Journal of Computer Virology and Hacking Techniques

Malware Detection and Classification Using Community Detection and Social Network Analysis

--Manuscript Draft--

Manuscript Number:	JICV-D-20-00046R1
Full Title:	Malware Detection and Classification Using Community Detection and Social Network Analysis
Article Type:	Original Paper
Corresponding Author:	N Balakrishnan, Ph.D. Indian Institute of Science Bangalore, Karnataka INDIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Indian Institute of Science
Corresponding Author's Secondary Institution:	
First Author:	Varshini Reddy
First Author Secondary Information:	
Order of Authors:	Varshini Reddy
	Naimisha Kolli
	N Balakrishnan
Order of Authors Secondary Information:	
Funding Information:	Ministry of Communication and Information Technology, Government of India (MITO/104) — Prof N Balakrishnan
Abstract:	Despite the efforts of antivirus vendors and researchers to overcome the threat of malware and its growth, malware remains a rampant problem causing significant economic and intellectual property loss. Malware developers evade commercial detection tools by introducing minor code changes and obfuscation, leading to the creation of variants of known malware families. The volume of malware variants being introduced is increasing every day, resulting in the need for new methods to detect and classify malware with high scalability in less time. To this end, we propose a novel technique that exploits community detection properties and social network analysis concepts. The proposed method is based on system call graphs obtained by extracting the system calls found in the execution of the malware files. To study the inherent characteristics of different malware families, we extract features conforming to community and social network properties and use them for classification. A set of 5 models ranging from using only OS-level actions, to the model that includes community-level features and social network features have been presented. The highest performance has been shown to arise when community-level features and social network features were used in combination with malware class-level features. A suite of 9 machine learning algorithms have been used, and the results have been compared. Our evaluation results demonstrate that our combined approach outperforms many previously used methods in malware detection and classification, being able to achieve precision, recall, and accuracy of more than 0.97 using Multilayer Perceptron and k-Nearest Neighbors.
Response to Reviewers:	Reviewers Comments and response/changes/ clarifications. Reviewer #1: This paper proposes an X86 malware detection and family classification strategy based on machine learning. In particular, the authors resort to a combination

	of system call-based features that rely on the related call graph generation. The extraction of these features is inspired by traditional system calls count, social networks (e.g., degree and closeness centrality) and community (e.g., number of members and edges) graphs analysis. The authors tested various combinations of such features against nine classifiers to detect nine malware families, showing that the simultaneous use of social networks and community significantly improved the performance indicators such as accuracy, precision, and recall. Our Response: Thank you sir. Overall, the paper is quite clear and nice to read. The idea of exploring the system call graph properties is of interest in the context of malware detection. At the same time, I feel that this paper features several problems that cannot be solved with a major revision but require a thorough rewriting, as well as additional work on the paper claims and motivation. In the following, I list the main issues of this work: 1. The paper is not clearly organized. The introduction essentially describes the related work in the field, but that description should go to a different section. The introduction should expand the motivation for the detection problem examined by the authors. Additionally, this paper would benefit from a new section that describes the technical background related to the Windows malware structure. Response: In the revised manuscript we have brought out the motivation very clearly in the introduction section and we have shifted the related work part to a separate section. Additions have been made to Section 3.3 to answer the above query. 2. Concerning the related work, this paper is missing many references. The authors should thoroughly discuss the related work in the field (not only in the contest of graph analysis) to reinforce their work motivation. I would recommend "Y.Ye et al.,'A survey of malware detection using data mining techniques', in ACM Computing Surveys" as a starting point. Additionally, the authors should clearly discuss works such as "J.Jang et al., Mal-Netminer: Malware Classification Approach Based on Social Network Analysis of System Call Graph, in Arxiv" which, albeit not directly published, contain many elements in common to the proposed approach. Response: We have added both the above references appropriately and have also added more references and descriptions to answer the later queries on Context-aware malware for anti-sandbox and packing in malware. 3. The biggest problem of this work is that it lacks proper motivation and novelty. It is unclear what is the "original" proposal made by the authors. By reading the paper, it seems that the tested proposal is a combination of already existing approaches (e.g., in Section 4.3, the authors mention the Louvian algorithm), but no original features have been devised. Additionally, the authors should better explain why they chose exactly these categories of features. At the current state, this paper seems more a test of feature combinations without a solid motivation. This has been addressed in Section 3.4.3 in better detail. Now the motivation is brought out more clearly as also the novelty of the work compared to what is available in the literature. 4. The experimental section lacks in-depth discussion, apart from the numerical results. The parameters with which the classifiers were trained are unclear. There are no indications of the computational performances of the proposed method. Additionally, the sole use of performance metrics such as F1-score is not enough to state that a combination of features is better than others. What about the feature resilience against possible adversaries? What about using a different train-test split (e.g., 50-50?). What are the families where the classifiers fail the most? Have the authors considered the presence of packing? (though dynamic analysis inherently avoids packing, it would be interesting to check if there are system calls invoked during the decompression phase). Response: This has been addressed in more detail in the Sec 4.0 Results and Discussions. We have used not only the F-Score but also the AUC as well as accuracy. Now this is also discussed in the Section. 4.3. Regarding possible adversaries, our approach uses not just the system level features but also the

	community-level and social network and class level features. Adversarial attacks on these not that easy. In any case, we have suggested this as our future work. Different combinations of train and data split have been tried in this as well our other research work. If you are above 50% in training set that too with proper split within that between benign and malware files, the results will be and in fact we found it to be similar. To address the question of which families for which the classifier fails, we have added the confusion matrix in Table 2. Which clearly shows the misclassification in most cases is very small or negligible. We have not found many files with packing. In the few cases (only two) that we found, we had detected the decript/decompression calls and have handled the packed portion separately. This explanation and the pointers to several work that already exist in the literature have been made in Sec. 3.6 Evaluation. Reviewer #2: Very interesting approach to detect malware based on Social Network Analysis methods. Good experimental results. But for some classes of malware it looks like would be some problems. Some malware can detect running in sandbox and generate flooding syscalls. Maybe you should test some binary instrumentation technics to extract systemcalls. Response: We checked all our files extensively. We did not find any malware file that ducks the sandbox and is context-aware. However, the issue is something that the community must be aware of. In view of this, we have added a description of this problem and the solutions giving two additional references in Sec 3.6 Evaluation.

Malware Detection and Classification Using Community Detection and Social Network Analysis

Varshini Reddy[1] Naimisha Kolli[2] N Balakrishnan[3]

[1]Research Assistant, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, 560012, India.

[2]Research Associate, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, 560012, India. Email:

[3]Honorary Professor, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, 560012, India.

