2015 IEEE International Conference on
Automation Science and Engineering (CASE)
Aug 24-28, 2015. Gothenburg, Sweden

Malware Detection via API calls, Topic Models and Machine Learning

G. Ganesh Sundarkumar, Vadlamani Ravi*, Ifeoma Nwogu, Venu Govindaraju

Abstract—**Dissemination of malicious code, also known as malware, poses severe challenges to cyber security. Malware authors embed software in seemingly innocuous executables, unknown to a user. The malware subsequently interacts with security-critical OS resources on the host system or network, in order to destroy their information or to gather sensitive information such as passwords and credit card numbers. Malware authors typically use Application Programming Interface (API) calls to perpetrate these crimes. We present a model that uses text mining and topic modeling to detect malware, based on the types of API call sequences. We evaluated our technique on two publicly available datasets . We observed that Decision Tree and Support Vector Machine yielded significant results. We performed t- test with respect to sensitivity for the two models and found that statistically there is no significant difference between these models. We recommend Decision Tree as it yields 'if-then' rules, which could be used as an early warning expert system.**

I. Introduction

Cyber security is a significant economic and national security challenge affecting all areas of government and the private sector in today's world. Cybersecurity thus involves the different measures taken to protect computer systems and networks against unauthorized access or attack. Malware takes a primary vehicle to attack the resources. Alazab et al. [1], defined malware as code added, changed or deleted from a program or system, in order to intentionally cause harm or subvert the intended function of the program or system [1]. Every two years, it is estimated that the financial loss to an organization is doubled due to the breach of cyber security and malware specifically is the most common method of this breach [2].

Malware, in general attempts to hide itself from the defensive mechanisms in a system, in order to perform its malicious tasks. Vital protective mechanisms include deploying signature-based antivirus programs. [3]. But, it is becoming more challenging for signature-based malware detection to detect variants of a family of malware [4]. In order to address these challenges, efficient machine learning algorithms are employed to recognize the new, unseen instances of malware, by extracting the efficient features either in a static or dynamic manner from the binary executables [5,6]. The original malicious executables were disassembled into a series of the API calls, they make to the system, which could be handled as text files, rather than executables. We therefore propose a model that takes advantage of the fact that malware can now be analyzed at the text level. We employed topic model (Latent Dirchlet Analysis) as feature selection method. The rest of the paper is organized as follows: In Section II, we present literature review and section III describes the importance of malware API in malware detection. Section IV explains our proposed methodology. The experimental setup and dataset are described in Section V. Section VI presents results and discussion, followed by conclusions in section VI.

G. Ganesh Sundarkumar was M. Tech., student at Institute for Development and Research in Banking Technology and University of Hyderabad, Hyderabad-500046 (AP),India (e-mail: govindaraju.gsk@gmail.com).

* Vadlamani Ravi is Professor and Head, Center of Excellence in CRM and Analytics, Institute for Development and Research in Banking Technology, Castle Hills Road No. 1, Masab Tank, Hyderabad – 500057 (AP), India (Corresponding Author's Phone: +914023294042 and email: rav_padma@yahoo.com).

Ifeoma Nwogu is Research Assistant Professor in Center for Unified Biometrics and Sensors at State University of NewYork (SUNY), Buffalo (email:inwogu@buffalo.edu).

Venu Govindaraju is Distinguished Professor and Director, Center for Unified Biometrics and Sensors at State University of NewYork, Buffalo (email: venu@cubs.buffalo.edu) .

II. Related Work

Since 1996, research on API calls has been carried out. One of the significant contributions is that of Forrest et al., [7], where they leveraged n- grams approach for anomaly detection. Similarly, Reddy & Pujari [8] proposed that byte sequence and byte n-gram could be considered for feature extraction. As the number of resulting features would be very large, they used various methods of feature selection and reported the use of information gain based selection methods as the best in the malware classification.

