A2A3A4

Once the *API Call-grams* are generated, we find common calls among *API Call-grams* of benign and malware samples (refer Table 3). The generated API gram displays frequency of individual API calls and their global occurance with unique API grams across the files. This list consists of

- Common API calls of benign and malware samples ($B \cap M$)
- Union of API calls among benign and malware ($B \cup M$)
- Discriminant calls present benign samples ($B \setminus M$)
- Discriminant calls present in malware samples ($M \setminus B$)

Similarly common and prominent API call are generated for 2, 3 and 4 *API Call-grams*. These prominent API strings of different classes are our parameters for input to pattern recognition algorithms.

2.4 Classification Technique

Classification is performed with WEKA [33] on the feature vectors generated for one to four *API Call-grams*. Feature vector table is normalized before submission to classifier. Weka library has implemented set of pattern recognition, machine learning and clustering algorithms to perform binary classification. We classify the feature vector attributes by executing feature vector (FV) tables on Random-Forest (RF) [7], Sequential Minimal Optimization (SMO) [28], J-48 decision tree [29] , Naïve–Bayes [31] and Voted Perceptron [16] [31]. We perform 10–fold cross–validation [22] as it is the most effective k–fold classification scheme.

Random Forest is an ensembled classifier. It aggregates results of multiple trees generated for the dataset. J-48 algorithm creates a decision tree that is based on the attribute of training data. It continues the process until it finds the attribute that can distinctly identify among the given instance based on minimum entropy value or maximum information gain.

Sequential Minimal Optimization (SMO) algorithm is implemented in WEKA library to train support vector classification. SVM rudimentary model is a classifier that performs learning of a decision boundary among two classes (eg. M and B) in an input space. This algorithm globally identifies missing values and replaces missing values, and transforms nominal attributes into binary ones. It also normalizes all attributes by default.

Voted Perceptron (VP) is a classifier where the input x is mapped to an output function. It is based on perceptron algorithm of Rosenblatt and Frank [16]. This algorithm advantages the data linearly separable and has large margins. VP is efficient with respect to computation time in comparison to Vapnik's SVM [17].

Naïve–Bayes classification is based on Bayes conditional probability [31]. It identifies all the attributes of data and analyses them independent of each other assuming that they are equally important and disjoint. This leads to low accuracy.

2.5 Evaluation Metrics

Evaluation metric is computed on basis of True positives (TP)–correctly classified malware instances, True Negative (TN)–correctly identified benign instances, False Positive (FP)–benign instances mis–classified as malware and False Negative (FN)–malicious samples mis–classified as benign [23]. *True Positive rate (TPR)* and *True Negative Rate (TNR)* are known as *sensitivity* and *specificity* respectively. Accuracy (ACC) of malware detection systems is measured with respect to high TPR and low FPR [31] [38].

- `TPR = TP/(TP + FN)`
- `FPR = FP/(FP + TN)`
- `TNR = TN/(TN + FP)`
- `FNR = FN/(FN + TP)`
- `ACC = (TP + TN)/(TP + TN + FP +FN)`

3. EXPERIMENTS, RESULTS, DISCUSSIONS

The experiments were performed on Intel Core i3 2.40 GHz machine, 4 GB RAM, Ubuntu 10.10 host operating system running Oracle Virtualbox [34] to host the sandbox environment. Cuckoo sandbox Version 0.2 is installed on the host with Windows XP [24] as guest Operating system (OS).

3.1 Results

Table 4 displays experimental results for our *API Call-grams* approach. The results of 1 API-gram displays an accuracy of 86%. This result of 1 gram is analogous to the global and file frequency of API calls discussed by Aksan sami et el [30] and Vinod P. et el [27]. As calls are primarily developed for use by software developers, false positives increase due to their presence in sane and malcode. As a result the false positives increase leading to lower Accuracy (ACC).

Two *API Call-grams* result show an improvement over single *API Call-gram*. 3 API call-gram show a significant increase in accuracy from 88 to 98.3%. This is due to the frequency of sequence of API calls made by malicious programs to perform a task that a normal program would make but have lower frequency of calling the sequence. For example, creating a file, writing to it and sending it also performed by benign code. But the frequency of calling such *API Call-grams* differ significantly in both types. This leads to higher accuracy (refer Table 4). Results of 4 API grams show a marginal increase in the detection accuracy, but the number of common and discriminant attributes is high compared to 3 *API Call-gram* model.

Table 4: Detection Accuracy in Percentage

Classifier	1 API-gram	2 API-gram	3 API-gram	4 API-gram
RF	86.2	88.3	98.3	97.9
SMO-SVM	84.7	86.9	**98.6**	98.1
VP	83.6	88.4	**98.7**	98.2
J-48	82.2	83.9	95.2	94.6
NB	72.2	79.3	91.1	90.4

3.2 Discussions

It is important to detect a malicious file from a sane file. We have compared our method against the most relavant work either implementing static analysis or behavioural methods for classifying malicious files with high accuracy. *API Call-gram* approach gives better accuracy compared to other approaches for malware detection as in general, the *Call-gram* model captures temporal snapshot of a program which reflects the behavioral patterns of an executable.