Corresponding author:

Prof. N Balakrishnan
Supercomputer Education and Research Centre,
Indian Institute of Science

Email: balki@iisc.ac.in

ORCID: 0000-0002-9744-3371

Abstract

Despite the efforts of antivirus vendors and researchers to overcome the threat of malware and its growth, malware remains a rampant problem causing significant economic and intellectual property loss. Malware developers evade commercial detection tools by introducing minor code changes and obfuscation, leading to the creation of variants of known malware families. The volume of malware variants being introduced is increasing every day, resulting in the need for new methods to detect and classify malware with high scalability in less time. To this end, we propose a novel technique that exploits community detection properties and social network analysis concepts. The proposed method is based on system call graphs obtained by extracting the system calls found in the execution of the malware files. To study the inherent characteristics of different malware families, we extract features conforming to community and social network properties and use them for classification. A set of 5 models ranging from using only OS-level actions, to the model that includes community-level features and social network features have been presented. The highest performance has been shown to arise when community-level features and social network features were used in combination with malware class-level features. A suite of 9 machine learning

algorithms have been used, and the results have been compared. Our evaluation results demonstrate that our combined approach outperforms many previously used methods in malware detection and classification, being able to achieve precision, recall, and accuracy of more than 0.97 using Multilayer Perceptron and k-Nearest Neighbors.

Keywords: Malware, Social Network Analytics, Community Detection, Machine Learning, Classification

Declarations

Funding: This work was supported by the grants from the Ministry of Communication and Information Technology of the Government of India under the Information Security and Awareness (ISEA) program.

Conflicts of interest/Competing interests: There is absolutely no conflict of interest amongst the authors, the organization that they work for or with any private or public entities.

Availability of data and material: Most of the codes used are in public domain such as CalmAV, Cuckoo Sandbox and the ML Packages. The feature extraction and feature reduction codes are custom built.

Noname manuscript No.
(will be inserted by the editor)

Malware Detection and Classification Using Community Detection and Social Network Analysis

Varshini Reddy Bogolu · Naimisha Kolli · N Balakrishnan

Received: date / Accepted: date

Abstract Despite the efforts of antivirus vendors and researchers to overcome the threat of malware and its growth, malware still remains to be a rampant problem causing significant economic and intellectual property loss. Malware developers evade commercial detection tools by introducing minor code changes and obfuscation, leading to the creation of variants of known malware families. The volume of malware variants being introduced is increasing every day, resulting in the need for new methods to detect and classify malware with high scalability in less time. To this end, we propose a novel technique that exploits community detection properties and social network analysis concepts. The proposed method is based on system call graphs obtained by extracting the system calls found in the execution of the malware files. To study the inherent characteristics of different malware families, we extract features conforming to community and social network properties for classification. Our evaluation results demonstrate that our approach outperforms many previously used methods in malware detection and classification, being able to achieve a precision, recall and accuracy of more than 0.97 using the Multilayer Perceptron, Random Forest and k-Nearest Neighbors algorithms.

Keyword —malware, social network analytics, community detection, machine learning

1. Introduction

With the increasing sophistication and enhanced incidences, malware detection, classification and analysis have become ever more challenging. The research community has also been continuously innovating with more advanced techniques particularly machine learning-based methods for malware analysis. Internet Society's Online Trust Alliance's 2018 report indicates that the world economic impact of cybercrime in 2018 alone was $45 billion, with overall adding up to $600 billion (Internet Society's Online Trust Alliance 2018 Cyber Incidents & Breach Trends Report) and is predicted to reach $6 trillion by 2021. Apart from monetary losses, system vulnerabilities are used for criminal purposes and are a threat to personal safety. The first step towards defending computers is the detection of malware. Malware is an abbreviation of the words malicious and software. The term refers to software that is deployed with malicious intent. Various types of malware are classified - based on their threat level and activity - into malware families. Some of these are worms, viruses, trojans, backdoors and ransomware. According to the statistics given by SonicWall, 2018 had a record-breaking 10.52 malware attacks, while logging 4.8 billion malware attacks by mid-2019. [1] indicates that out of all the malware involved in the malware attacks in 2019, 35.89% was zero-day malware and 64.1% was known malware. Latest reports presented by AV-TEST registers 350,000 new malicious programs (malware) and Potentially Unwanted Applications (PUA) per day. The aim of our work begins with this step, i.e. detection and classification of malware files.

Significant research for the malware detection based on machine learning techniques for portable executables (PE-32) in Windows environment has been proposed in the literature in [2-5]. The primary idea is to extract individual malware level features from these PE-32 files such as symbol frequencies, opcode 1-gram register's usage, API function calls, segment counts, instruction traces, imported functions in the PE header, function length frequency, PE file metadata and then predict the probability of a particular file as malware or not based on these extracted features from the file. An obvious drawback of the process and methodologies adopted in this line of research is that they focus mostly on the individual malware features and do not take advantage of the family similarity between malwares that are well represented by their community and social attributes. Recent research has shown the advantages of using social network analysis [6,7] for malware classification. The idea [6] is based on the fact that network structures are ubiquitous in nature. In [6], API call sequence graphs are used for the android malware family classification. The authors in [7] exploit the social networking features such as degree distribution, degree centrality, average distance, clustering coefficient, network density driven from the graph structure of the malware system calls. It deals with the underlying system call graphs of the PE-32 files. In the call graph, the nodes represent the system calls and the edges depict the sequence of calls. The calls are extracted from the code section of the PE-32 files. The main assumption of this line of work [6,7] is that malwares in the same family exhibit similar structures in their sequence of system calls and hence a significant improvement in prediction performance can be achieved by exploiting features extracted from these system call graphs.
In our present work, we recognize the importance of the

role played by the system call graph similarities to better understand the underlying behavior of malware files and incorporate novel features of these call graphs along with the traditional individual file features. The motivation for this work is to explore the combined use of not only the social network features as in [6,7] with system level features but also the community level features. To this end, we propose a hybrid model that not only combines the traditional features extracted from the PE-32 files along with social network and community features extracted from the system call graphs. We demonstrate through experiments that the addition of combined novel features will be able to capture subtle changes in malware behavioral patterns and thus improve the performance of malware detection algorithms.

To illustrate the effectiveness of the proposed technique, a dataset consisting of 10 classes wherein 9 classes depict various malware families and another belonging to the benign class with a total of 60,000 files of system calls has been used. Of the 60,000 files, more than 30,000 files are from [8]. The features extracted have been classified using Logistic Regression, Naive Bayes, k-Nearest Neighbours, Support Vector Machine (SVM), Decision Tree, Gradient Boosted Tree, AdaBoost, Random Forest, and Multilayer Perceptron. Our approach has resulted in a classification accuracy of 0.973, an F-Score of 0.977 and an AUC score of 0.978 for our Multilayer Perceptron architecture. Further, the other algorithms employed in this paper have shown similar performance levels.

The rest of the paper is organized as follows. Section 2 gives detailed background information used in the paper. Section 3 describes the methodology. The experimental results are given in Section 4 while Section 5 summarizes the conclusions.