O'Kane et al., [9] and Chandramohan et al., [10] argue that a major issue with malware classification based on n-gram analysis is that it is performed on an extremely large explosion of features that occurs when n is increased. and hence, they presented a scheme based on "subspace analysis using eigenvector", and claim that this produces a suitable filter for malware detection [9]. Zhuang et al., [4] extracted instructions from API calls and used a hybrid method comprising hierarchical clustering and K-medoids algorithm for malware classification. Shankarapani et al., [12] showed that the calling sequence of Windows APIs reflected the behavior of a particular piece of code and could help identify malware. Kasama et al., [13] developed the notion of observing suspicious behavior based on system calls in a dynamic environment. Bai et al., [14] proposed critical API calling graph (CAG) and extracted CAG from the control flow graph for detecting the malware. Sheen et al., [15] proposed to prune the ensemble which detects the malware using harmony search and most recently, Nissim et al., [16] proposed a novel active learning method to detect new Malware files. Ahmed et al., [17] had used spatio- temporal information in API calls as feature selection for classifying the malware samples. In this paper, we propose a model where LDA and data mining algorithms are employed in tandem.

III. Overview of API Calls in Malware Detection

API level information within the byte code is a convenient method of analyzing software malevolence tendencies, since it conveys substantial semantics about the behavior of the executable from which the API call sequence was generated. More specifically, we focus on critical API calls, their package level information, as well as their parameters. Majority of the Windows-based applications make use of API calls to execute tasks [18, 19] Hence, malware authors readily use API calls as a way to perform malicious actions. Malcode has the ability to use API functions provided under the Win32 environment to execute malicious tasks [20] on the file management system, console, process and thread, or registry (the primary vulnerable targets) [24]. Table I presents a few typical API calls and their behavior.

TABLE I SOME API CALLS AND RESPECTIVE BEHAVIOR [21]

API function calls	***Suspicious behavior description***
GetWindowsDirectory, GetSystemDirectory	Obtain the system directory
FindFirstFile, FindNextFile	Search files to infect
CreateFile, OpenFile, WriteFile, CloseHandle.	File write
GetFileAttributes, etFileAttributes	Modify file attributes
GetFileTime, SetFileTime	Modify time of file
GlobalAlloc, GlobalFree	Distribute global memory
VirtualAlloc, VirtualFree	Distribute virtual memory
RegCreateKey, RegSetValue, RegCloseKey	Load register

IV. Proposed Methodology

A. Preprocessing

Text Mining is the process of finding correct, potentially useful and ultimately understandable knowledge from a large text dataset [22]. It includes tokenization, removal of stop words and stemming [23], then extracting a large number of features, followed by feature selection. Term Frequency-Inverse Document Frequency (TF-IDF) is a common feature extraction method in information retrieval and text mining ([23, 24] .

TF-IDF can be computed as follows:

$$tf * idf = \sqrt{\frac{n_{ij}}{\sum_k n_{kj}}} * \ln\left(\frac{|D|}{d_j : t_i \in d_j + 1}\right)$$

where $n_{i,j}$ is the number of occurrences of term t_i in document d_j, and the denominator is the number of occurrences of all terms in document; d_j and |D| is the total number of documents in a data set, and $\{d_j: t_i \in d_j+1\}$ is document frequency, i.e. the number of documents where term t_i appears. With the selected features, a document-term matrix is built.

B. Feature Selection

To discover the structural properties in the data, various models have been proposed. Of them, the most popular model includes "Topic Model" [26].

A challenging aspect of text document classification problem is the choice of features [27]. Treating individual words as features yields a rich but very large feature set. Therefore, we used Latent Dirichlet Allocation (LDA) to select the features and build a vector space model. LDA is a generative Bayesian graphical model which represents each document as a collection of topics from a mixture model, where each mixture component is a topic [28]. Each topic consists of words that co-appear in a document regularly and is the representation of the document. The input to LDA is the standard bag of words representation of a collection of documents, where a sparse vector of |W| represents document collection D, where W is the unique list of words [26]. It models each document d as a mixture θ_d over T Latent topics, where each topic $\emptyset_t$ is drawn from a Dirichlet distribution with parameter β, and θ_d is drawn from symmetric Dirichlet distribution with parameter α. In general, a document can deal with multiple topics and the words in the document indicate the particular set of topics it addresses [26]. LDA can therefore help to uncover the hidden structure of the document. Given a matrix of order m × n where a row represents a document and each column represents the words, LDA generates a word-topic matrix ϕ, in which $\phi^{(j)} = p(w|\ z=j))$ is the multinomial distribution of words over the topic j.