In case of smaller value of *API Call-gram* model for n=1 or 2, fewer number of features exist in the feature space and the frequency of features is also similar in both classes (i.e. malware and benign), which leads to mis–classification. In case of higher value of *API Call-gram* for n=4, 5, 6 and above large number of features exist in the feature space. These features are either absent in one of the classes, or they appear with very small frequency in both the classes. Thus, these features are referred by us as redundant features and they degrade classification accuracy. The figure below depicts TPR and FPR for different *API Call-gram*. This figure depicts that the 3 *API Call-gram* model produces better classification accuracy compared to other models.

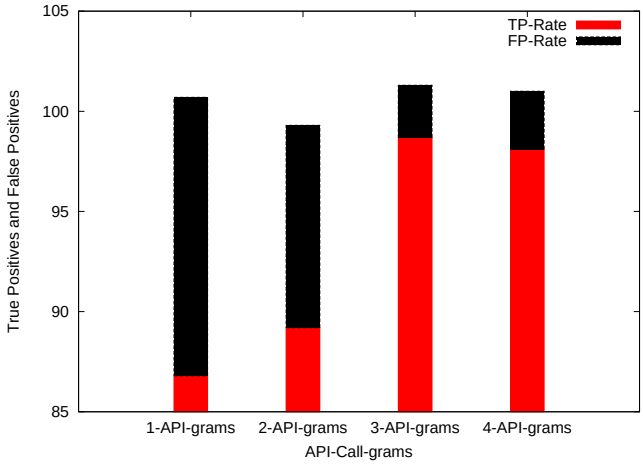

Figure 2: Comparison of `API Call-gram` TPR, FPR

4. COMPARISON WITH PREVIOUS WORK

Comparison of our proposed detection model with existing techniques is summarized in Table 5. Ronghua Tian et el [32] used global frequency and occurance of API calls across the sane and malcode files for malware detection. The authors used an imbalanced data–set for analysis, but did not use any normalization mechanism to remove the bias for a small set of benign files. The authors have reported accuracy 97.3% for detecting malicious code.

Yoshiro et el [18] implemented the behaviour of process by executing the malicious files on a virtual environment with an aim to minimize FP–mis–classification of benign reported as malware towards zero. The reported TP Rate for their approach is 60% with close to zero false positives.

Faraz Ahmed et el [1] have considered detection of malware based on different categories such as Network activity, Memory management, File manipulation, graphic control, registry activity and created a core API set of 237 malware known as critical API's to detect malicious code. Classification accuracy of their method is reported 97%. The authors have not discussed the false positive generation in their approach. In [37] the authors have discussed their method based on behaviour of API calls with respect to their frequency of occurance. The sequence of API calls is generated using distance measure methods. The authors claim to have reported the malware from the wild undetected by known Anti-Virus software.

F. Karbalaei et el [20] have used frequent subgraph mining approach for detection of malware. The authors extract the graph behaviour, then perform frequent subgraph mining using binary occurance of a subgraph in executable files to detect malicious content. The authors have reported detection accuracy 96.6%.

5. RELATED WORK

Malware detection approach is either static, dynamic or hybrid. Dynamic analysis is performed by analysing the trace and strains of executable sample in a controlled environment running on virtual or emulated platform. The problem is all the paths and conditions may not occur for executing the samples. Static detection methods analyse the structural information of an executable with reverse engineering techniques. Analysis is performed without executing the malafide code. Static detection is able to cover all the paths, but is hindered by obfuscation techniques [13]. Hybrid detection is combination of both methods. We discuss the most relevant contribution from different authors on either static approach to malware detection, or behavioral analysis analysis.

Ronghua et el [32] proposed a behavioral system to differentiate malware and clean files by analyzing the traces API calls of Portable Executable (PE) files by executing the samples on virtual machine with Microsoft Detours [19] and HookMe as tracing tools. They extract the API calls from the generated trace reports. The authors have used imbalanced data set that consists of huge number of benign and a very small set of clean files. The API call report classifies the malware and clean files based on global frequency and occurance of API strings in individual files. The authors report accuracy of above 97% with low false positive. The imbalanced data set tends to be biased towards the malware API calls of malware files as authors use imbalanced dataset. Moreover, authors have not specified the categories of clean files, as different categories of benign programs also use varied API calls in software develpment.

Yoshiro et el [18] proposed malware detection on a small size balanced dataset with a focus on minimizing false positives. The authors consider system calls used by benign as well as malware programs to increase the coverage of common API calls. This approach reduces the detection rate to around 60%, but their main focus is on reducing false positives (FP). The authors have worked on a very small malware data set of 83 samples consisting 19 viri, 23 worms and 25 trojan programs. Benign data set consists of 18 installer programs 6 uninstallers, and some binaries related to web functioning. Proper analysis and effective detection model is difficult in case of such a small dataset.

In [30], authors have used reverse engineering technique to extract system calls from the Import Address Table (IAT) of Windows PE files. The authors then, look for global occurance of API calls in benign and malware samples. They have classified the malware and benign samples with data

Table 5: Comparison of `API Call-gram` approach with existing work

Author	Dataset size Benign-B, Malware-M	Method	DR, ACC, TPR
Ronghua Tian [32] *et al*	1456-B,456-M	Behaviour Analysis using API frequency	97% (ACC)
Yoshiro Fukushima [18] *et al*	26-B,25-M data too less	Behaviour Analysis of suspicious process	60% (DR), 0% (FP)
Akshan Sami [30] *et al*	Not specified	Static Analysis Frequency of API Calls	97% (ACC)
Gérard Wagener [37] *et al*	10-B,104-M	Behavior Analysis Similarity score of API	Not specified
F. Karbalaei [20] *et al*	349-B,404-M	Behavioral Subgraph Mining	96.6% (DR), 3.4% (FP)
`Proposed Method`	**2465-B, 2387-M**	**Behaviour Detection using** `API Call-gram`	**98.6**% (ACC), 2.6% (FP)

mining techniques. This method can be easily evaded by use of obfuscation tools. System calls extracted from IAT of obfuscated file belong to the encypted layer instead of the actual file. Moreover, unpacking the code before analysis is time-consuming and does not guarantee accurate unpacked code.

In [37] the authors use dynamic behavioral pattern for malware detection using pattern mining techniques. The authors execute the samples on a virtual environment and generate the API traces of the files. Authors compute similarity score, and distance calculation among malware behaviour by calculating similarity matrix depending on the number of system calls of particular file. The authors calculate similarity using hallinger distance to generate similarity score. The malware data set used for experimentation is very small, 104 samples.

Faraz Ahmed et el [1] claim to have used arguments and sequence of API calls by categorizing the system calls into six different functional categories. The authors use API Monitor [3] to extract system calls from windows executable files. Authors have implemented data mining techniques to detect benign from malicious code. They report an accuracy of 97% on a small set of core API set limiting to 237.

In [20] the authors have used virtual machine based behavioural malware detection. API call log is generated after executing the samples on the guest O.S. The API calls are extracted from the log file to generate the graphs. The generated graphs and their subgraphs are compared for similarity. The subgraphs are mined with respect to their presence/absence. Binary classification is performed based on subgraphs. The authors have used the dataset consisting 858 malware samples and 393 benign files.

Ulrich Bayer et el [4] used ANUBIS [2] sandbox to execute the malicious samples in a controlled environment based on QEMU [5] emulator. They collect the trace of system calls that the PE file invokes during execution. The authors added the taint generation facility to the environment for tracking the sample. The sample when executed, taint is generated and behaviour profile for an executable file is created. Clusters are generated based on their behaviour based on system calls.

In [27] the authors extract the API calls on a dynamic environment using QEMU. The authors detect metamorphic malware samples based on API call frequency, by looking for calls prominently used by malware and benign programs. They classify the samples using pattern recognition techniques.

6. CONCLUSIONS AND FUTURE SCOPE

In this paper we discuss behavioural analysis of Windows PE Executables by extracting the API call features, then generating API-call-grams for effective detection of malware and reducing false positives. Abstract representation of a program is obtained from the sequence of API calls made by a file during execution. Novelity of our approach is considering the API grams, to reduce false positives. Mere frequency of API strings leads to high false positives as API strings are originally written for windows SDK. API-grams consider the sequence in which calls are made one after another which leads to high detection and low false positives. Our contribution includes extracting API calls from sandbox running on a controlled virtual environment. The log which represents abstraction of a program is converted upto 4 API-grams and generating individual and common API call strings. Then these API grams are used for differentiating benign and malware with an accuracy of 98.6% and False positives of 2.6%. Our approach performs better than the recent models on a comparatively varied and big database. In future we would like to go for higher API Call-gram sequencing and apply feature reduction methods to reduce false positives close to theoretically 0%.

7. REFERENCES

[1] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq. Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pages 55–62. ACM, March 2009.

[2] ANUBIS. ANUBIS-Automated Sandbox Environment. http://anubis.iseclab.org.

[3] ApiMonitor. API Monitor–Spy and Display API Win32 Calls. http://www.apimonitor.com.

[4] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, Behavior-based Malware Clustering. In *NDSS*, IJCAI'95, pages 1137–1143. NDSS, 2009.

[5] F. Bellard. QEMU, a fast and portable Dynamic Translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, Berkeley, CA, USA, 2005. USENIX Association.

[6] G. Bonfante, M. Kaczmarek, and J.-Y. Marion. Architecture of a Morphological Malware Detector. *Journal in Computer Virology*, 5(3):263–270, 1991.

[7] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[8] Claudio-Guarnenieri. Cuckoo–Open Source Malware Analysis Sandbox–V.0.2. http://www.cuckoobox.org/, Last Accessed March 2012.

[9] CNET. Free Software Downloads and Software Reviews. http://download.cnet.com/windows/?tag=hdr;brandnav, Last Accessed March 2012.

[10] Damballa. DAMBALLA Threat Report 2011. Technical report, DAMBALLA Malware Intelligence Labs, 2012.

[11] Danny-Quist. Offensive Computing. http://offensivecomputing.net/, Last Accessed March 2012.

[12] K. Dunham. Malcode context of api abuse. Technical report, SANS Institute, 2011.

[13] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Computer Survey*, 44(2):6, 2012.

[14] R. W. N. Ì. Finn Michael Halvorsen and ard H ÌŁavard Vegge. Zero–Day Malware. Master's thesis, Norwegian University of Science and Technology, Trondheim, 2008.

[15] FireEye. FireEye Advanced Threat Report Second Half, 2011. Technical report, FireEye Malware Intelligence Labs, 2012.

[16] Y. Freund and R. E. Schapire. Large Margin Classification using the Perceptron Algorithm. In *11th Annual Conference on Computational Learning Theory*, pages 209–217, New York, NY, 1998. ACM Press.

[17] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, dec 1999.

[18] Y. Fukushima, A. Sakai, and Y. Hori. A Behavior Based Malware Detection Scheme for Avoiding False Positives. In *Sixth Workshop on Secure Network Protocols*, NPSec 2010, pages 79–84. IEEE, 2010.

[19] G. Hunt and D. Brubacher. Detours: Binary Interception of Win32 Functions. In *Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3*, WINSYM'99, pages 1–9. ACM, 1999.

[20] F. Karbalaei, A. Sami, and M. Ahmedi. Semantic Malware Detection by Deploying Graph Mining. In *Proceedings of the Student International conference for IT Security for the next generation*, pages 1020–1025. City University of HongKong and Kaspersky Academy, 2010.

[21] D. Kirat, G. Vigna, and C. Kruegel. Barebox: efficient malware analysis on bare-metal. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 403–412, New York, NY, USA, 2011. ACM.

[22] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[23] V. Laxmi, M. S. Gaur, P. Faruki, and S. Naval. Peal-Packed Executable AnaLysis. In *Proceedings of International Conference on Advanced Computing Networking and Security*, ADCONS '11. SPRINGER LNCS, 2011.

[24] Microsoft. Windows XP Service Pack–3. http://windows.microsoft.com/en-IN/windows/products/windows-xp.

[25] MICROSOFT CORPORATION. *Microsoft Portable Executable and Common Object File Format Specificaton*, 1999.

[26] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici. Unknown Malcode detection via Text Categorization and the Imbalance Problem. In *ISI*, pages 156–161, 2008.

[27] V. P. Nair, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi. MEDUSA: MEtamorphic malware Dynamic analysis Using Signature from Api. In *5th International Conference on Malicious and Unwanted Software*, MALWARE 2010, pages 263–269. ACM, 2010.

[28] J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998.

[29] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[30] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze. Malware Detection based on Mining API Calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1020–1025. ACM, 2010.

[31] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education Inc. South Asia, Noida, India, 1 edition, MAY 2006.

[32] R. Tian, R. Islam, and L. Batten. Differentiating Malware from Cleanware Using Behavioural Analysis. In *Proceedings of the 3rd international conference on Security of information and networks*, SIN '10, pages 23–30. IEEE, March 2010.

[33] UniversityOfWaikato. WEKA:Data Mining with Open Source Machine Learning Software. http://www.cs.waikato.ac.nz/ml/weka, Last Accessed March 2012.

[34] VirtualBox. An x86,amd64/intel64 virtualization product. https://www.virtualbox.org/wiki/Linux$_D$ownloads.

[35] VirusTotal. VirusTotal - Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/, Last Accessed January 2012.

[36] VX-Heavens. Virus Collections (VXheavens). http://vl.netlux.org/vl.php/, Last Accessed February 2012.

[37] G. Wagener, R. State, and A. Dulaunoy. Malware Behaviour Analysis. *Journal in Computer Virology*, 4(4):279–287, 2008.

[38] WikiPedia. Type-i and type-ii errors. http://en.wikipedia.org/wiki/TypeI and TypeIIerrors.