2. Related Work

An excellent overview of machine learning techniques used so far for the analysis of portable executables in Windows environments can be found in [2-4,7-9]. The survey in [2-4,7-9] provides an overview of the way machine learning has been used so far in the context of malware analysis for the analysis of Portable Executables by systematizing the surveyed papers according to the detection approach, their objectives, feature set, the machine learning techniques employed and the accuracy obtained. Authors in [2] additionally introduce the novel concept of malware analysis economics, regarding the study of existing trade-offs among key metrics, such as analysis accuracy and economical costs. Kolosnjaji et. al. [10] have used a neural network based on convolutional and recurrent network layers to obtain the best features for classification using system call sequences. This way a hierarchical feature extraction architecture is generated that combines convolution of n-grams with full sequential modelling.

Use of social network analysis has been proposed by Kim et.al [6] for android malware classification. They have proposed a dynamic malware analysis method that uses Natural Language Processing (NLP) concepts on API system calls and has shown that Linear Support Vector Machines (SVM) optimized by Stochastic Gradient Descent and the traditional Coordinate Descent on the Wolfe Dual form of the SVM are effectively achieving an accuracy as high as 96% with 95% recall score. Social network analysis-based classification of malware families using system call graphs have been reported by Jang et. al. [7].

Venkatesh et. al. [11] have used community detection algorithms and graph analysis to detect structured P2P botnets. Bhattacharya and Goswami have proposed the community-based feature reduction technique for Android malware detection [12]. Kim et. al. [13] have proposed a method using community detection algorithms to classify Android malware families based on common behavioral characteristics of malware families and show improved performance. Du et. al. [14] present a new malware detection method that automatically divides a function call graph into community structures. The features of these community structures are then used to detect Android malware. They have also shown a reduction in the computation time by improving the Girvan-Newman algorithm [15]. Fan et al. in [16] propose a novel approach that constructs frequent subgraphs to represent the common behaviors of Android malware in the same family for familial classification

3. Malware Classification Modeling

3. 1 Problem Statement

Malware detection problem can be defined as a supervised classification problem. The aim of this work is to build a malware detection model based on system level, social networking and community features extracted from the system call graphs.

3. 2 Basic Framework

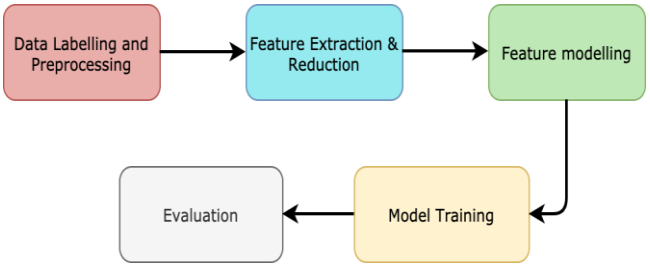

Fig. 1 Block diagram for the proposed method

The basic framework is depicted in Figure 1. In our proposed method, we consider a 10-class classification problem, consisting of 1 benign and 9 malware families. First, the system calls are extracted as a graph from the PE-32 executable files. Then the features are computed based on the system call graph and feature engineering is performed to extract the feature set for each malware and benign sample. This will form the input for the classifier. The steps involved in the feature extraction from system calls are shown in Figure 2. In the following sections, we describe each of these steps in detail.

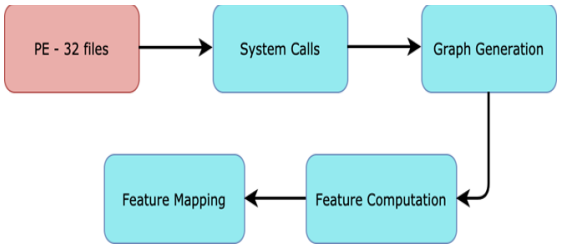

Fig. 2 Steps involved in feature extraction from system calls

3. 3 Dataset Labeling and Preprocessing

The first step in our proposed model is data preprocessing. The dataset used in this paper consists of 70,302 Portable Executable (PE32) files. The dataset was collected from various sources old and new so that one could possibly include malwares which were recirculated after obfuscation. The distribution of data set is around 30839 files from ref. [8], and the rest are from public domain datasets from kaggle competitions, vx heaven, virus share, app.any.run/submissions/ and Malheur dataset. Labelling of the dataset is done by passing the files through a virus scan test using the Clam-AV antivirus software. Based on the results of the scan, duplicate files were removed and the remaining 70,186 PE32 files were classified into 14 classes, consisting of 13 malware families and one benign class. Of these, 4 malware classes had less than 300 files each and hence have been removed, leaving only 10 classes, one benign and 9 malware families and each having a uniform size of 6000 files each. These 60,000 files with classes - benign, Win.Downloader, Win.Spyware, Trojan.Virut, Trojan.Agent, Trojan.Generic, Dropper.Agent, Worm.Allaple, Trojan.Udr, Trojan.Delf - form our dataset. The system calls invoked by each of these files during execution are extracted. This forms the base for our proposed method for malware detection and classification on which we perform feature extraction.

The basic structure of the PE file is as given in Figure 3. The beginning of the DOS header is a pointer to the Portable Executable (PE) File header. DOS will print the error message and terminate, and Windows will then follow this pointer to the next batch of

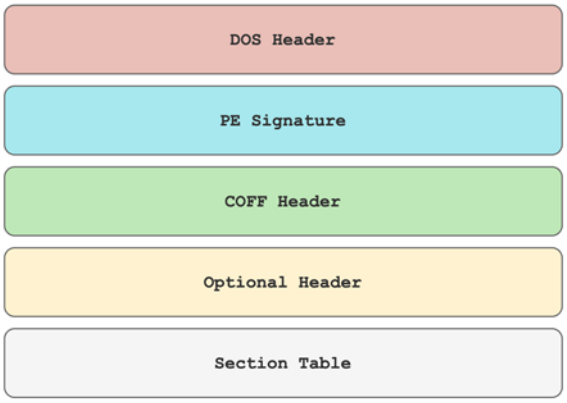

Fig. 3 Structure of the PE-32 executable file

information. The PE header consists of a File ID signature. This signature indicates a) that this file is a legitimate PE file, and b) the byte order of the file. Byte order for all PE files are assumed to be in "little endian" format. The COFF header is present in both COFF object files (before they are linked) and in PE files where it is known as the "File header". The COFF header has some information that is useful to an executable, and some information that is more useful to an object file. The "PE Optional Header" is required in Executable files. The Optional header includes information about the file structure. It contains a signature that identifies the image. The section table consists of an array of IMAGE_SECTION_HEADER structures. The number of structures found in the file are determined by the member NumberOfSections in the COFF Header. Each structure is 40 bytes in length. A PE loader will place the sections of the executable image at the locations specified by these section descriptors.