LDA in general is accomplished of the following process [27]

a) Choose N words ~ Poisson (ε)
b) Choose θ ~ Dirchlet (α)
c) For each of the N words w_n :
   - Choose a topic Z_n ~ Multinomial (θ)
   - Choose a word W_n from p(W_n | Z_n, β), a multinomial condition on topic Z_n

A k-dimensional Dirichlet random variable θ can take values in the (k−1)-simplex, and has the following probability density on this simplex:

$$p(\theta|\alpha) = \frac{\mathrm{r}(\sum_{i=1}^{k}\alpha_i)}{\pi_{i=1}^{k}\mathrm{r}(\propto_i)}\ \theta_1^{\propto_i - 1} \dots \theta_k^{\propto_k - 1}$$

where the parameter α is a k-vector with components $\alpha_i > 0$, and where r(x) is the Gamma function. The Dirichlet is a convenient distribution on the simplex—it is in the exponential family, has finite dimensional sufficient statistics, and is conjugate to the multinomial distribution.

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics **z**, and a set of N words **w** is given by:

$$p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\theta|\alpha)\ \pi_{n=1}^{N} p(z_n|\theta) p(w_n|Z_n, \beta)$$

where $p(Z_n|\theta)$ is simply θ_i for the unique i such that $z^i_n = 1$. Integrating over θ and summing over z, we obtain the marginal distribution of a document:

$$p(w|\alpha,\beta) \;= \int p(\theta|\alpha)(\pi_{n=1}^{N}\sum_{z_n} p(Z_n|\theta)\, p(w_n|Z_n,\beta))d\theta$$

Finally, taking the product of the marginal probabilities of single documents, we obtain the probability of a corpus:

$$p(D|\alpha,\beta) = \pi_{d=1}^{M}\int p(\theta|\alpha)(\pi_{n=1}^{N_d}\sum_{z_{dn}} p(Z_n|\theta_d)\, p(w_{dn}|Z_{dn},\beta))d\theta$$

V. EXPERIMENTAL SETUP

A. Data set description

We analyzed two datasets. In the case of first one, named Dataset 1, we obtained the malware (320) and benign (68) API call logs sequences from CSMINING group [30] available in public domain. It is obviously unbalanced. Further, we carried out experimentation on another dataset [17], named Dataset 2, by us. The dataset consists of 416 malware samples and 100 benign executables. Of the 416 malware samples, Trojans are 117, 165 are virus samples and worms are 134. The corresponding API call logs have been obtained from [31].

B. Experimental Setup

In order to apply the text mining techniques discussed in the preceding section for Dataset 1, we computed the TF-IDF values and obtained 313 unique tokens or variables. Further, from the 388 text files comprising a series of API calls, we derived the topic-word matrix as discussed above using LDA. From the matrix ϕ, $\phi(i)(j) = p(w = w_i \mid z = j)$ is the value of i– j^{th} cell of the matrix, defined as the probability of i^{th} word belonging to the j^{th} topic. We then sorted the column from each topic and extracted the top word from each topic and represented it as the efficient feature set for classification in the Document-Term matrix. We tested with the number of topics being 5, 8, 11 and 14, and discovered that 8 topics yielded the best results. We empirically chose 8 topics as a suitable number, out of repetitive experiments with varying numbers of topics, for the given dataset. Furthermore, for the hyper-parameters for LDA [32], we have set alpha=6 and beta value as 0.1. We trained our LDA model for 100 iterations. Further, Dataset1 is unbalanced.

Features extracted by the proposed methodology are presented in Table V. The document term matrix having 8 features is fed to classifiers. To further validate the proposed methodology, we carried out the experimentation on another dataset, originally available in [31] and previously analyzed by Ahmed et al. [17]. With the proposed methodology of LDA as feature selection, we selected 7 features. We ran the LDA for the entire corpus consisting of 516 documents of the executables with the hyper parameters of alpha as 5, beta as 0.1, number of topics as 7 and number of iterations set to 100. We have chosen these parameters after a repeated set of experiments.