Common sections are:

1. .text/.code/CODE/TEXT - Contains executable code (machine instructions)
2. .textbss/TEXTBSS - Present if Incremental Linking is enabled
3. .data/.idata/DATA/IDATA - Contains initialised data
4. .bss/BSS - Contains uninitialised data
5. .rsrc - Contains resource data

It is from this code section that the system calls are extracted for further analysis.

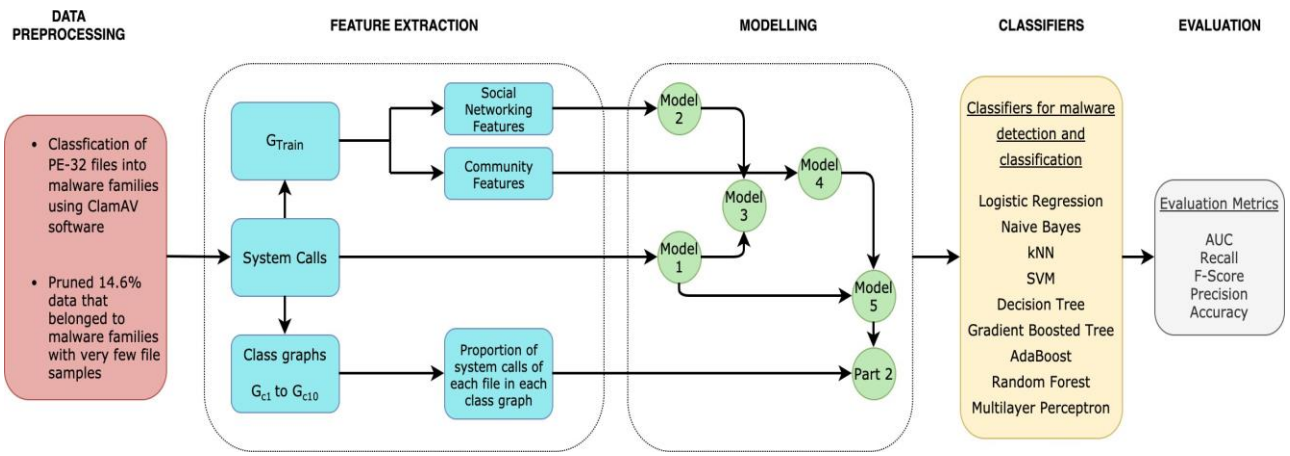

Fig. 4 Block diagram for feature extraction and modeling

3. 4 Feature Extraction and Reduction

The feature extraction process is achieved in 4 major steps starting with the PE-32 files categorized in the previous section. The entire feature extraction process is presented as a block diagram in Figure 4.

3. 4. 1 System call extraction

For this experiment, we consider each of the PE-32 files (both malware and benign) individually. Python code is run on each of the files to return a sequential list of systems calls *S* each, invoked during the program execution. This program is run in a cuckoo sandbox environment to ensure that the execution of malware files does not harm the system. During the system call extraction in the sandbox environment no system call flooding was observed for all malware and benign PE-32 files.

The resultant set of PE-32 files are considered class-wise with each class having 6000 files. Each of these classes is split randomly into training and testing in a 70-30 ratio. This split is done across each class and not across the entire dataset. This is done to reduce any bias that could arise if the dataset is unbalanced.

3. 4. 2 Construction System Call Graphs

From the executable malware files in the training set, a system call graph is constructed using the call sequences such as for example, LoadLibraryA, GetProcAddress, VirtualAlloc, VirtualFree, and VirtualProtect. These system calls either simply pass information or invoke other system calls. The graph is a directed graph G = (V, E), where V represents system calls and E indicates that a particular system call, say S2 was invoked after another system call, say S1. The graph is weighted in each direction. Using the system calls obtained from the training data, a directed graph G_{Train} = (V, E) with |V| = 538 and |E| = 30836 is constructed weighted in each direction.

3. 4. 3 Community Detection in System Call Graphs

Communities are natural and fundamental elements that exist in a wide variety of network systems. Community structure is one of the most important features of real networks. Identifying communities in networks is a crucial step for gaining an in-depth understanding of network structure, dynamics, and interactions. Detecting closely-knit network groups in a large-scale network is a challenge and is crucial in many applications such as social networks, biological interactions such as correlated protein structures, psychology, businesses, e-commerce from recommendations and so on.

Community detection is the process of clustering nodes in a graph into groups such that the nodes composing a group are generally admitted to sharing common properties and/or can be involved in the same role or function. The premise behind the use of community detection is that each file can be represented as a weighted graph of system calls. The newer variants of existing malware families are made with slight code changes to circumvent detection by anti-virus softwares. But, when represented as a graph, these malware graphs belonging to the same family show a high degree of similarity. Thus, analysis of this structure would lead to reduced classification time and would detect minor variations leading to improved accuracy.

There are several community detection algorithms that have been reported in the literature. In this, the Louvain Algorithm has been chosen in view of its successful application in detecting structured P2P bots [11]. Louvain method to be highly scalable and still able to produce consistent communities on very large graphs in various other domains such as Botnets [11] and other social graphs [17] and hence our motivation for the choice of this community detection algorithm. Louvain algorithm [18] is a non-overlapping community detection algorithm based on local optimization of Newman-Girvan modularity in the neighborhood of each node. The quality of the communities referred as partitions is measured by Modularity of the partition. Modularity has a scale value between -1 and 1 that measures the density of edges inside communities to edges outside communities.

Modularity (Q) is defined as [14],

$$Q(C) = \frac{1}{2m}\Sigma\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c(i), c(j)) \quad (1)$$

where c(i) is the community to which node i is assigned and (c(i),c(j)) is the Kronecker delta function, if it is 1 then nodes *i* and *j* belong to the same community and 0 otherwise. k_i and k_j are the sums of the weights of the edges attached to nodes *i* and *j* respectively, *m* is the sum of all the edge weights in the graph and Aij represents the edge weight between nodes *i* and *j*.

The optimization is performed in two steps. First, the method looks for "small" communities by optimizing modularity locally. Second, it aggregates nodes belonging to the same community and builds a new network whose nodes are the communities. To the communities formed using the Louvain algorithm other measures of centrality and similarity are computed to generate a feature vector.

On the graph G_{Train} described above, community detection is performed. Louvain algorithm is run on this graph to obtain a set of *m* communities In our case, it turned out that *m=23*. Each of these communities is considered as an individual directed graph G_{C1}, G_{C2} ... G_{Cm}. Each graph $G_{C}i$ is a graph of interconnections within the nodes in the community i. The set of community features is extracted from G_{Train} after performing community detection. The community features that are extracted from the 23 communities are community in-degree, community out-degree, the density of community, number of communities as neighbors and number of members.

In the next section, various social networking and community level features extracted These graphs are used to compute features as specified in the following sections.

3. 4. 4 Feature Computation

Using the graphs generated in the previous section, we compute the features for the proposed mode. The features computed in our present work can be broadly put into two categories, social networking features and community features.

Social Networking Features [19]*: We consider* G_{Train} to compute the following social networking properties to form our feature vector:*Degree centrality, Closeness centrality, Eigenvector centrality, Between-*

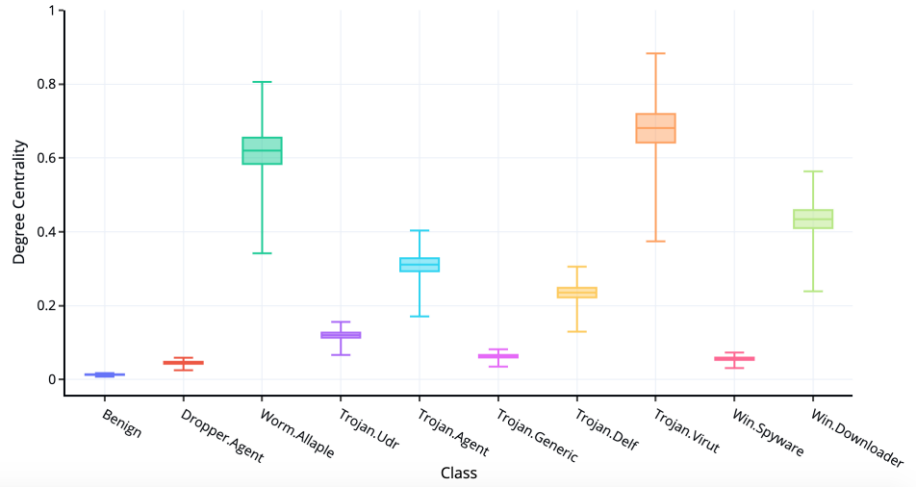

Fig. 5 Boxplot depicting the trend of the degree centrality across the classes

ness centrality, In degree centrality, Out degree centrality, Local node connectivity, Local edge connectivity, Maximum flow, Harmonic centrality, Local reaching centrality

Community Features: Each of the communities obtained from the community detection algorithm used is taken as input to compute the following community features:

Number of members: The number of nodes that constitutes that community. Every node belonging to a community is assigned the same value.

Number of edges: The number of edges that constitutes that community. Every node belonging to this community is assigned the same value.

Density of communities: The density of a community is the ratio of the number of edges and the number of possible edges. In our case of weighted directed graph G = (V, E), the density is computed as:

$$d = \frac{\Sigma_{u \in V, v \in V, u \neq v} weight(u,v)}{|V| \cdot |V-1|} \quad (2)$$

This value remains the same for every node u ∈ V in the graph G. Since we are interested in calculating the density of a community, we calculate the intra-cluster density of each community. It is the ratio of the number of internal edges of the community, divided by the number of possible connections inside the community. The denominator is the number of pairwise combination of nodes within the community or the number of edges if that community were a clique. This calculated intra-cluster density value remains same for every node in that community.

In-degree: For a graph G = (V, E), the in-degree of a node u ∈ V indicates the sum of incoming edges to the node u

$$deg^{-}(u) = \Sigma_{v \in V, u \neq v} weight(v,u) \quad (3)$$

Indegree of a node within the community is the taken as the sum of incoming edges to that node from other nodes within the community.

Out-degree: For a graph G = (V, E), the out-degree of a node u ∈ V indicates the sum of edges outgoing from node u.

$$deg^{+}(u) = \Sigma_{v \in V, u \neq v} weight(u,v) \quad (4)$$

Outdegree of a node within the community is taken as the sum of the outgoing edges from that node to other nodes within the community

3. 4. 5 Feature Mapping

Each of the features computed in the previous section are done so with respect to nodes of the graph, which are system calls in our case. For the proposed method, we need the features to be associated with files which form our dataset. Each file has a set of system calls as discussed previously, the aim is to now map the features associated with system calls to the files. Rather than choosing a random system call value for each file, we address this issue by taking the max value of all the system calls of a file for each feature. Figure 5 shows the boxplot depicting the trend of the degree centrality across the classes. As can be seen in this figure, the trend across other forms of aggregation such as min, mean and mode show a similar trend to that of max and exhibit good classification features. It should be emphasized here that this trend is similar and consistent across all the features considered in this work, justifying the choice of features. In view of this, the max values of the features are sufficient and have been used in this work to capture the overall trend.

For a feature, say X, computed on graph G_{Train} with nodes n_1, n_2 ... n_j, j values are obtained, represented as $x_1, x_2 \ldots x_j$. To map a file, say f_i, which has k system calls, in turn contributing k nodes in the G_{Train}, we choose the maximum and minimum of the subset X belonging to the k nodes.

$$f_i = max(x_1, x_2, x_3 \ldots x_k) \quad (5)$$

For the features extracted using community detection, the method used for the mapping is slightly different as each file has system calls that could be spread across multiple communities. Thus, for file a f_i with system calls/nodes n_1, n_2 ... n_j spread across k communities, we start at a higher level of hierarchy i.e., first map a file to one community. To achieve this, we choose the community wherein the maximum number of nodes belonging to f_i are present. In case of ambiguity, i.e. an equal number of nodes spread across multiple communities, the community graph with fewer members is selected. To map the features, as mentioned above for graphs G_{Train}, we take the maximum of the X values associated with the j nodes belonging to f_i.

3. 4. 6 Feature Reduction

After the feature extraction step as described in detail above, the feature vector generated consists of 554 features. The feature reduction step step consists of identifying the related features from the dataset and removing the irrelevant or less important features which do not contribute much to our target variable in order to achieve better accuracy for our model. Feature reduction is performed for the following reasons:

- *Reduces Overfitting:* Less redundant data means less opportunity to make decisions based on noise.

- *Improves Accuracy:* Less misleading data means modelling accuracy improves.
- *Reduces Training Time:* fewer data points reduce algorithm complexity and algorithms train faster.

We achieve feature reduction by combining correlation matrix with heatmap and univariate selection, which is further bolstered by computing the information gain for each feature. At the end of the feature reduction phase, the size of the feature vector came down to 389 features.

3. 5 Model Training

The feature vector obtained after feature reduction is then fed to the set of machine learning classification algorithms consisting of Logistic Regression, Naive Bayes, kNN, SVM, AdaBoost, Decision Tree, Gradient Boosted Tree, Random Forest and Multilayer Perceptron [20]. All the models are implemented using sklearn and Tensorflow packages of python [21]. The model learns the social networking patterns and community detection features to detect and classify the variants belonging to known malware families. The features extracted from the graphs formed on the training data is used to train the model and the features mapped to the testing data is used to evaluate the effectiveness of the model. Though the same communities are used to assign feature values for both the train and test set there is no inherent data leakage. The reason is that the data split is prior to graph creation and thus, no test data is used to create the graphs. Additionally, only those features that are directly attributed to G_{Train} are chosen. Further, the feature mapping is performed without regard to the class, hence the features assigned are equivalent to those that can be extracted from the graphs formed using the test data. Throughout the feature mapping process, there is no indication of the class or particular community for assigning the values.
This paper puts across the efficacy of the proposed method using 9 machine learning classification algorithms. The following sub-sections discuss in brief the packages used and the hyper parameter tuning performed.