With the thus obtained 7 features from the top most features from 7 topics, we formed the document term matrix. In order to compare our results with that of Ahmed et al. [17], we followed the same methodology followed by them. We segregated the samples into four classes and developed 3 binary classification tasks viz., Benign vs. Trojan, Benign vs. Virus and Benign vs. Worm. We followed 10 fold cross validation throughout the process similar to [17].

VI. RESULTS AND DISCUSSION

We employed Neuroshell 2.0 [33] and RapidMiner [35] for conducting the experiments. As regards the Dataset 1, Table II presents a comparison of our results with that of the most recent work [34], where they employed oversampling and mutual information for malware identification. , From Table II, we noticed that the proposed approach significantly improved results despite the data being imbalanced.

They achieved the sensitivity of 100 % using SVM with specificity being 0. In the proposed approach, we observed MLP, SVM, GMDH, DT and RF yielded a sensitivity of 98.61%, 96.87%, 94.12% 92.95% and 91.5% respectively. With respect to Area Under ROC Curve (AUC-computed as 50* (sensitivity + Sensitivity)), DT and GMDH turned out to be strongest performers. However, we preferred DT as it yielded rules while building the model. Rules extracted from DT are presented in Table VI. Further, in the Dataset 2 too (see Tables III and IV), the proposed methodology yielded excellent results in identifying various flavors of the malware when classified against the benign samples. When compared to [17] the proposed methodology performed well with significantly high results without any combinatorial feature selection performed in [17]. DT and SVM outperformed the results of [17] in identifying the Trojan, Virus, and Worm versus Benign Samples. Rules extracted from DT for dataset 2, are presented in Table VII.

Finally, we performed t- test (with respect to sensitivity) between DT and SVM at 18 degrees of freedom and 1% level of significance. We found statistically there is no significant difference between DT and SVM. However,, we prefer DT as it yielded rules for classification whereas SVM is a black box. Rules extracted for 3 binary classification problems are presented in Table VI. In [17], while Naïve Bayes performed well compared to the rest of the classifiers, it is also a black box. We finally conclude that LDA as feature selection in combination with data mining algorithms is a viable option for the malware detection.

VII. CONCLUSION

We propose a novel approach to detect malware that uses text mining for feature extraction; topic modeling for feature selection using Latent Dirichlet Allocation to detect malware based on the types of API call sequences and machine learning for classification. We evaluated the effectiveness of our technique on a two publicly available datasets For the Dataset 1, despite the imbalance, LDA as a feature selection method yielded significant results when compared with the earlier approaches including Sundarkumar and Ravi [34]. Further in the Dataset 2 too, the proposed methodology performed well. As DT and SVM are found to be statistically significantly not different, we prefer DT as it yields rules for constructing the model.

TABLE II AVERAGE RESULTS OF 10 FOLDS ON DATASET 1

	Method of [34] with 10 features				***Proposed Methodology with 8 features***			
Model	***Accuracy***	***Sensitivity***	***Specificity***	***Area Under Curve***	***Accuracy***	***Sensitivity***	***Specificity***	***Area Under Curve***
SVM	70.27	100	0	5000	88.15	**96.87**	41.66	6926.5
PNN	66.13	78.43	41.57	6000	82.68	67.9	87.07	7748.5
D.T	73.43	80.54	59.09	6981.5	88.67	**92.95**	54.99	**7397**
GMDH	70.47	94.37	27.93	6115	89.46	**94.12**	50.462	**7229.1**
MLP	70.05	90.01	23.45	5673	87.64	**98.61**	29	6380.5
RF	-	-	-	-	85	**91.5**	50	7078

TABLE III DETECTION ACCURACY ON DATASET 2 BY [17]