3. 5. 1 Packages used

The entire code for this work is written in Python programming language. The data preprocessing step is performed in a Cuckoo sandbox environment. The features of the malware and benign files are extracted using custom Python codes. Community detection employed in this paper is achieved using the Python Louvain library.
The machine learning algorithms used in this paper are implemented using scikit-learn and tensorflow.keras libraries of Python. Each algorithm has a different set of parameters that have been altered to best suit the task at hand, which is discussed in the next sub-section.

3. 5. 2 Hyperparameter Tuning

1. Logistics Regression - The C value that indicates the inverse of regularization strength - that specifies the extent of regularization - has been tuned to 0.01. It was found that a larger C value, owing to the number of files and the possibility of similar patterns in the malware variants, resulted in lower performance.
2. k-Nearest Neighbours - For the experiments, k, has been varied from 9 to 47. It was found that both *k=13* and *k=29* give comparable performance in terms of accuracy, while *k=13* provides works better in terms of time. Regardless, it was found that the k-NN algorithm was the slowest amongst all algorithms considered.
3. SVM - The regularisation parameter used here is 0.1 with "rbf" kernel as the data is not linearly separable.
4. AdaBoost - Owing to limited computing power, the number of estimators was restricted to 100 with a learning rate of 0.1. While a higher number of estimators can give a better prediction confidence, the combination used was learned to provide the best computation - performance trade-off.
5. Decision Tree - For the features extracted from the call graphs, decision trees with entropy and maximum depth of 50 gave the best performance. A depth higher than this resulted in overfitting.
6. Random Forest - The number of trees in the forest was limited to 100 to avoid overfitting. A maximum depth of 50 gave the best performance.
7. Multi-layer perceptron - Our MLP architecture consists of 5 hidden layers with 4096, 1024, 512, 512 and 256 nodes. There are additionally 2 dense layers with 1024 nodes each. To ensure that there is no overfitting, the last 2 layers have dropout layers with probabilities of 0.3 and 0.2. With Adam optimisation all odd the hidden layers have eLU activation and all even hidden layers have ReLU activation. The output layer uses Softmax activation.

3. 6 Evaluation

Once the model is trained, we perform an evaluation to verify the effectiveness of our proposed method. For a quantitative comparison of different machine learning algorithms in the application of malware detection and classification using community detection and social network analysis, we use the following evaluation criteria:

- AUC - The area under the ROC curve is used in classification analysis in order to determine which of the used models predicts the classes best. The AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.
- Recall - Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (6)$$

- Precision - It is the number of correct positive results divided by the number of positive results predicted by the classifier

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (7)$$

- F-Score - F-Score is the harmonic mean between precision and recall. It is defined as:

$$F - Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \quad (8)$$

- Accuracy - It is the ratio of the number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \quad (9)$$

Two pints worth noting in the classification of malware files. First is the fact that the attackers may use a packer, a tool that compresses, encrypts, and/or modifies a malicious file's format [22]. Traditional methods may fail in that case. There are several methods available in the literature to take care of the packed malware [23.24]. In our case, we had a few examples of packed malware examples. In these cases, we had first detected

the call for unpacking and separated these blocks and executed them separately and added to the call graph.

The second issue with our technique is that the modern-day malware writers create malware that are environment-aware or context-aware. These may also be anti-sandbox malware that may generate flooding while being executed. It is designed to run under specific conditions in the computing environment of the target system through tactics such as fingerprinting followed by reverse Turing test [25]. The malware evades detection in the sandbox environment using different techniques. For example, playing benign for a certain time, following which it is automatically activated. There has been, proportionality, an increase in research in understanding the evading techniques exploited by such malware and in developing methodologies that detect environment-aware malware despite the evading techniques. Authors in [26] present statistical models that capture a system's age and degree of use, which can be used to aid sandbox operators in creating system images that exhibit a realistic wear-and-tear state. In [27], the authors propose a novel technique to detect malware samples that exhibit semantically different behavior across different analysis environments. But, in our case, none of 60,000 files analyzed, none of them were of the anti-sandbox or environment aware files.

4. Results and Discussion

4.1 Part 1

Based on the features extracted in the previous stage, we built various models to evaluate the effectiveness of the use of social network properties and community detection for the application of malware detection and classification.

The following provides a detailed description of the various models (represented in Figure 1) used in the proposed method:

- Model-1: The first model is in line with current literature. It is the set of OS-level actions performed by a malware file. This is extracted using the system calls. For our proposed method, we use this model as the baseline.
- Model-2: This consists of only the social networking features associated with G_{Train}.
- Model-3: This model has the features of Model-1 and Model-2.
- Model-4: For this model, we include all the social networking features computed on the graph G_{Train} i.e. Model-2 and the community features that are directly attributed to G_{Train}.
- Model-5: This model consists of all the features of Model-1 and Model-4.

For each of the following graphs, the x-axis depicts the model number described previously i.e. Model-1, Model-2, Model-3, Model-4 and Model-5 while the y-axis represents Precision (Figure 6), Recall (Figure 7), F-Score (Figure 8), AUC (Figure 9) and Accuracy (Figure 10) respectively for each graph.

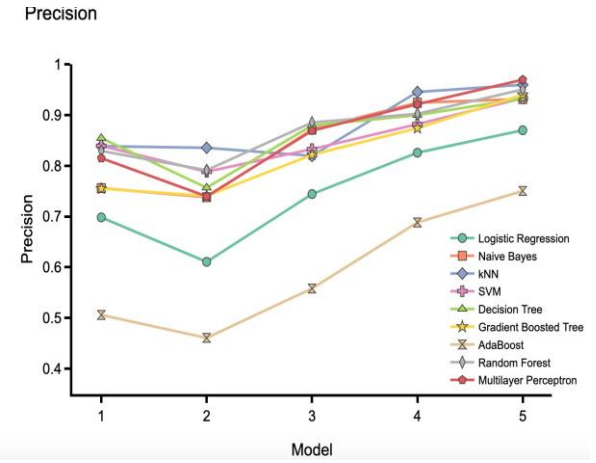

Fig. 6 Precision of the proposed model

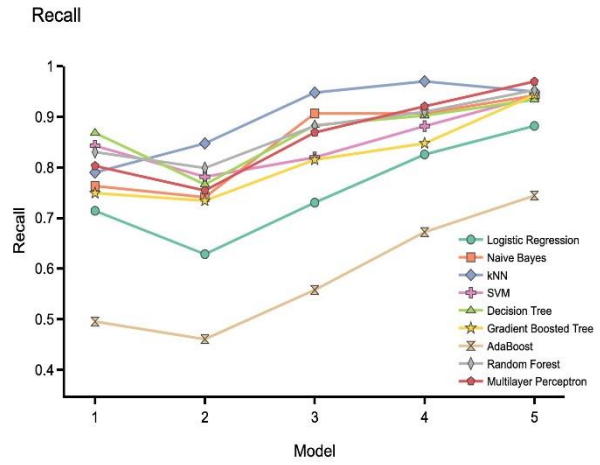

Fig. 7 Recall of the proposed model

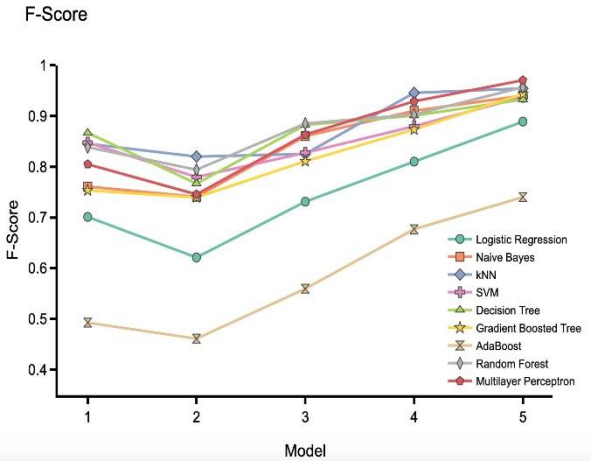

Fig. 8 F-Score of the proposed model

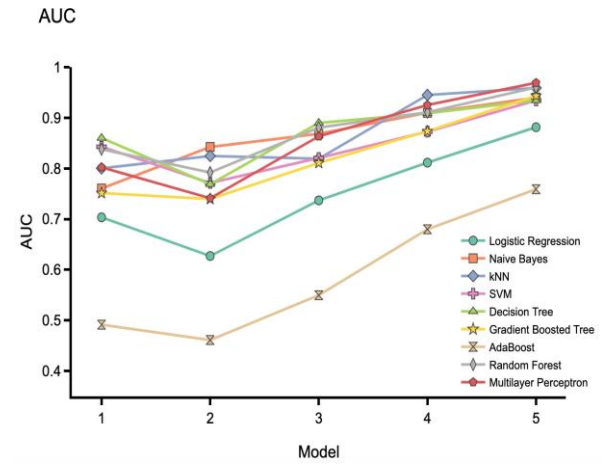

Fig. 9 AUC of the proposed model

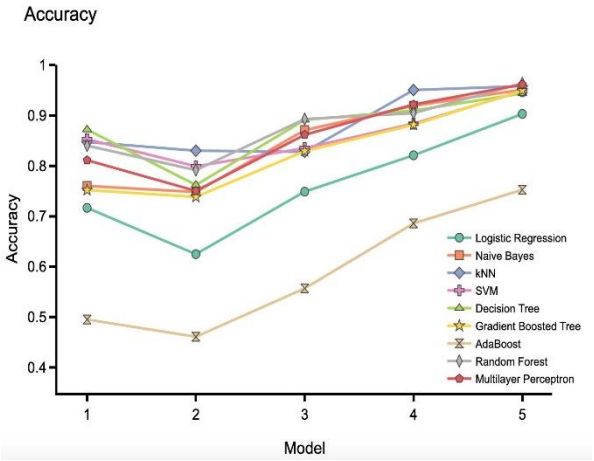

Fig. 10 Accuracy of the proposed model

Model-1 consists of only the actions performed by the files during execution. This provides an average accuracy of 0.77 across all classifiers. Decision Tree classifier provides the highest accuracy of 0.87 for Model-1. Model-2 is based on the social networking features of G_{Train}. It shows a slight decline in performance when compared to Model-1 with an average accuracy of 0.72. For the feature vector of Model-2 AdaBoost falls behind while k-NN gives the best results with an accuracy of 0.83. Model-3 has an average accuracy of 0.81 which is higher than both Model-1 and Model-2. Both Decision Tree and Random Forest exhibit comparable performance with an accuracy slightly above 0.89.

The social networking and community features extracted from G_{Train} included in Model-4 yields high performance in case of k-Nearest Neighbours with an accuracy of 0.95 followed by Multilayer Perceptron with an accuracy of 0.92. Since, kNN algorithm works by identifying the closest neighbours, it seems intuitive that it is able to pick up on the trends shown by the social networking features and community detection.

From the graphs (Figure 6-10) shown it can be seen that the model, Model-5 provides a much better classification when compared to the other 4 models for all classifiers considered thus justifying the stated motivation for this work. Considering all classifiers, an average of 5.12% increase in performance (Precision 4.53% from 0.87 Model-4 to 0.92 Model-5, Recall 5.1% from 0.91 Model-4 to 0.86 Model-5, F-Score 5.27% from 0.86 Model-4 to 0.90 Model-5, AUC 5.3% from 0.87 Model-4 to 0.92 Model-5, Accuracy 5.4% from 0.86 Model-4 to 0.91 Model-5) is seen from Model-4 to Model-5.

From the results discussed above, it can be observed that the use of community detection and social networking properties improve the performance of the classifiers significantly when compared to Model-1.

4.2 Adding Class-level features

To further enhance our proposed method, we add class-level features. To this end, for each of the ten classes, we construct a directed graph G_1 to G_{10}. Each graph belongs to one class of malware or benign family. For each file f_i, in the dataset, we find the proportion of systems calls present in each class.

$$f_{ij} = \frac{Number\ of\ system\ calls\ present\ in G_j and f_i}{Total\ number\ of\ system\ calls\ in f_i} \quad (10)$$

where j = 1, 2 ...10, and G_j indicates the corresponding class graph. These 10 features are added as additional features for each file to Model-5 (described above) to get the following results given by Table 1. On an average Model-5 with class features exhibits a 3% improvement in performance (Precision 4.4%, Recall 3.3%, Accuracy 1.3%, 3.3% AUC, 2.2% F-Score) when compared to that of Model-5. MLP classifier with Model-5 with class-level features shows the highest accuracy of 0.973 across all combinations discussed in this paper. When compared to Model-1, Model-5 with class-level features demonstrates a 22.4% increase in overall performance (Precision 22.1%, Recall 23.7%, Accuracy 20.1%, AUC 23.7%, F-Score 22.6%) across all classifiers.

The confusion matrix for the MLP Classifier for Model-5 with class-level features is presented in Table 2. It can be observed from Table 2. Majority of the families of malware are classified correctly. However, as seen from the Table 2. the Dropper.Agent family is classified correctly 0.86 times followed by Trojan.Agent (0.92 times) and Trojan.Virut (0.96 times). The results for other classifiers have been found to be similar.