Alg	**IB_K**	**J_{48}**	**NB**	**RIPPER**	**SMO**	**Avg**
			Trojan			
Spatial (S)	**92.6**	91.0	77.7	87.1	76.0	84.9
Temporal (T)	95.8	90.3	94.5	93.2	**97.0**	94.2
S&T	96.3	90.8	**96.6**	95.9	97.0	95.3
			Virus			
Spatial (S)	91.5	94.0	85.9	**94.9**	91.5	91.6
Temporal (T)	93.7	95.6	**97.4**	94.5	97.1	95.7
S&T	95.4	96.3	**99**	95.4	96.8	96.6
			Worm			
Spatial (S)	97.3	96.1	94.6	**96.3**	80.6	93.6
Temporal (T)	94.2	**96.8**	96.2	95.3	96.3	95.8
S&T	95.8	96.6	**98.4**	97.5	96.3	96.9
Avg	96.3	94.7	**98.0**	96.0	96.8	96.3

TABLE IV. DETECTION ACCURACY OF PROPOSED METHODOLOGY ON DATASET 2

Category	**D.T**	**SVM**	**MLP**	**GMDH**	**PNN**	**RF**
Trojan	**98.182**	**98.462**	93.63	93.05	92.8	85
Virus	**98.75**	**98.75**	89.35	96.712	88.23	95
Worm	**98.46**	**98.7**	89.5	93.72	87.98	92.5
Average	**98.46**	**98.63**	90.8	94.49	89.67	90.3

TABLE V FEATURES EXTRACTED FOR THE MALWARE DATASETS

10 Features obtained using MIFS for Dataset 1[34]	**8 Features obtained using LDA for Dataset 1**	**7 Features obtained using LDA for Dataset 2**
GetCurrentProcess, GetCurrentProcessId, GlobalMemoryStatus, IsBadReadPtr, IsBadStringPtrW, IsBadWritePtr, LocalAlloc, LocalFree, RegOpenKeyW and *RegSetValueExW*	*GetProcAddress, GetCurrentThreadId, GlobalAlloc, IsBadReadPtr, Sleep, HeapFree, HeapSize, HeapAlloc*	*HeapSize, HeapFree, IsBadReadPtr, HeapAlloc, GetCurrentThreadId, GetCurrentProcessId, LocalAlloc*

TABLE VI RULES OBTAINED BY DT FOR BEST FOLD IN DATASET 1

#	Antecedent	Consequent
1.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress = 0.068 and IsBadReadPtr=0.369 and GetCurrentThreadId =0.027	Malware
2.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress = 0.068 and IsBadReadPtr=0.369 and GetCurrentThreadId >0.027 and sleep =0.001	Benign
3.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress = 0.068 and IsBadReadPtr=0.369 and GetCurrentThreadId =0.027 and sleep =0.003	Malware
4.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress = 0.068 and IsBadReadPtr=0.369 and GetCurrentThreadId =0.027 and sleep >0.003	Benign
5.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress = 0.068 and IsBadReadPtr>0.369	Malware
6.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc = 0.008 and GetProcAddress > 0.068	Malware
7.	If HeapSize >0.07 and HeapAlloc = 0.536 and GlobalAlloc > 0.008	Benign
8.	If HeapSize >0.07 and HeapAlloc > 0.536 and sleep=0.000	Malware
9.	If HeapSize >0.07 and HeapAlloc > 0.536 and sleep>0.000	Benign
10.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress = 0.001	Malware
11.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress > 0.001	Benign
12.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress > 0.063	Malware
13.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress = 0.147	Malware
14.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress > 0.147	Benign
15.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetCurrentThreadId = 0.000 and GetProcAddress > 0.174	Malware
16.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetProcAddress = 0.027 and GetCurrentThreadId > 0.000	Benign
17.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetProcAddress = 0.027 and GetCurrentThreadId = 0.072	Benign
18.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetProcAddress = 0.027 and GetCurrentThreadId > 0.072	Malware
19.	if HeapSize = 0.007 and GlobalAlloc = 0.003 and GetProcAddress > 0.027 and GetCurrentThreadId > 0.044	Malware
20.	if HeapSize = 0.007 and GlobalAlloc > 0.003 and GetProcAddress > 0.578	Malware
21.	if HeapSize = 0.007 and GetProcAddress = 0.631	Malware
22.	if HeapSize = 0.007 and GetProcAddress > 0.631	Benign
23.	if HeapSize = 0.007 amd GetProcAddress > 0.674	Malware

TABLE VII RULES OBTAINED BY DT FOR BEST FOLD IN DATASET 2

DT Rules extracted for Benign vs Virus		
#	**Antecedent**	**Consequent**
1.	If(HeapSize<= 0.004),	Virus
2.	If(HeapSize>0.004 and GetCurrentProcessId <= 0.001 and GetCurrentThreadId <= 0.001)	Benign
3.	If(HeapSize>0.004 and GetCurrentProcessId <= 0.001 and GetCurrentThreadId > 0.001 and HeapSize >0.313),	Virus
4.	If(HeapSize>0.004 and GetCurrentProcessId <= 0.001 and GetCurrentThreadId > 0.001 and HeapSize <= 0.313),	Benign
5.	If(HeapSize<=0.044 and GetCurrentProcessId <= 0.004)	Benign
6.	If(HeapSize<=0.044 and GetCurrentProcessId > 0.004)	Virus
7.	If(HeapSize>0.044 and GetCurrentthreadId <= 0.007)	Benign
8.	If(HeapSize>0.044 and GetCurrentthreadId > 0.008)	Benign
9.	If(HeapSize>0.044 and GetCurrentthreadId <= 0.008)	Virus
DT Rules extracted for Benign vs Trojan		
1.	If(HeapSize<= 0.001 and GetCurrentThreadId > 0.006 and HeapAlloc >0.001)	Benign
2.	If(HeapSize<= 0.001 and GetCurrentThreadId > 0.006 and HeapAlloc <=0.001)	Trojan
3.	If(HeapSize<= 0.001 and GetCurrentThreadId > 0.004 and GetCurrentThreadId < 0.006 and HeapAlloc >0.001)	Trojan

4.	If(HeapSize<= 0.001 and GetCurrentThreadId > 0.004 and GetCurrentThreadId < 0.006 and HeapAlloc <=0.001)	Benign
5.	If(HeapSize<= 0.001 and GetCurrentThreadId <= 0.003)	Trojan
6.	If(HeapSize>0.001 and HeapFree <= 0.003)	Trojan
7.	If(HeapSize>0.001 and HeapFree > 0.003)	Benign
	DT Rules extracted for Benign vs Worm	
1.	If(HeapSize <= 0.001)	Worm
2.	If(GetCurrentProcessId<= 0.001 and GetCurrentThreadId >0.016 and HeapSize <=0.138)	Worm
3.	If(GetCurrentProcessId<= 0.001 and GetCurrentThreadId >0.016 and HeapSize <=0.451)	worm
4	If(GetCurrentProcessId<= 0.001 and GetCurrentThreadId >0.016 and HeapSize >0.451)	Worm

REFERENCES

[1] A. Altaher, S. Ramadass, and A. Ali, " Computer virus detection using features ranking and machine learning", Journal of Applied Sciences Research, 7(9), 2011.

[2] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls", ACM Symposium on Applied Computing, 2010, pp. 1020-1025.

[3] Y. Ye, T. Li, Q. Jiang, and Y. Wang, "Cimds: adapting post processing techniques of associative classification for malware detection", Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 40(3), 2010, pp.298-307.

[4] W. Zhuang, Y. Ye, Y. Chen, and T. Li, "Ensemble clustering for internet security applications", Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 42(6),2012, pp.1784-1796.

[5] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap", In 5th ACM workshop on Security and artificial intelligence, 2012, pp.3-14.

[6] M. Siddiqui, M. C. Wang, and J. Lee, "Data mining methods for malware detection using instruction sequences", Proceedings of Artificial Intelligence and Applications, 2008.

[7] S. Forrest, S. A. Hofmeyr, A. Somyaji, T. A. Longstaff, " A sense of self for unix process", IEEE Symposium on Security and Privacy, 1996, pp:120-128.

[8] D. K. S. Reddy, and A. K. Pujari, "N-gram analysis for computer virus detection", Journal in Computer Virology, 2(3), 2006, pp. 231-239.

[9] P. O'Kane, S. Sezer, K. McLaughlin, and Eul Gyu Im, "Svm training phase reduction using dataset feature filtering for malware detection", Information Forensics and Security, IEEE Transactions on, 8(3), 2013, pp. 500-509.

[10] M. Chandramohan, H. B. K. Tan, L.C. Briand, L. K. Shar, and B. M. Padmanabhuni, "A scalable approach for malware detection through bounded feature space behavior modeling", In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, pp. 312-322.

[11] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high- dimensional distribution", Neural Computation, 13(7),2001, pp. 1443- 1471.

[12] M. Shankarapani, K. Kancherla, S. Ramammoorthy, R. Movva, and S. Mukkamala, "Kernal Machines for malware classification and similarity analysis", In: International joint conference on Neural Networks , 2010, pp: 1-6.

[13] T. Kasama, K. Yoshioka, D. Inoue, and T. Matsumoto, "Malware detection method by catching their random behavior in multiple executions", In: Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on, 2012, pp. 262- 266.

[14] L. Bai, J. Pang, Y. Zhang, W. Fu, and J. Zhu, "Detecting malicious behavior using critical api-calling graph matching", In: Information Science and Engineering (ICISE), 2009 1st International Conference on, 2009, pp. 1716-1719.

[15] S. Sheen, R. Anitha, and P. Sirisha, "Malware detection by pruning of parallel ensembles using harmony search", Pattern Recognition Letters, vol. 34(14), 2013, pp.1679-1686.

[16] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Novel active learning methods for enhanced pc malware detection in windows os", Expert Systems with Applications, 41(13), 2014, pp. 5843-5857.

[17] F. Ahmed, H. Hameed, M. Z. Shafiq, 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, 2009, pp: 55- 62.

[18] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on api function calls and their arguments", In: Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on, 2012, pp. 563-568.

[19] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis", In: Computer Security-ESORICS2008, pp. 481-500.

[20] A. Altaher, S. Ramadass, and A. Ali, "Computer virus detection using features ranking and machine learning", Journal of Applied Sciences Research, 7(9), 2011.

[21] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behavior by the extraction of api calls", In: Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second, pp. 52-59.

[22] P. Losiewicz, D. W. Oard, and R. N. Kostoff, " Textual data mining to support science and technology management", Journal of Intelligent Information Systems, 15(2), 2000, pp. 99-119.

[23] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets", In: SDM, SIAM, volume 3, 2003, pp. 59- 70.

[24] K. Lee, D. Palsetia, R. Narayanan, M. M. A. Patwary, A. Agrawal, and A. Choudhary, "Twitter trending topic classification", In: Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on, pp. 251-258.

[25] L. Ronghui, Z. Jianguo, and W. Xiang, "A method for clustering e-business contents", In: Information Engineering (ICIE), 2010 WASE International Conference on, volume 2, pp. 43- 46.

[26] N. Rasiwasia, N. Vasconcelos, "Latent Dirichlet Allocation models for Image classification", In: IEEE Tansactions on Pattern analysis and Machine Intelligence, 2013, pp: 2665-2679.

[27] D. M. Blei, A. Y. Ng, and M. L. Jordan, "Latent dirichlet allocation", Journal of machine Learning research, 3, 2003, pp. 993-1022.

[28] D. Gujraniya, and M. N. Murty, "Efficient classification using phrases generated by topic models", In: Pattern Recognition (ICPR), 2012 21st International Conference on, pp. 2331- 2334.

[29] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behavior by the extraction of api calls", In: Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second, pp. 52-59.

[30] Malicious datasets csmining group, 2014. URL http://www.csmining.org/index.php/ malicious-software-datasets-.html.

[31] http://nexginrc.org/Datasets/Default.aspx

[32] T. L. Griffiths, and M. Steyvers, "Finding scientific topics", In: Proceedings of the National academy of Sciences of the United States of America, 101(Suppl 1), 2004, pp. 5228-5235.

[33] Neuroshell: http://www.neuroshell.com/

[34] G. G. Sundarkumar, and V. Ravi, "Malware detection by text and data mining", In: Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on, pp. 1-6, Dec(2013).

[35] RapidMiner: http:// www. http://rapidminer.com/