5. Conclusion and Future work

In this paper, we proposed a method that exploits the use of community detection and social network analytics for the application of malware detection and classification. To further bolster the effectiveness of our method, we tested for multiple models over a set of different machine learning algorithms. Based on the work carried out, we can conclude that the use of community detection leads to a high degree of accuracy across all algorithms chosen. Our work outlined above has revealed that the proposed method that exploits social networking and community features has significantly improved the performance of our malware classification algorithm. An accuracy, precision, recall and F-Score of 0.96 and above is obtained by classifiers such as Multilayer Perceptron, Random Forest and k Nearest Neighbours. It is safe to conclude that irrespective of machine learning technique our approach achieves high accuracy with reduced classification time.

Most feature based ML algorithms are susceptible to adversarial attacks. It needs to be explored in the future to develop models and features that are resilient to adversarial attacks. However, our approach includes many combinations of attributes besides the system level features such as social network features,

community level and class-level features.

Table 1 **Results of classifiers for Model-5 with class-level features**

Algorithm	Precision	Recall	Accuracy	AUC	F-Score
Logistic Regression	0.909	0.910	0.909	0.907	0.912
Naive Bayes	0.956	0.959	0.961	0.960	0.958
k Nearest Neighbours	0.967	0.962	0.948	0.969	0.967
Support Vector Machine	0.957	0.948	0.946	0.952	0.954
Decision Tree	0.942	0.949	0.939	0.940	0.941
Gradient Boosted Tree	0.958	0.950	0.955	0.957	0.950
AdaBoost	0.793	0.782	0.790	0.789	0.790
Random Forest	0.963	0.969	0.958	0.960	0.965
Multilayer Perceptron	0.980	0.980	0.973	0.978	0.977

Table 2 Confusion matrix of MLP classifier for Model-5 with class-level features

	Benign	Dropper. Agent	Worm. Allaple	Trojan. Udr	Trojan. Agent	Trojan. Generic	Trojan. Delf	Trojan. Virut	Win. Spyware	Win.Downloader
Benign	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Dropper. Agent	0.0	0.86	0.0	0.0	0.12	0.0	0.0	0.02	0.0	0.0
Worm. Allaple	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Trojan. Udr	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
Trojan. Agent	0.0	0.06	0.0	0.0	0.92	0.0	0.0	0.02	0.0	0.0
Trojan. Generic	0.02	0.0	0.0	0.0	0.0	0.98	0.0	0.0	0.0	0.0
Trojan. Delf	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Trojan. Virut	0.0	0.04	0.0	0.0	0.0	0.0	0.0	0.96	0.0	0.0
Win. Spyware	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Win. Down loader	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

References

1. Infographic - Internet Security Insights Q1 2019. https://www.watchguard.com/wgrd-resource-center/infographic/internet-security-insights-q1-2019 .
2. Ucci, D., Aniello, L., Baldoni, R.: Survey of machine learning techniques for malware analysis. Computers & Security **81**, 123-147 (2019).
3. Souri, A., Hosseini, R.: A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Computing and Information Sciences **8**(1), 3 (2018).
4. Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. Journal of Network and Computer Applications **153**, 102526 (2020).
5. Harsha Latha, P.a.R.M.: Classification of Malware detection using Machine Learning Algorithms- A Survey. International Journal of Scientific Research and Technology. **9**(2), 1796-1802 (2020).
6. Kim, H.M., Song, H.M., Seo, J.W., Kim, H.K.: Andro-simnet: Android malware family classification using social network analysis. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST) 2018, pp. 1-8. IEEE
7. Jang, J.-w., Woo, J., Mohaisen, A., Yun, J., Kim, H.K.: Malnetminer: Malware classification approach based on social network analysis of system call graph. Mathematical Problems in Engineering **2015** (2015).
8. Jang, J., Brumley, D., Venkataraman, S.: Bitshred: feature hashing malware for scalable triage and semantic analysis. In: Proceedings of the 18th ACM conference on Computer and communications security 2011, pp. 309-320
9. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR) **50**(3), 1-40 (2017).
10. Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep learning for classification of malware system call sequences. In: Australasian Joint Conference on Artificial Intelligence 2016, pp. 137-149. Springer
11. Venkatesh, B., Choudhury, S.H., Nagaraja, S., Balakrishnan, N.: BotSpot: fast graph based identification of structured P2P bots. Journal of Computer Virology and Hacking Techniques **11**(4), 247-261 (2015).
12. Bhattacharya, A., Goswami, R.T.: Community Based Feature Selection Method for Detection of Android Malware. Journal of Global Information Management (JGIM) **26**(3), 54-77 (2018).
13. Kim, C.W.: Ntmaldetect: A machine learning approach to malware detection using native api system calls. arXiv preprint arXiv:1802.05412 (2018).
14. Du, Y., Wang, J., Li, Q.: An android malware detection approach using community structures of weighted function call graphs. IEEE Access **5**, 17478-17486 (2017).
15. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the national academy of sciences **99**(12), 7821-7826 (2002).
16. Fan, M., Liu, J., Luo, X., Chen, K., Chen, T., Tian, Z., Zhang, X., Zheng, Q., Liu, T.: Frequent subgraph based familial classification of android malware. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE) 2016, pp. 24-35. IEEE
17. Kolli, N., Balakrishnan, N.: Hybrid Features for Churn Prediction in Mobile Telecom Networks with Data Constraints.
18. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), P10008 (2008).
19. Van Steen, M.: An introduction to graph theory and complex networks. Copyrighted material (2010).
20. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning (Adaptive Computation and Machine Learning series). In. e MIT Press, Cambridge, England, (2016)
21. Géron, A.: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, (2019)
22. Roccia Thomas, https://www.mcafee.com/blogs/enterprise/malware-packers-use-tricks-avoid-analysis-detection/
23. Devi, D., Nandi, S. , Detection of Packed Malware, SecurIT '12: Proceedings of the First International Conference on Security of Internet of ThingsAugust 2012 Pages 22-26 https://doi.org/10.1145/2490428.2490431
24. Yan, W., Zhang, Z., Ansari, N. Revealing Packed Malware, in IEEE Security & Privacy, vol. 6, no. 5, pp. 65-69, Sept.-Oct. 2008, doi: 10.1109/MSP.2008.126.
25. Afianian, A., Niksefat, S., Sadeghiyan, B., Baptiste. D., Malware Dynamic Analysis Evasion Techniques: A Survey. ACM Computing. Survey. 52, 6, Article 126 (January 2020), 28 pages. DOI:https://doi.org/10.1145/336500
26. N. Miramirkhani, M. P. Appini, N. Nikiforakis and M. Polychronakis. Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts. 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, 2017, pp. 1009-1024, DOI: 10.1109/SP.2017.42.
27. Lindorfer, M., Kolbitsch, C., Comparetti. P.M., Detecting Environment-Sensitive Malware. In: Sommer R., Balzarotti D., Maier G. (eds) Recent Advances in Intrusion Detection. RAID 2011. Lecture Notes in Computer Science, vol 6961. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-23644-0_18

List of Figures:

List of Tables: