Journal of Computer Security 29 (2021) 651–688
DOI 10.3233/JCS-210057
IOS Press

A graph-based framework for malicious software detection and classification utilizing temporal-graphs

Helen-Maria Dounavi, Anna Mpanti, Stavros D. Nikolopoulos * and Iosif Polenakis
Department of Computer Science & Engineering, University of Ioannina, Ioannina, Greece
E-mails: edounavi@cse.uoi.gr, ampanti@cs.uoi.gr, stavros@cs.uoi.gr, ipolenak@cs.uoi.gr

Abstract. In this paper we present a graph-based framework that, utilizing relations between groups of System-calls, detects whether an unknown software sample is malicious or benign, and classifies a malicious software to one of a set of known malware families. In our approach we propose a novel graph representation of dependency graphs by capturing their structural evolution over time constructing sequential graph instances, the so-called Temporal Graphs. The partitions of the temporal evolution of a graph defined by specific time-slots, results to different types of graphs representations based upon the information we capture across the capturing of its evolution. The proposed graph-based framework utilizes the proposed types of temporal graphs computing similarity metrics over various graph characteristics in order to conduct the malware detection and classification procedures. Finally, we evaluate the detection rates and the classification ability of our proposed graph-based framework conducting a series of experiments over a set of known malware samples pre-classified into malware families.

Keywords: Malicious software, malware classification, security, temporal graphs, graph similarity

1. Introduction

Malware or malicious software is a software type intended to cause harm to end point computers, systems or networks [54]. In this work we design and propose a graph based model that develops an algorithmic technique for malware detection and classification. Our method is applied on unknown software samples in order to detect whether they are malicious or not, and further classify them to one of a set of known malware families (i.e., set of malicious malware samples with similar functionality), as they have been developed by various antivirus software vendors.

1.1. Protection against mutated malware

On the contrary part of our scientific field, malware authors, have developed and deployed various techniques in order to avoid the traditional byte-level signature based detection methods. Since such detection methods appear to be significantly fragile against even the least mutation of the initial subject (i.e., ancestor malware sample), they mutate their software products (malware) creating structurally different but functionally similar copies of them. Except from the mutation methods that leverage one, or more, levels of encryption, there also exist more advanced mutation methods. Some of the most applicable malware mutations are the oligomorphism which is achieved through obfuscation techniques, the

* Corresponding author. E-mail: edounavi@cse.uoi.gr.

polymorphism [48] where the code is modified through encryption techniques and the metamorphism, in which multiple structurally different copies of a malware sample are generated.

Hence, while the main functionality of a malware sample remains immutable during its mutations, malware samples can be merged into groups of malware samples with common functionality, the so called malware families. So, in this work we developed an algorithmic technique that not only detects if a program is malicious or not, but additionally, given a malicious software it can decide the malware family that it belongs to.

Since malicious software poses a major threat, several protection approaches have been proposed and implemented in order to eliminate such threats. The main corpus of the defence line is mainly developed over three axes, namely malware analysis, malware detection and malware classification. Malware analysis [54] is the process of determining the purpose and the functionality or, abstractly, the behaviour of a given malicious code. The term *malware detection* referrers to the process of determining whether a given program P is malicious or benign according to an *a priori knowledge* [35]. Finally, the term *malware classification* refers to the process of determining the malware family to which a particular malware sample, let M belongs to, in order to provide the ability to generalize detection signatures from sample level to family level.

1.2. Contribution

In our approach, we leverage the use of System-call Dependency Graphs (ScDG) to represent the software samples in order to distinguish if they are malicious or not and further classify them to a malware family. More precisely, in order to make the detection and classification procedures more resilient to known malware mutation techniques, we construct a directed edge-weighted graph, which we call Group Relation Graph – GrG [45], that is a generalized graph structure resulting from ScDG after grouping disjoint subsets of its vertices. Throughout these processes, over specific time intervals, we preserve instances of GrGs creating hence temporal graphs that depict their structural evolution over time. Given a ScDG graph that represents a known malware sample and a ScDG graph that represent an test sample, we utilize these instances (i.e., temporal graphs) representing the evolution of their corresponding GrG, produced both on each ScDG, in order to perform graph similarity towards the processes of malware detection and classification.

In this work, we propose, develop and present an integrated graph-based framework for distinguishing graph representations referencing malicious software samples and further classifying them in sets of known malware families. Firstly, we discuss our proposed graph abstractions that are based on the ScDG graphs representing the relations between system-call groups (i.e., GrG), and present another graph representation that describes the temporal evolution on the structure of the GrG graphs. In our approach we distinguish two types of temporal graphs according to the registration of the structural modifications and compute the graph similarity between such temporal graphs utilizing specific characteristics of graphs Finally, we present the potentials of our approach, by evaluating the detection and classification rates exhibited by our framework against malicious samples.

1.3. Related work

Detection: Ding *et al.* [10] propose an algorithm to extract the common behaviour graph, which is used to represent the behavioral features of a malware family, and they suggest a graph matching algorithm which is based on the maximum weight sub-graph in order to detect malicious code. Based on

executable files static analysis, A.V. Kozachok and V.I. Kozachok [28], experimentally investigate the evaluation of the developed detection mechanism efficiency on the basis of neural networks and decision tree composition. In addition, Hashemi *et al.* [19] present a malware detection method based on the OpCodes within an executable file, which generates a graph of operational codes (OpCode) within an executable file and embeds this graph to eigenspace using "Power Iteration" method. Hashemi and Hamzeh [20], recently presented a method which convert executable files into digital images and extract visual features of the executable files and they use machine learning methods to detect malware. Wüchner *et al.* [60] present a novel malware detection approach based on metrics over quantitative data flow graphs and their approach is able to detect new malware. The same authors [59], focused on the incremental construction of aggregated quantitative data-flow graphs, suggest another novel behavioural malware detection approach based on a generic system-wide quantitative data-flow model. Recently, John *et al.* [25] propose a novel Android malware detection mechanism using Graph Convolutional Nets which uses centrality measures of the graph as input features, modelling the system calls as graphs.

Classification: In malware classification, there have been proposed other non graph-based malware classification models. A scalable automated approach for malware classification using pattern recognition algorithms and statistical methods, is presented by Islam *et al.* in [23], utilizing the combination of static features extracted by function length and printable strings. Nataraj *et al.* [42] classify malware samples using image processing techniques. Visualizing as gray-scale images the malware binaries, they utilize the fact that,for many malware families, the images belonging to the same family appear very similar in layout and texture. In [43] Nataraj *et al.* utilize a static analysis technique called binary texture analysis in order to classify malicious binary samples into malware families. Makandar and Patrot [34] focus on detection and classification of the Trojan viruses using image processing techniques. In their proposed algorithm Gabor wavelet is used for key of feature extraction method and their experimental results are analysed compared with two classifications such as KNN and SVM. In [21], Hassen and Chan investigate a linear time function call graph (FCG) vector representation based on function clustering that has significant performance gains in addition to improved classification accuracy. They also show how this representation can enable using graph features together with other non-graph features. In addition, a graph matching algorithm that is based on the maximum weight subgraph is used to detect malicious code. In [40], Mukesh et al. propose a machine learning based architecture to distinguish existing and recently developing malware by utilizing network and transport layer traffic features. In [41], Narra *et al.* apply cluster analysis to the problem of classifying unknown malware. They also score samples from malware families that were not used for training or generating the clusters, and classify these new samples based on the existing clusters in order to determine how well new malware families can be classified using cluster results based on a set of previously-known malware families. Following up on the same approach, Xiao *et al.* [62] propose a graph repartition algorithm by transforming API call graphs into fragment behaviours based on the dynamic execution traces of program. Their proposed algorithm relies on the N-order sub-graph (NSG) for constructing the appropriate fragment behaviour. Furthermore, Eskandari *et al.* [13] introduced a new signature type, called ERES (Extended Regular Expression Signature), which generates a more specific signature leading to a more accurate detection and combines token extraction with sequence alignment, accelerated the signature extraction process. In [36], Ming *et al.* propose normalized basic block memorization to speed up semantics-based binary difference and introduce an union-find set structure that records semantically equivalent basic blocks. Recently, in the context of the malware detection problem, Basole *et al.* [4] conduct experiments based on byte n-gram features to quantify the relationship between the generality of the training dataset and the accuracy of the corresponding machine learning models. In [16], Ghanaei *et al.* introduced a supervised

learning method, which classifies new malware variants into their related malware families based on an effective statistical approach to identify, and render critical malicious patterns into malware families.

1.4. Road map

In Section 2 we present the prerequisite theoretical background in order for the graph-based techniques for detection and classification to be developed. Next, in Section 3 we demonstrate the key components of our model, in Section 4 we discuss extensively the main principles and design aspects over the development of our detection and classification scheme concerning the two processes, where in Section 5 we measure the detection and classification potentials of our integrated framework against malicious samples. Finally, in Section 6, we set our further research landmarks concerning the processes of malware detection and classification.

2. Conceptual framework

In this section we discuss the semantics behind the processes of malware analysis, detection and classification. We discuss the principles of the utilization of behaviour-based approaches towards the deployment of resilient detection and classification techniques. Firstly we present the major process preceding the development of detection and classification methods, that is malware analysis, and next we depict the state-of-the art behavioural approaches applied on malware detection and classification.

2.1. Analysing susceptible samples

The traditional signature-based malware detection, despite its fast real-time protection, is still not resilient against malware mutations. Robust detection techniques prerequisite the procedure of *malware analysis*, during which, the analyst collects all the required information, in order to be effective and efficient. The effectiveness of signature scanning, relying on pattern matching fails to detect new malware strains or mutated variants of existing ones [8]. The procedure of malware analysis is consisted by the collection of valuable information concerning either static artifacts or generally behavioural patterns, that could characterize the maliciousness, or not, of a sample, being categorized to two main categories namely static analysis and dynamic analysis, respectively [6,53,57].

Static malware analysis of software is performed over the programming artifacts and structural characteristics of a software sample [9], without the need of its execution. The information obtained during static malware analysis may refer to opcode sequences, control flow graphs, etc. and can be used at will for malware detection [8]. However, since the sample does not need to be executed can be surpasses by easily foiled by obfuscation and packing techniques hence its scalability consists one of its assets [31,53].

On the other hand, Dynamic Malware Analysis refers mostly to the extraction of behavioural features exhibited during the execution of a malicious software sample, mainly captured and depicted through API-calls sequences and the system-calls dependencies [8]. The scalability of dynamic malware analysis may be reduced due to the demand of real time execution [53]. Moreover, despite that obfuscation techniques can easily be defeated through dynamic analysis the time needed for analysis is disproportionate to the rate of birth of mutated malware samples [31].

A specific type of dynamic analysis, called *dynamic taint analysis* or DTA (stands for dynamic taint analysis) traces data flows in programs or systems during execution time. Briefly speaking, taint analysis

distinguishes three elements namely *taint sources*, *taint sinks*, and *propagation rules*. Data flows are taint variables introduced by taint sources (i.e., the output parameters of system calls) and propagated according to the propagation rules to taint sinks (i.e., the input parameters of system calls) [3].

2.2. Detecting malicious behaviors

As we mentioned previously, malicious software samples are intended to compromise the privacy, the confidentiality or the integrity of a system, of data or any other cyber-source constituting hence an intrusion. To this end, Intrusion Detection Systems, or, for short IDS, are deployed in order to monitor the execution of applications, the traffic of networks or whole systems, aiming on spotting malicious activity patterns [2]. The system supervision through an IDS can be performed through the application of *malware detection* techniques, that reference file comparisons against signatures of malicious software [12], behaviour monitoring of malicious patterns and system supervision [2].

However, the increasing birth-rate of new or mutated malware samples has raised the need for efficient and elaborated malware detection techniques that can effectively detect new malware strains in reasonable amounts of time. The detection approaches are strongly connected to the features set provided through the previous stage of malware analysis, and are distinguished to static and dynamic features, respectively. Static features may include, statistical analysis on n-grams or opcodes, properties of control flow graphs, while dynamic features are obtained the execution time of a program and concern its general behaviour (i.e., interaction with the host-environment – O.S.), access events or any other interconnection patterns [18]. Malware detection approaches are divided into two main categories, namely signature-based malware detection and behaviour-based malware detection [1,5,8,22,55]. Next we briefly discuss these two methods and present some of the approaches deployed in each one.

Signature-based malware detection is the dominant technique deployed by antivirus software products due to its time efficacy that provides real-time protection against malicious threats [9]. A byte-level signature is a sequence (i.e., pattern) of bytes used to identify each newly discovered malware, using a scanning scheme of exact correlation and a repository of signatures in order to detect malicious software samples [8]. A signature may represent a byte-code sequence, a binary assembly instruction, an imported Dynamic Link Library (DLL), or function and system calls. [1,5]. However, signature-based detection techniques can easily be evaded through code obfuscation techniques (e.g., polymorphic malware samples)that even the least modification on the code sequence would lead to a completely different byte-sequence [8].

On the other hand, behaviour-based malware detection [24] mainly focuses on capturing the interaction (in terms of interconnection, relations or dependencies between system-elements i.e., system-calls or API calls) between the executed software and the system (i.e., Operating System) [3,5,50,52]. Behaviour-based detection systems as expected require the execution of the software sample in order to extract dynamically (i.e., through dynamic taint analysis) the exhibited behaviours. In order for these dynamic systems to perform the mining of the specified behaviours they utilize software and hardware virtualization technologies, alongside with imitation conditions [55], providing the test sample with an environment as close to reality in order to evade the sandbox-detection mechanisms deployed occasionally by malicious software samples, and letting them exhibit their intentions. Despite the fact that such techniques deploy quite elaborate algorithms on their implementations, the incident that malware families tend to evolve in order to avoid detection [1], results to the need of the development of more elastic and mutation resilient techniques like the one we propose in this work.

2.3. Classifying malware samples

Malware authors, in order to avoid traditional detection methods, produce new (mostly mutated) malware samples rapidly, utilizing existing ones in order for the new strains to preserve the functionality inherited from their ancestors. As referred in [5] mutated malware samples are generated from existing ones utilizing automated techniques [63] or integrated tools, generating new samples from libraries and code parts from code exchange networks. The term *malware classification* has been confused several times with *malware detection*. Distinguishing these procedures, malware detection is a *binary classification*, where a set of unknown samples is classified against a collection of malicious and benign samples, while malware classification refers to the indexing of an already detected malware sample belongs to a particular family or type [17] of similar samples. As described in [27], malicious software samples that belong to the same malware family tend to exhibit similar behavioural and structural profiles, and hence malware classification augments the analysis of new, or mutated, malicious samples where their signatures have not been constructed yet [46].

Another field of malware analysis applied in malware classification is *malware phylogeny* [38], which aims on inferring evolutionary relationships between instances of families. The major profit from creating a phylogeny model is the fact that newly developed elaborated detection systems that deploying such techniques can detect that a sample that has not been previously seen can be related to a malware family, when analysed along an evolution path [32]. Throughout this process the main target is to reveal similarities and relations among a set of specific malware samples coexist and are exhibited by all the members of the set (i.e., malware family) [58], distinguishing its type or family. Such approaches can be utilized to identify evolution trends in over a set of malware samples [5], constituting hence valuable tool for more generalized signatures or, in general, more elaborated detection-techniques. The models applied on phylogeny, using mostly phylogenetic networks, model evolutionary relations among malware families, describing temporal ordering among samples, defining ancestor-descendent relations, as also relationships between families, augmenting hence malware classification and unveiling evolutionary trends [33].

3. Analysing the evolution of temporal graphs

In this section, we discuss the properties of our initial graph representation i.e., the System-call Dependency Graph (i.e., ScDG) and the proposed structural components of our model, namely, the Group Relation Graph (i.e., GrG) and its corresponding graphs that depict its structural evolution through time (i.e., the temporal graphs). Given such graph representations, we present the construction of the key components of our proposed detection and classification model, i.e., their corresponding derivative graphs that depict the temporal evolution of a GrG graph.

3.1. Graph representation

The system-calls invoked during the execution of a program can be traced through taint analysis, and hereafter the behaviour of a program can be represented with a directed acyclic graph (dag), the so called System-call Dependency Graph see, Fig. 1(a). The vertex set of a ScDG is consisted by all the system-calls invoked during the execution of a program and its edge set represents the communication between system-calls as described in [3,39,44]. Recalling that the suspicious sample needs to be executed in a contained environment (i.e., a virtual machine), where during its execution time, dynamic taint analysis

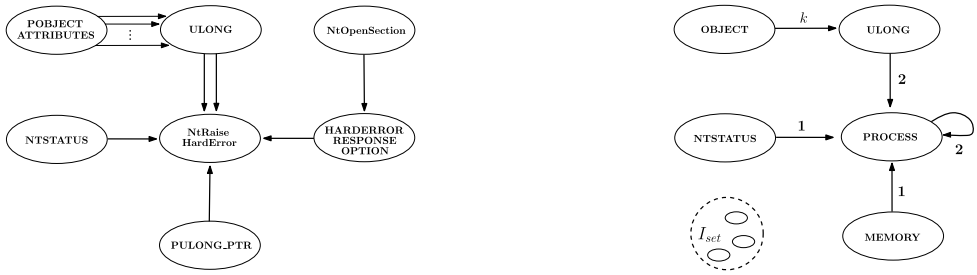


Fig. 1. (a) the system-call dependency graph of a program; (b) the corresponding group relation graph of a program.

is performed in order to capture system-call traces, next we illustrate a simple example that includes the system-call traces obtained, constructing the ScDG of a program. In Fig. 1(a), it is easy to see that the vertex set of this graph is consisted from the system-calls invoked during the execution of the sample and its edge set is consisted by their in between data-flow dependencies, constructing a directed acyclic graph (DAG).

Moreover, as described in [39,44] given a graph representation of malware-behaviour such a ScDG, let G, a more abstract graph representation of a program's behaviour can be constructed based on the fact that system-calls of similar functionality can be classified into the same group [45]. The produced graph representation is a directed weighted graph called Group Relation Graph, that we denote as G^*; see, Fig. 1(b). The whole procedure for constructing the GrG graph from a given ScDG for a program is described in details in [44,45]. Hence, having the grouping of system-calls and a system-call dependency graph ScDG G, the corresponding GrG graph G^* is a directed edge-weighted graph on n vertices $v_1, v_2, \ldots, v_n$ constructed as follows:

(i) for every pair $\{v_i, v_j\} \in V(G^*)$, a directed edge (v_i, v_j) is added in $E(\widehat{G})$ if the two system-calls communicating with each other, let (S_p, S_q), is an edge in $E(G)$ and, S_p belongs to the ith system-call group and S_q belongs to the jth system-call group;
(ii) for each directed edge $(v_i, v_j) \in E(G^*)$, a weight $w(v_i, v_j) \in \Re$ is assigned on it if there are $w(v_i, v_j)$ invocations from a system-call in the ith group to a system-call in the jth group.

which consist the basic component upon which the core graph representation of our model is based.

3.2. Temporal graphs

Throughout the development of our research, we have noticed that, to the best of our knowledge, there does not exist any approach on the literature that utilizes the temporal evolution of a graph in malware detection and classification. Similarly to *phylogeny* that examines the temporal evolution of malware families, the key component of our proposed detection and classification model, leverage the temporal evolution of graphs (in our case GrG graphs) in order to depict the structural modifications performed on the graph and that could distinguish either a malware sample or to a further extent a malware family.

In our model, we define a type of graphs that depict the temporal evolution of our initial graph structures, namely the Group Relation Temporal Graphs or, for short, GrTG. In order to implement such graph structures we approach this modeling by creating instances of the initial GrG graphs during their construction. As we mentioned above, GrG graphs are constructed by the sum of the system-calls invoked interconnecting pairs of system call groups. Hence, since we are given the system-call dependencies in a

series that depicts the time correlation among (i.e., an edge sequence of the ScDG that shows the system-call invocations during execution time), such constructions can be obtained by creating an instance of the produced GrG graphs at specific steps.

Formalizing our previous claim, we can define that for a set of time-partition, let $t_1, t_2, \ldots, t_n$ we can construct n instances of a given GrG graph denoting them by $T(G^*)$ as: $T_1(G^*), T_2(G^*), \ldots, T_n(G^*)$, that depict the structural evolution of a graph in terms of edges, vertex-degrees and vertex-weights of the corresponding graphs at specific time slots. Through this approach we can maintain information about the temporal evolution of the graph thorough its construction procedure,and further leverage such information in order to perform more elaborated graph similarity techniques.

In our proposed model for the development of temporal graphs we define as quantum the elementary interaction between two system-calls in the initial ScDG graph, i.e. an edge, while the `region` represents a range of time-quantums, i.e., a sequence of system-call invocations, i.e., set of edges, in the initial ScDG graph, or equivalently a set of group relations in the GrG graph. Hence a `region` contains a sequence of System-calls, or reversely, the overall sequence of System-calls in a graph are partitioned into equal-sized ranges that contain parts of the overall sequence of system calls. The factor of time actually does not represent the actual quantum of run-time, but each time-quantum corresponds to the invocation of one system-call (i.e., a dependency) or, equivalently, a relation between two System-call Groups (i.e., an edge of the Group Relation Graph). Hence, the total time-line depicts the slots or time-partitions from the appearance of the first to the last group relation. Additionally, in our model, we define as `regions` the set of time-partition, i.e., $t_1, t_2, \ldots, t_n$, and a `region`, let t_i, contains the structural modifications performed (i.e., edges added on the corresponding GrG) from the begin to the end of the ith `region`, where $1 < i \leqslant n, \forall n \in D$, and $D = \{\forall n \in \mathbb{N} : |E(G)| \mod n = 0\}$.

3.3. Discrete and cumulative modification temporal graphs

The factor of time actually does not represent the actual quantum of run-time, but each time-quantum corresponds to the invocation of one system-call (i.e., a dependency) or, equivalently, a relation between two System-call Groups (i.e., an edge of the Group Relation Graph). Hence, the total time-line depicts the slots or time-partitions from the appearance of the first to the last group relation. To this point we should notice that the conceptual substance of Temporal Graphs is to depict the structural evolution of the GrG graphs through the time. However, the structural modification on the instances of the graph over the time can be described either as addition of edges over the exact previous graph instance, or as successive additions of edges performed on all the previous graph instances.

Through this aspect, we can transform a given ScDG into a temporal graph, by registering the modifications performed on the corresponding GrG over is evolution during time, taking into account the sequence (i.e., chronological order) the system-calls were invoked during the execution of the program. Next, we discuss the construction of the corresponding Temporal Graphs according to the two approaches.

In the first approach of our proposed scheme, the construction of the Temporal Graph, that represents the evolution of GrG graphs during time, constructs the induced sub-graph of GrG, respectively, including only the edges that where added on a specific `region`, constructing the Temporal Graph $GrTG$, of the graph GrG, denoting it with $\widehat{T}(G^*)$. So, let the `region` t_i we construct the ith instance of the graph GrG, denoting it as $\widehat{T}_i(G^*)$, containing only the edges added during this particular `region`. In Fig. 2, we depict the discrete structural modification (i.e., temporal evolution) a GrG graphs over the construction of their corresponding Temporal Graph $\widehat{T}(G^*)$ during n `regions`.

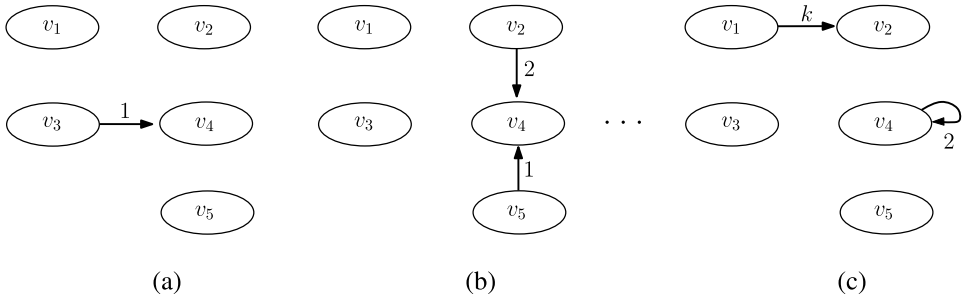


Fig. 2. The temporal evolution of a GrG graph G^* represented by its discrete modification temporal graph $\widehat{T}(G^*)$ over n `regions`: (a) $\widehat{T}_1(G^*)$, (b) $\widehat{T}_2(G^*)$ and (c) $\widehat{T}_n(G^*)$.

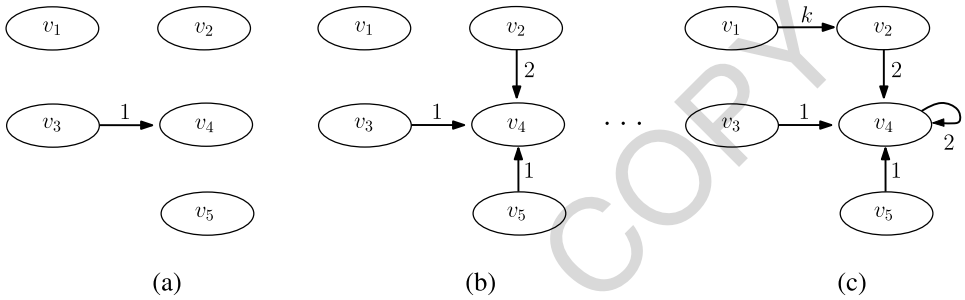


Fig. 3. The temporal evolution of a GrG graph G^* represented by its cumulative modification temporal graph $\widetilde{T}(G^*)$ over n `regions`: (a) $\widetilde{T}_1(G^*)$, (b) $\widetilde{T}_2(G^*)$ and (c) $\widetilde{T}_n(G^*)$.

In this type of Temporal Graphs, the evolution of the graph is represented as an additive procedure, since once an edge has been created at a given time, let i, between two system-call groups on the GrG graph, it will remain permanent on the ancestor Temporal Graphs (i.e., if $\{u, v\} \in E(\widehat{T}_i(G^*)) \rightarrow \{u, v\} \in E(\widehat{T}_1(G^*)), \forall i < j < n$), since it consists a predecessor of the following temporal graphs. In the second approach of our proposed scheme, the construction of the Temporal Graph, that represents the evolution of GrG graphs during time, actually extends the graphs GrG during time, by accumulating the edges that where added on a specific `region`. So, let the `region` t_i we construct the Cumulative Modification Temporal Graphs of the graphs GrG, denoting them with $\widetilde{T}(G^*)$, containing the number of edges added from `region` t_1 until `region` t_i. In Fig. 3, we depict the cumulative structural modification (i.e., temporal evolution) of a GrG graphs over the construction of its corresponding Temporal Graph $\widetilde{T}(G^*)$ during n `regions`.

4. System architecture

In this section we present the key components of our detection and classification model, and describe the key insights that constitute the basis of the corresponding procedures. Discussing the design principles that rule the deployment of our model's components, we present an overview of our detection and classification techniques.

4.1. Design principles

Malware detection and classification are two interconnected procedures. In malware detection the main target is to determine whether a given program is malicious or benign according to an a priori knowledge over what known to be malicious, where malware classification is the following procedure and its intent is to determine the malware family to which the sample, that has been detected as malicious, belongs to. It easily follows that, an *a priori* knowledge of characteristics of known maliciousness has to be stored in a knowledge database, as also that in order to compare two subjects, a similarity measure among them is required. Moreover, the proposed theoretical approach specifies the form of the subjects, where graph-based models interact with similarity metrics that measure specific types of characteristics that represent evolutionary commonalities between temporal graphs. Next, we present the architectural considerations, the design principles, the functionality, and the corresponding deployment of the key components of our proposed detection and classification model.

4.1.1. Knowledge database

The knowledge database is consisted by a set of known malicious samples that have been classified to malware families according to their functional, structural and mostly behavioural commonalities. More precisely, various anti-virus vendors have classified these samples to families based on their own heuristic rules concerning shared behavioural patterns and functionally similar execution profiles. Regarding the detection process, the knowledge database, except the known malicious samples, also includes benign samples in order to measure the false positive rates (i.e., benign samples that have been detected as malicious) evaluating the detection ability of our model. On the other hand, regarding the classification process, the benign samples are not needed, as in such procedures a classification model only has to decide the family in which a sample, already distinguished as malicious, belongs to.

4.1.2. Graph structures

The major target of our approach is to utilize the graphs that depict the temporal evolution of the produced GrG graphs (i.e., GrTG graphs) in order to measure the graph similarity among test sample and samples that have been already detected as malicious, leveraging their structural modification that take place during the execution time of the programs that they represent. In our work we have a theoretically stable intuition that the factor of time, regarding the structural evolution of a graph is a strong qualitative characteristic that could definitely distinguish the behaviour of a program and further be utilized to the development of more elaborated detection and classification techniques over unknown samples.

To this end, we ought to notice that regarding the time quantization procedure, where the time slots where the graph instances have to be retained, there could be applied several different approaches, that would affect the application results. In other words, the implementation of our proposed model on a fine-grained time quantization scheme, would be more precise against a more coarse-grained once, where on the other hand a trade-off between the precision on temporal structural modifications and the construction of more distinguishing patterns poses the basis of further tuning issues.

4.1.3. Measuring the similarity between temporal graphs

Next, we discuss the computation of similarity between the temporal graphs of a given test sample τ, let $T(G^*_\tau)$, and a known malicious sample s, let $T(G^*_s)$. In our approach, we distinguish three types of characteristics that depict the similarity between graph objects with respects to the edges, the weights and the overall structure of the corresponding GrG graphs G^*_τ and G^*_s, as they are evolved over time. As we show next, the instances (i.e., Temporal Graphs) of the GrG graphs through a series of

n `regions`, let $t_1, t_2, \ldots, t_n$, are denoted as $T_1(G^*_\tau), T_2(G^*_\tau), \ldots, T_n(G^*_\tau)$, for the test sample τ, and $T_1(G^*_s), T_2(G^*_s), \ldots, T_n(G^*_s)$, for the known malicious sample s, respectively.

A. Similarity Metrics

Next, we present the similarity metrics deployment for the measurement of different structural characteristics, i.e., relational, qualitative, and quantitative characteristics, exhibited through graphs.

- **Measuring Relational Characteristics** The relational characteristics refer to the structure of a given graph regarding its edges. Hence, given two graphs (in our case temporal graphs of a known and a test sample) we measure the similarity w.r.t. their relational characteristics utilizing the Jaccard Index applying it over the existence, or not, of the edges in both graphs. Note that, since the graphs are labelled graphs, we can indicate each edge between any pair of corresponding vertices. Hence the computation of Jaccard Similarity can be computed by the fraction of the common edges (intersection of the edges) between the temporal graphs for the test and the known samples and the total edges (union of the edges) of the two graphs, as follows:

$$J\left(T\left(G^*_\tau\right), T\left(G^*_s\right)\right) = \frac{|E(T(G^*_\tau)) \cap E(T(G^*_s))|}{|E(T(G^*_\tau))| + |E(T(G^*_s))| - |E(T(G^*_\tau)) \cap E(T(G^*_s))|}, \tag{1}$$

  where $E(T(G^*_\tau))$ and $E(T(G^*_s))$ refer to the edge sets of $T(G^*_\tau)$ and $T(G^*_s)$, respectively.
- **Measuring Qualitative Characteristics** The qualitative characteristics mainly concerning the structural likeness between two given graphs w.r.t to the importance of the corresponding vertices regarding both their in/out-degree (i.e., number of directed arcs) and their respective in-/out-weight. So, given two graphs (in our case temporal graphs of a known and a test sample) we measure the similarity w.r.t. their quantitative characteristics utilizing the Δ-Similarity metric as defined in [44] and deploy it over the temporal graphs of a test and a known sample as follows:

$$\Delta\left(T\left(G^*_\tau\right), T\left(G^*_s\right)\right) = \frac{\Gamma}{\Gamma + \delta(T(G^*_\tau), T(G^*_s))}, \tag{2}$$

  where Γ is a constant factor, in order for the similarity metric to return values in the range [0, 1], and $\delta(\cdot)$ refers to the Euclidean distance between the corresponding vertices of the two given graphs. Note that, we can also utilize exclusively either only the in-, or the out-degrees (respectively, the in-/out-weights) of the given graphs.
- **Measuring Quantitative Characteristics** The quantitative characteristics refer to the weights of a given graph regarding the relation between the weights of edges that both graphs have in common. So, given two graphs (in our case temporal graphs of a known and a test sample) we measure the similarity w.r.t. their quantitative characteristics utilizing the Cosine Similarity applying it over the weights of corresponding edges formed during the evolution of the GrG graph depicted by its temporal graph GrTG. Cosine Similarity is measured as follows:

$$CS\left(T\left(G^*_\tau\right), T\left(G^*_s\right)\right) = \frac{\sum_{i=1}^{n} w(e_{\tau,i}) \times w(e_{s,i})}{\sqrt{\sum_{i=1}^{n} w(e_{\tau,i})^2}\sqrt{\sum_{i=1}^{n} w(e_{s,i})^2}}, \tag{3}$$

  where $w(e)$ refers to the weight of the edge $e = (u, v)$, and the edge e_i indicates the ith edge in both temporal graphs (i.e., $e_{\tau,i}$ and $e_{s,i}$ for the GrTG of the test and the GrTG of the known sample,

respectively) among the corresponding vertices $u_\tau, v_\tau \in V(T(G^*_\tau))$ and $u_s, v_s \in V(T(G^*_s))$, such that $(u_\tau, v_\tau) \in E(T(G^*_\tau))$ corresponds to the edge $(u_s, v_s) \in E(T(G^*_s))$

B. Measuring the Similarity between Graph Segments

Next, in order to compute the similarity S between the temporal graphs of a test and a known malicious sample, let $T(G^*_\tau)$ and $T(G^*_s)$, over n `regions` we compute the corresponding similarity between the corresponding temporal graphs and average the exhibited values over each `region`, let $t_1, t_2, \ldots, t_n$, as follows:

$$S\big(T\big(G^*_\tau\big), T\big(G^*_s\big)\big) = \sum_{t=1}^{n} \frac{S(T_t(G^*_\tau), T_t(G^*_s))}{n}, \tag{4}$$

where the $S(T_t(G^*_\tau), T_t(G^*_s))$ refers to the similarity between the temporal graphs of the test and the known sample as they have been formed until the ith `region`.

Similarly, in order to compute the similarity between the temporal graphs of the test and the known sample as they have been formed over n `regions` regarding their relational characteristics, we compute the Jaccard Similarity as:

$$J\big(T\big(G^*_\tau\big), T\big(G^*_s\big)\big) = \sum_{t=1}^{n} \frac{J(T_t(G^*_\tau), T_t(G^*_s))}{n}, \tag{5}$$

On the other hand, w.r.t the computation of similarity regarding the quantitative characteristics exhibited between the temporal graphs of the test and the known sample as they have been formed over n `regions`, we compute the Cosine Similarity as:

$$CS\big(T\big(G^*_\tau\big), T\big(G^*_s\big)\big) = \sum_{t=1}^{n} \frac{CS(T_t(G^*_\tau), T_t(G^*_s))}{n}, \tag{6}$$

Finally, the computation of the similarity concerning the qualitative characteristics between the temporal graphs of the test and the known sample as they have been formed over n `regions`, we compute the Δ-Similarity metric as follows:

$$\Delta\big(T\big(G^*_\tau\big), T\big(G^*_s\big)\big) = \sum_{t=1}^{n} \frac{\Delta(T_t(G^*_\tau), T_t(G^*_s))}{n}, \tag{7}$$

where, the Δ-Similarity metric is denoted as Δ^--Similarity when only the out-degree and out-weights are taken into account, or Δ^+-Similarity, when utilized only the in-degree and in-weight.

To this end, we ought to notice that the similarity measurements mentioned above can be computed utilizing either the discrete modification temporal graphs, or the cumulative modification temporal graphs, at their basis. Hence, for these cases the general formula for the similarity computation is developed as follows:

$$S\big(\widehat{T}\big(G^*_\tau\big), \widehat{T}\big(G^*_s\big)\big) = \sum_{t=1}^{n} \frac{S(\widehat{T}_t(G^*_\tau), \widehat{T}_t(G^*_s))}{n}, \quad \text{or} \tag{8a}$$

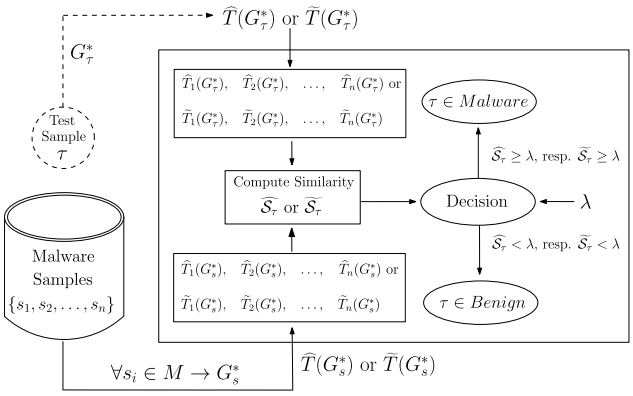


Fig. 4. Architecture of the detection model.

$$S\big(\widetilde{T}(G^*_\tau), \widetilde{T}(G^*_s)\big) = \sum_{t=1}^{n} \frac{S(\widetilde{T}_t(G^*_\tau), \widetilde{T}_t(G^*_s))}{n}, \tag{8b}$$

where, for a given test sample τ and a known malicious sample S, the $\widehat{T}_t(G^*_\tau)$ and $\widehat{T}_t(G^*_s)$ denote the corresponding temporal instances of their G^* evolved utilizing the discrete modification on the `region` t, while the $\widetilde{T}_t(G^*_\tau)$ and $\widetilde{T}_t(G^*_s)$ denote the corresponding temporal instances of their G^* on the `region` t, evolved utilizing the cumulative modification.

4.2. Malware detection

We implement our malware detection model by first performing a transformation to the initial ScDG graphs G, converting them to GrG graphs G^*, and then we compute for these graphs their corresponding Temporal Graphs (i.e., in our case the $\widehat{T}(G^*)$ and the $\widetilde{T}(G^*)$ graphs, respectively) referencing the discrete modification and the cumulative modification temporal graphs, respectively) as we described in the previous section, and then, for any given test sample we follow the same procedure as to conclude with the computation of a similarity metric in order to measure the similarities between the graphs of two samples, i.e., a given test sample τ, let $\widehat{T}(G^*_\tau)$, or $\widetilde{T}(G^*_\tau)$, and a known malicious sample s, $\widehat{T}(G^*_s)$, or $\widetilde{T}(G^*_s)$, with regarding specific types of their characteristics. Next, we describe the main process of determining if an unknown sample is malicious or benign based on the results of our similarity metrics when applied on the corresponding Temporal Graph of a test sample and a set of Temporal Graphs that represent known malicious software samples. In Fig. 4 we depict the total architecture of our proposed model for detecting malicious software samples.

In the example of Fig. 4 we suppose we are given an unknown test sample τ that we do not know if it is malicious, and we are asked to decide if τ is malicious or benign. Having a database with the Temporal Graphs of known malware samples. Once the corresponding Temporal Graphs have been constructed, we compute the our similarity metrics between the Temporal Graph of τ and each Temporal Graph that represents a malware sample in our database. So, let M the total number of malware samples in our database, we result to M values in our measurements on our similarity metrics (one per pair $\tau - s_i$). From the S total number of similarity values we select the maximum one, exhibited from a sample, let s_i, in order to distinguish the maliciousness of the test sample τ by its maximum similarity resulted

with s_i. Obviously, in case we depict the evolution of the graph by their temporal instances constructed utilizing the discrete modifications, the maximum similarity is computed as follows:

$$\widehat{\mathcal{S}}_\tau = \max\left\{S\left(\widehat{T}\left(G_\tau^*\right), \widehat{T}\left(G_s^*\right)\right), \forall 1 \leqslant s \leqslant M, s \in \mathbb{N}\right\}, \tag{9}$$

while, if we depict the evolution of the graph by their temporal instances constructed utilizing the cumulative modifications, the maximum similarity is computed as follows:

$$\widetilde{\mathcal{S}}_\tau = \max\left\{S\left(\widetilde{T}\left(G_\tau^*\right), \widetilde{T}\left(G_s^*\right)\right), \forall 1 \leqslant s \leqslant M, s \in \mathbb{N}\right\}. \tag{10}$$

Finally, in order to distinguish if the test sample τ is malicious, or not, we compare the corresponding maximum similarity exhibited between τ and a known malicious sample s_i, and if it is above a predefined threshold $\lambda \in (0, 1)$ and $\lambda \in \mathbb{R}$, it indicates that τ is malicious, or benign otherwise as depicted by the following rule:

$$\tau = \begin{cases} \texttt{malicious}, & \text{if } \widehat{\mathcal{S}}_\tau \geqslant \lambda \text{ (respectively, } \widetilde{\mathcal{S}}_\tau \geqslant \lambda) \\ \texttt{benign}, & \text{otherwise} \end{cases} \tag{11}$$

4.3. Malware classification

Our proposed method is based on application our proposed similarity metrics over the set of known malware families in order to classify on them an unclassified malware sample, let τ. More precisely, our method selects the family that is most similar to τ according to the similarity results exhibited by the measurements performed utilizing the similarity metrics. A malware family is called `dominant` family, if it contains as a member the sample that produce the maximum similarity exhibited among it and the test sample. More precisely, using our proposed similarity metrics, we iterate over all the members of all the known malware families measuring the similarity between each pair of τ, s_{ij}, where s_{ij} is the jth member of the ith malware family (i.e., F_i). Then, for each family we select a member, let s, that is the most similar to τ, according to maximum similarity exhibited either utilizing the $\widehat{\mathcal{S}}_\tau$, or the $\widetilde{\mathcal{S}}_\tau$ for discrete or cumulative modification temporal graphs respectively, calling this member `representative` sample for this specific family. Finally, among all the `representative` samples of all the known malware families, we select to classify the unclassified test sample τ to a malware family that its `representative` sample, exhibits the maximum similarity with τ, denoting this family as `dominant family`.

In Fig. 5 we show a representation of the procedure for classifying an unknown test sample τ to a known malware family utilizing the aforementioned methods. More formally, our classification technique proceeds as follows: given a set of known malware families $F_1, F_2, \ldots, F_N\}$ and an unclassified malware sample τ, we measure either $\widehat{\mathcal{S}}_\tau$, or the $\widetilde{\mathcal{S}}_\tau$ over all the members of each family, keeping the maximum result (i.e., `representative` sample) for each family resulting to N results (i.e., N `representative` samples), one per family. Then, we classify the test sample to the family that exhibited the maximum value among all results. In other words, we compute the aforementioned similarity metrics between τ and all the malware families of the data-set, selecting as the `dominant family`, the one that has the `representative` sample that exhibits the maximum value in our similarity measurements.

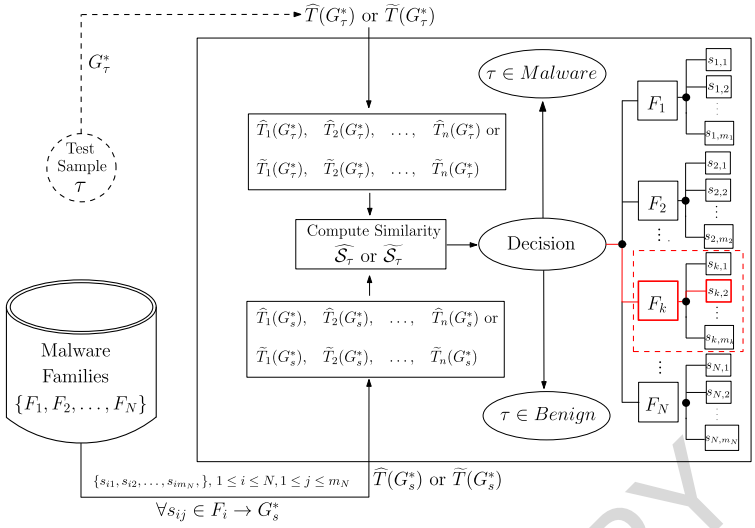


Fig. 5. Architecture of the classification model.

4.4. Framework

The deployment model of the proposed components is consisted by the graph structures utilized to represent the software's behavioral characteristics (i.e., the Temporal Graphs that represent their temporal evolution through time), the knowledge base, that is a database storing Temporal Graphs representing known malware samples, and the similarity metrics developed to capture structural and qualitative commonalities among such behavioral graphs. Our proposed graph-based malware detection and classification model is partitioned into two phases. The first phase concerns the detection procedure, where an unknown sample, let τ, is needed to be detected as malicious or benign. Our model's implementation utilizes the Temporal Graphs taken from a database of known malware samples and the Temporal Graph of test sample τ in order to compute their structural similarities across their temporal evolution. The second phase concerns the classification procedure, where an unknown sample, let τ, that has been already detected as malicious is needed to be classified to one of a set of known malware families. Our mode's implementation utilizes the Temporal Graphs taken from a database of known malware samples already been classified to malware families and the Temporal Graph of test sample τ in order to compute their structural similarities across their temporal evolution and further classify τ to one malware family from our data-set. In Fig. 6, we represent an abstract overview of the deployment of our proposed graph-based model for malware detection and classification.

5. Evaluation

In this section we present the experimental set-up utilized for the evaluation of our proposed graph-based technique for malware detection and classification utilizing the discrete modification temporal graphs and the cumulative modification temporal graphs, and discuss the exhibited results for the two procedures, namely malware detection and malware classification, presenting the potentials of our framework through a series of experiments over a data-set of malicious and benign samples.

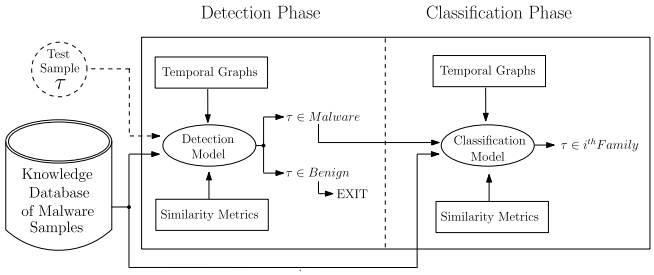


Fig. 6. The deployment of malware detection and malware classification processes in our model.

5.1. Experimental set-up

Next, we present the methodology followed for the evaluation of our framework on detecting and classifying malicious samples, the data-set of the known and the test samples, the methodology we followed in order partition the data-set to train and test set and the structure of a series of evaluation experiments, and finally the experimental design regarding the utilization of different similarity metrics over the two distinct types of temporal graphs.

5.1.1. Data-set

In order to evaluate the first phase of our proposed framework against malicious samples, regarding its detection potentials we utilized a data-set of 2631 malicious samples, and a particularly for the evaluation of the false-positive rates of the detection procedure a set of 35 benign samples that cover a wide range of commodity software types (i.e., commodity software types including editors, office suites, media players, browsers, standardized procedures, etc.). To this end, we ought to notice that the diversity of the benign sample data-set covers a wide range of commodity software samples, compensating the number of malicious samples, regarding that in the benign ones there exist actually one sample of each kind, or in other words singleton (i.e., one member per family) benign families. For the second phase of our proposed framework regarding its ability to index the detected samples to known malware families, we utilized the grouping of the 2631 malicious samples into 48 distinct malware families including from 5 to 317 samples pre-classified into them through heuristic rules as described in [3]. Next, in Table 1 we list the set of the 48 malware families along with their sizes (i.e., number of members).

Our proposed framework operates on graph representations of software samples (malicious or benign), the so called System-call Dependency Graphs obtained through dynamic taint analysis over their execution in a contained environment. These graphs (i.e., ScDG graphs) containing System-call dependencies among the invoked System-calls are transformed to their corresponding GrG graphs by either adding an edge between the corresponding groups of the pairs of each invoked System-calls (if this edge does not already exist) or by increasing by one unit the weight of this edge (if this edge already exists). Hence the ScDG graphs that constitute our initial data-set, and thus the train/test sets utilized for the evaluation of our approach, are then transformed to their corresponding Group Relation Graphs, (i.e., GrG graphs) following the procedure described in Section 3.1. To this point it is quite important to refer that the initial data-set that contains the ScDG graphs from malicious sample pre-indexed into malware families and the set of benign samples, is constructed by behavioural graphs that their vertex-sets contain $\lceil V(G) \rceil = 494$ System-calls, while their corresponding edge sets may contain $\lceil E(G) \rceil = 244.036$ System-call invocations. Whoever, transforming the initial data-set of ScDG graphs into their corresponding GrG graphs, it results to the creation of behavioural graphs that their vertex-sets contain $\lceil V(G^*) \rceil = 30$ System-calls, while their corresponding edge sets may contain $\lceil E(G^*) \rceil = 900$ System-call Group Relation.

Table 1
The set of the 48 malware families $F_1, F_2, \ldots, F_{48}$ provided by Babic *et al.*, along with their sizes (i.e., number of members), as used in [3]

Family Name	Size	Family Name	Size	Family Name	Size
ABU, Banload	16	DNSCGR, DNSCGR	22	OLG, Mmorpg	19
Agent, Agent	42	Downloader, Agent	13	OLG, OLG	23
Agent, Small	15	Downloader, Delf	22	Parite, Pate	71
Allaple, RAHack	201	Downloader, VB	17	Plemood, Pupil	32
Ardamax, Ardamax	25	Gaobot, Agobot	20	PolyCrypt, Swizzor	43
Bactera, VB	28	Gobot, Gbot	58	Prorat, AVW	40
Banbra, Banker	52	Horst, CMQ	48	Rbot, Sdbot	302
Bancos, Banker	46	Hupigon, ARR	33	SdBot, SdBot	75
Banker, Banker	317	Hupigon, AWQ	219	Small, Downloader	29
Banker, Delf	20	IRCBot, Sdbot	66	Stration, Warezov	19
Banload, Banker	138	LdPinch, LdPinch	16	Swizzor, Obfuscated	27
BDH, Small	5	Lmir, LegMir	23	Viking, HLLP	32
BGM, Delf	17	Mydoom, Mydoom	15	Virut, Virut	115
Bifrose, CEP	35	Nilage, Lineage	24	VS, INService	17
Bobax, Bobic	15	OLG, Delf	11	Zhelatin, ASH	53
DKI, PoisonIvy	15	OLG, LegMir	76	Zlob, Puper	64

Throughout this approach, it is achieved a more abstract and mutation resilient representation of behavioral graphs, retaining not only the valuable information regarding the intentions of the software sample under consideration depicted by its behaviour, while, on the other hand, reduces the order and the size of the graph and the corresponding sub-graph areas that have to be examined in order to detect similarities among known and unknowns samples.

Hence, having defined a number of `regions` over which a temporal instance of a given GrG is developed, we construct two types of temporal graphs, either by selecting distinct or accumulating the modifications performed until the ith `region`, constructing the Discrete Modification Temporal Graphs (i.e., DMTG graphs) or the Cumulative Modification Temporal Graphs (i.e., CMTG graphs), respectively. Following this procedure we can transform the initial data-set of ScDGs into a set of corresponding DMTGs, or CMTGs, by registering the modifications performed on the corresponding GrG over is evolution during time, taking into account the time sequence the system-calls were performed (i.e., chronological order of system-call invocation) during the execution of the program.

5.1.2. Methodology

In order to evaluate the detection and classification ability of our framework, we perform a five-fold cross validation procedure, partitioning the data-set into a 80% train-set and a 20% test-set, rotating the samples over each iteration. These samples refer to the graph objects that are consisted by the types of temporal graphs, i.e., CMTG and DMTG, that resulted by registering the modification over the evolution of a GrG graphs according to its corresponding ScDG graph, with respect to both the temporal graph type and the `regions` that the GrG graph is partitioned into. The results exhibited over the evaluation procedure are averaged over the five folds and are extracted over a series of experiments for different number of `regions`, different similarity metrics and various values of threshold. In particular, the series for experiments that regarding the evaluation of the detection potentials of our model concern the 2631 malicious samples in order to be distinguished as malicious or benign, alongside the benign samples utilized for the measurement of the false-positive rates, while for the evaluation of the classification

accuracy we utilized the indexing of the malware samples into malware families by their appended labels in order to measure the correctly classified detected samples.

Regarding the evaluation of our detection ability model is achieved by performing a series of five-fold cross validation experiments over different similarity metrics that measure the graph-similarity regarding different types of graph characteristics (i.e., relational, quantitative, qualitative) and various number of `regions` that the initial GrG graph is partitioned with respect to the underlying methodology for the depiction of its temporal evolution (i.e., discrete modification temporal graphs, and cumulative modification temporal graphs). In each series of experiments we compute the exhibited results averaged over the five folds, investigating the performance of our model over a specific similarity metric in a range of `regions` (i.e., one number of `region` per five-fold experiment). More precisely, for a given fold (i.e., 20% train-set) we compare each sample τ of that set, with each sample s of our knowledge base (i.e., 80% train-set) computing the similarity between them, as presented in Equations (8a), and (8b). Then, iterating the same procedure computing the similarity between τ and each sample s of the train-set, regarding the underlying procedure for the evolution of temporal graphs, we compute the maximum similarity exhibited, according to Equation (9), or Equation (10) regarding the utilization of the discrete modification and the cumulative modification temporal graphs, respectively. Hence, having the maximum similarity exhibited among the test sample τ and a sample s, we apply the detection rule presented in Equation (11), comparing it with a pre-defined threshold value λ in order to distinguish if the test sample τ is malicious, or benign otherwise.

Then, this procedure is iterated over various values of threshold λ in order to depict the behaviour of our model across this parameter. The procedure mentioned so far, iterates over a different number of `regions`, note that, in our model we experimented with 2^n, $1 \leqslant n \leqslant 8$ numbers of `regions` (i.e., $2, 4, \ldots, 256$ `regions`). Having so far a series of experiments for a range of `regions` we can repeat the experiment utilizing a similarity metric that measures the similarity between different characteristics of the graphs under consideration. Next, we present the achieved results grouped over the two distinct approaches for the temporal evolution of the GrG graphs, (i.e., utilizing the discrete modification and the cumulative modification temporal graphs, respectively), then for the deployment of different similarity metrics (i.e., Jaccard index, Cosine Similarity, and Δ-Similarity metric) in each group of experiments, and in each case iterating over the specific range of `regions` (i.e., $2, 4, \ldots, 256$), averaging the results of each case over the five folds.

Similarly,in each case of experiment, except the maximum similarity exhibited between the sample under consideration, i.e., τ, and each of the known malicious samples, let s, having the labels of each known malicious sample, let $\ell(S_{ij})$ indicating that this sample is the jth member of the ith malware family, we list the `dominant member` of each malware family and selecting the one with the maximum similarity exhibited between this sample s_{ij} and τ we can result to the corresponding `dominant family` that is the family (i.e., $\ell(S_{ij})$)in to which τ will be classified. The attestation of a correct classification is concluded by comparing the family of the most similar sample, let $\ell(s_{ij})$, with the actual family of τ, let $\ell(\tau)$, which is hidden for the evaluation purposes. Then, integrating this classification procedure as the second phase of our framework, we compute the averaged results over the five folds, as described, for each case of experiment. To this end, we ought to notice that in our experiments, we distinguish three different types of classification result evaluation, taking into account the case where the previously detect as malicious sample τ is classified by our model into a family that is similar to its actual family (attested by the comparing a relation between $\ell(\tau)$ and $\ell(S_{ij})$).

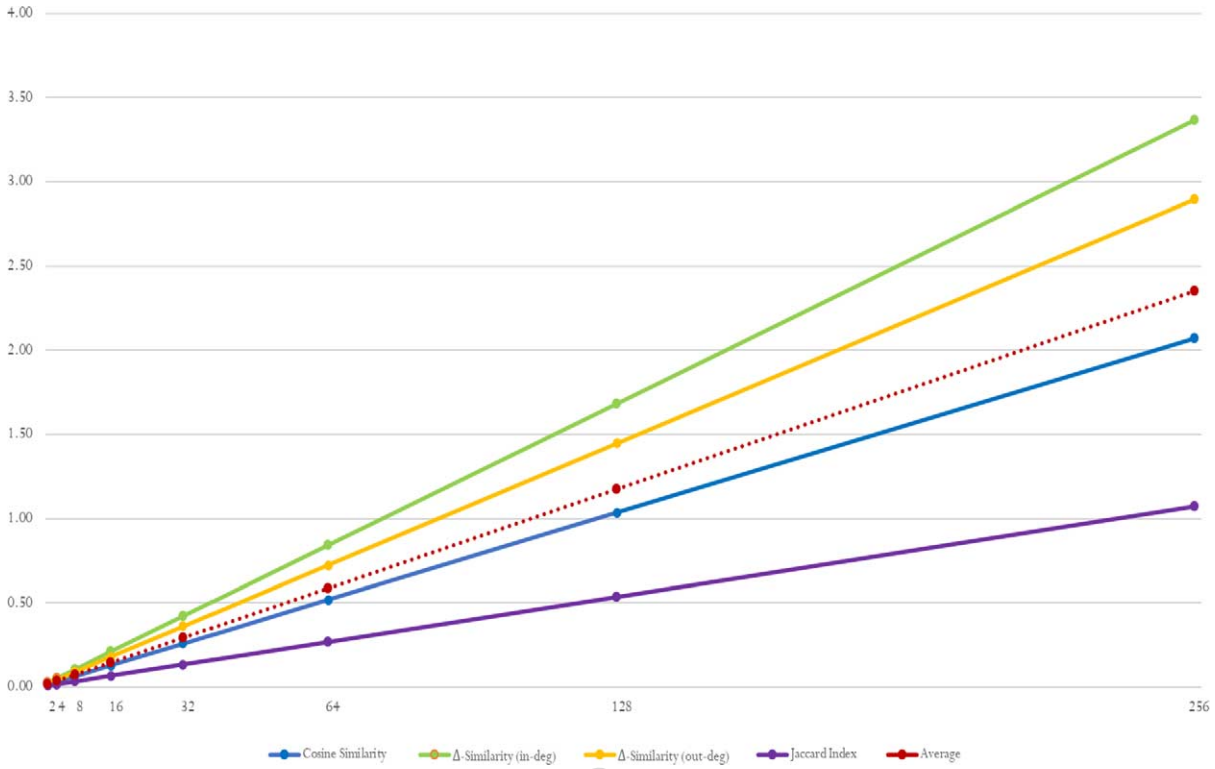


Fig. 7. Execution time (in hours) required from the framework to perform a five-fold experiment for 8 series of `regions` deploying both the utilization of DMTG, and CMTG graphs.

5.1.3. System performance

Throughout the evaluation of our framework we performed a five-fold cross validation series of experiments iterating over the 8 different series of `regions` (i.e., 2, 4, . . . , 256 `regions`), utilizing four different similarity metrics, namely the Cosine Similarity, the Jaccard index, the Δ-Similarity with only the in-degree/weights and the Δ-Similarity with only the out-degree/weights, over the two distinct types of temporal graphs (i.e., DMTG and CMTG, respectively) resulting to a series of 320 experiments. With each experiment of a file-fold cross validation containing (5×2631) 13155 malicious samples and utilizing more than 2.5k as test set for a full series of `regions` i.e., 2, 4, . . . , 256 `regions`)over the two distinct types of temporal graphs the total execution time took on average less 2.5 hours depending in each case on the similarity metric deployed. As we can see in Fig. 7 our model performs linearly over the increase of the number of `regions` while an increase in the time demanded for a full five-fold cross validation for the two types of temporal graphs is caused exclusively by the similarity metric deployed.

To this end, note that in the corresponding plot the x-axis represents the values the `regions` a temporal graph is partitioned into, while the y-axis represents the time (in hours) required for a complete execution of the two phases (i.e., detection and classification) over the five-folds, while the solid color line represent the corresponding execution time required by the deployment of different similarity metrics and the doted line represents the average time required. Through this representation we can observe that only the application of the Δ-Similarity metric lies above the average required time since it is the only similarity that requires to construct an auxiliary data structure for the storage of the demanded information (i.e., the in- or out-degrees and the in- or out-weights of each vertex of the corresponding

temporal graphs in order to measure their Euclidean distance on their planar projection. It is notable to refer that totally our system's implementation in Java performing the experiments on a commodity computer with 9th generation Intel i7 with 16 Gb of RAM detecting and classifying almost 2.5k test samples in almost two and a half hours.

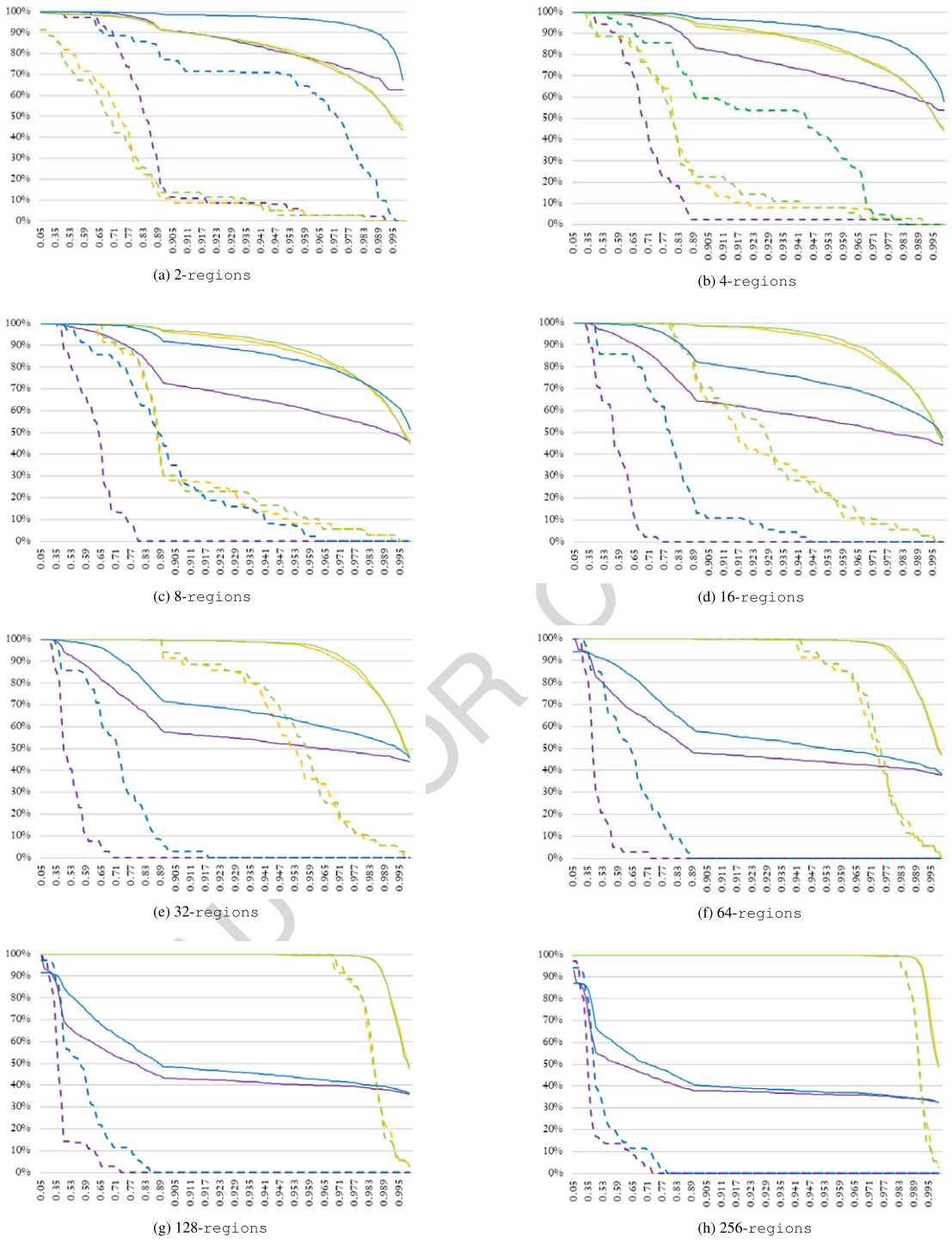
5.2. Evaluating the detection ability

In the next experimental study of our model, we utilize the data-set discussed in previous section in order to perform a five-fold cross validation utilizing in each case the 80% of the data-set as train-set and the rest 20% of the data-set as test-set. In the following experiments we distinguish two classes of a series of experiments performed over different settings. Namely, the first class of experiments contains the evaluation of our model deploying the discrete modification temporal graphs (DMTG) for the representation of the samples, while the second class of experiments contains the evaluation of our model deploying the cumulative modification temporal graphs (CMTG) for the representation of samples. Additionally in each class of experiments we conduct a series of experiments utilizing different similarity metrics that compare the graph structures based on different characteristics (i.e., the relations, the qualitative, and the quantitative characteristics), deploying the Jaccard index, the Δ-Similarity with in-out degree and in-out weights (i.e., Δ^{+}-Similarity, and Δ^{-}-Similarity), and the Cosine Similarity metrics, respectively. Moreover, during each series of experiments we compute the corresponding similarity metric between temporal graphs of a particular type (i.e., DMTGs or CMTGs), for various numbers of `regions`, i.e., we perform in each case a five-fold cross validation experiment for each given number of `regions` from the set $2, 4, \ldots, 256$. Finally, in the next figures we depict the exhibited results regarding the averaging the exhibited True-Positive (TP) (i.e., malicious that are not detected as malicious) and False-Positive (FP) (i.e., benign that are detected as malicious) rates over the five folds. In the plots presented in the following figures, the x-axis represents the values of threshold λ, the y-axis represents the averaged percentages of the malicious samples been detected as malicious (i.e., TP-rate) depicted in red-line, and the averaged percentages of the benign samples been detected as malicious (i.e., FP-rate) drawn in green-line. The procedure followed for the detection of software samples as malicious or benign, compared to known malicious samples,is the one described in Section 4.2, see Fig. 4.

5.2.1. Detecting malicious samples utilizing DMTG

In this class of experiments we evaluate the detection potentials of our model, utilizing the discrete modification temporal graphs (DMTG) for the representation of software samples under consideration in order to represent the temporal evolution of its corresponding GrG graph. In the experiments discussed next, we measure the graph similarity of between any sample form a set of unknown samples against a set of known malicious sample. Particularly, there are examined four types of similarity measurements between DMTG graph representations, the one that concerns the measurement of similarity regarding the relational characteristics of graphs through the utilization of Jaccard Similarity, the next that concerns the measurement of similarity regarding the quantitative characteristics through the utilization of the Δ-Similarity metric, and finally the measurement of similarity regarding the qualitative characteristics through the utilization of Cosine similarity metric. In each case we perform 8 five-fold cross validation experiments, one for any number of `regions` that the DMTG graph is partitioned, examining the result that represent the detection potentials of our model with respect to the corresponding settings. To this end we recall that in the next plots the x-axis represents the values of threshold λ, the y-axis represents the averaged percentages of the TP-rates (red-line) and FP-rates (green-line).

- Detection using Relational Characteristics of DMTG
  In Fig. 8, with purple continuous and dashed lines for TP-rates and FP-rates, repetitively, we cite the results exhibited by the application of Jaccard Similarity in order to measure the similarity regarding the relational characteristics appeared in common in the graphs of malicious and benign test samples against known malicious samples, all represented by their discrete modification temporal graphs (DMTG). The exhibited difference between TP and FP rates is maximized in the following cases: 2 `regions` when $\lambda = 0.919$ (i.e., FP $= 89\%$, TP $= 9\%$), for 4 `regions` when $\lambda = 0.88$ (i.e., FP $= 86\%$, TP $= 2\%$), for 8 `regions` when $\lambda = 0.8$ (i.e., FP $= 89\%$, TP $= 0\%$), for 16 `regions` when $\lambda = 0.69$ (i.e., FP $= 85\%$, TP $= 0\%$), for 32 `regions` when $\lambda = 0.6$ (i.e., FP $= 80\%$, TP $= 0\%$), for 64 `regions` when $\lambda = 0.57$ (i.e., FP $= 70\%$, TP $= 0\%$), for 128 `regions` when $\lambda = 0.5$ (i.e., FP $= 55\%$, TP $= 0\%$), and for 256 `regions` when $\lambda = 0.73$ (i.e., FP $= 44\%$, TP $= 0\%$). As we can observe through the exhibited results the increase of the number of `regions` results to a more sharp decrease in the number of false-positive detections than the number of true-positive detections. Additionally an interesting observation results from the fact that as we can see from the plots, the increase on the number of `regions` makes the FP rates to decay to nearly zero values from very low values of threshold λ, while on the other hand, the TP rates tend to be stabilized in lower values across the increase of λ.
- Detection using Qualitative Characteristics of DMTG
  In Fig. 8, with orange/green continuous and dashed lines for TP-rates and FP-rates, repetitively, we present the results exhibited by the application of Δ-Similarity metric in order to measure the similarity regarding the qualitative characteristics appeared in common in the graphs of malicious and benign test samples against known malicious samples, all represented by their discrete modification temporal graphs (DMTG). Since the GrG graphs are directed weighted graphs, over the corresponding experiments we distinguished two cases where in order to leverage the valuable information depicted in the degree and the weight of each vertex, and respectively how they are evolved during time (i.e., modifications performed during a `region`, we performed two distinct experiments, where in the first case we take into account of only the out-degree and the out-weight on each vertex, defining hence the Δ^--Similarity, whose detection results are denoted by the orange lines, while in the second case we take into account of only the in-degree and the in-weight on each vertex, defining hence the Δ^+-Similarity, whose detection results are denoted by the green lines. As we can observe through the exhibited results of Fig. 8 the proposed framework regarding the detection rates performs quite similar concerning the application of either Δ^--Similarity or Δ^+-Similarity metric. However, a quite interesting observation results from the fact that despite that the behavior of the TP rates exhibits an increase through the increase of the number of `regions`, on the other hand, the FP rates in the increase over the 16 `regions` inverts the exhibited curve (i.e., from nearly concave curve to nearly convex curve) been stabilized even for higher values of λ decreasing with higher rate in higher values of λ. For the case of the application of Δ^--Similarity metric, the exhibited difference between TP and FP rates is maximized for 2 `regions` when $\lambda = 0.906$ (i.e., FP $= 91\%$, TP $= 9\%$), for 4 `regions` when $\lambda 0.922$ (i.e., FP $= 91\%$, TP $= 8\%$), for 8 `regions` when $\lambda = 0.952$ (i.e., FP $= 88\%$, TP $= 8\%$), for 16 `regions` when $\lambda = 0.96$ (i.e., FP $= 90\%$, TP $= 11\%$), for 32 `regions` when $\lambda = 0.977$ (i.e., FP $= 86\%$, TP $= 11\%$), for 64 `regions` when $\lambda = 0.984$ (i.e., FP $= 87\%$, TP $= 17\%$), for 128 `regions` when $\lambda = 0.989$ (i.e., FP $= 89\%$, TP $= 22\%$), and for 256 `regions` when $\lambda = 0.993$ (i.e., FP $= 91\%$, TP $= 29\%$). On the other hand, for the case of Δ^+-Similarity metric, the exhibited difference between TP and



(a) 2-regions (b) 4-regions

(c) 8-regions (d) 16-regions

(e) 32-regions (f) 64-regions

(g) 128-regions (h) 256-regions

Fig. 8. From (a) to (h) we depict the detection rates (continuous line) and the corresponding false positive rates (dashed line) for various regions using the Jaccard similarity (purple color), the Δ^--similarity (orange color), the Δ^+-similarity (green color), and the Cosine Similarity (blue color), on DMTG graphs.

FP rates is maximized for 2 regions when $\lambda = 0.948$ (i.e., FP = 82%, TP = 3%), for 4 regions when $\lambda = 0.943$ (i.e., FP = 88%, TP = 8%), for 8 regions when $\lambda = 0.964$ (i.e., FP = 84%, TP = 6%), for 16 regions when $\lambda = 0.966$ (i.e., FP = 89%, TP = 11%), for 32 regions when $\lambda = 0.974$ (i.e., FP = 90%, TP = 14%), for 64 regions when $\lambda = 0.985$ (i.e., FP = 87%, TP = 11%), for 128 regions when $\lambda = 0.989$ (i.e., FP = 89%, TP = 17%), and for 256 regions when $\lambda = 0.994$ (i.e., FP = 81%, TP = 14%). Comparing the results exhibited by the two case of application of the Δ-Similarity, after a detailed study on the exhibited results we concluded that the Δ^+-Similarity performs better than Δ^--Similarity for regions in the set $\{2, 4, 8, 16\}$, while for numbers of regions the Δ^--Similarity metric performs better since the difference between TP and FP rates is increased.

- Detection using Quantitative Characteristics of DMTG
  In Fig. 8, with blue continuous and dashed lines for TP-rates and FP-rates, repetitively, we cite the results exhibited by the application of Cosine Similarity deployed in order to measure the similarity regarding the qualitative characteristics appeared in common in the graphs of malicious and benign test samples against known malicious samples, all represented by their discrete modification temporal graphs (DMTG). The exhibited difference between TP and FP rates is maximized for 2 regions when $\lambda = 0.994$ (i.e., FP = 84%, TP = 3%), for 4 regions when $\lambda = 0.981$ (i.e., FP = 84%, TP = 0%), for 8 regions when $\lambda = 0.962$ (i.e., FP = 81%, TP = 0%), for 16 regions when $\lambda = 0.946$ (i.e., FP = 74%, TP = 0%), for 32 regions when $\lambda = 0.918$ (i.e., FP = 69%, TP = 0%), for 64 regions when $\lambda = 0.89$ (i.e., FP = 59%, TP = 0%), for 128 regions when $\lambda = 0.85$ (i.e., FP = 53%, TP = 0%), and for 256 regions when $\lambda = 0.79$ (i.e., FP = 47%, TP = 0%). Although, an interesting observation results from the fact that the TP rates exhibited a slower decay compared to the FP rates increasing the values of threshold λ across the number of regions. More precisely an interesting fact appears when observing that increasing the number of regions a more sharp decay is exhibited in the TP rates for lower values of λ, while the TP rates are decaying somehow later for greater values of λ resulting hence to greater differences among the TP and FP rates leading further to inspect promising aspects of that approach.

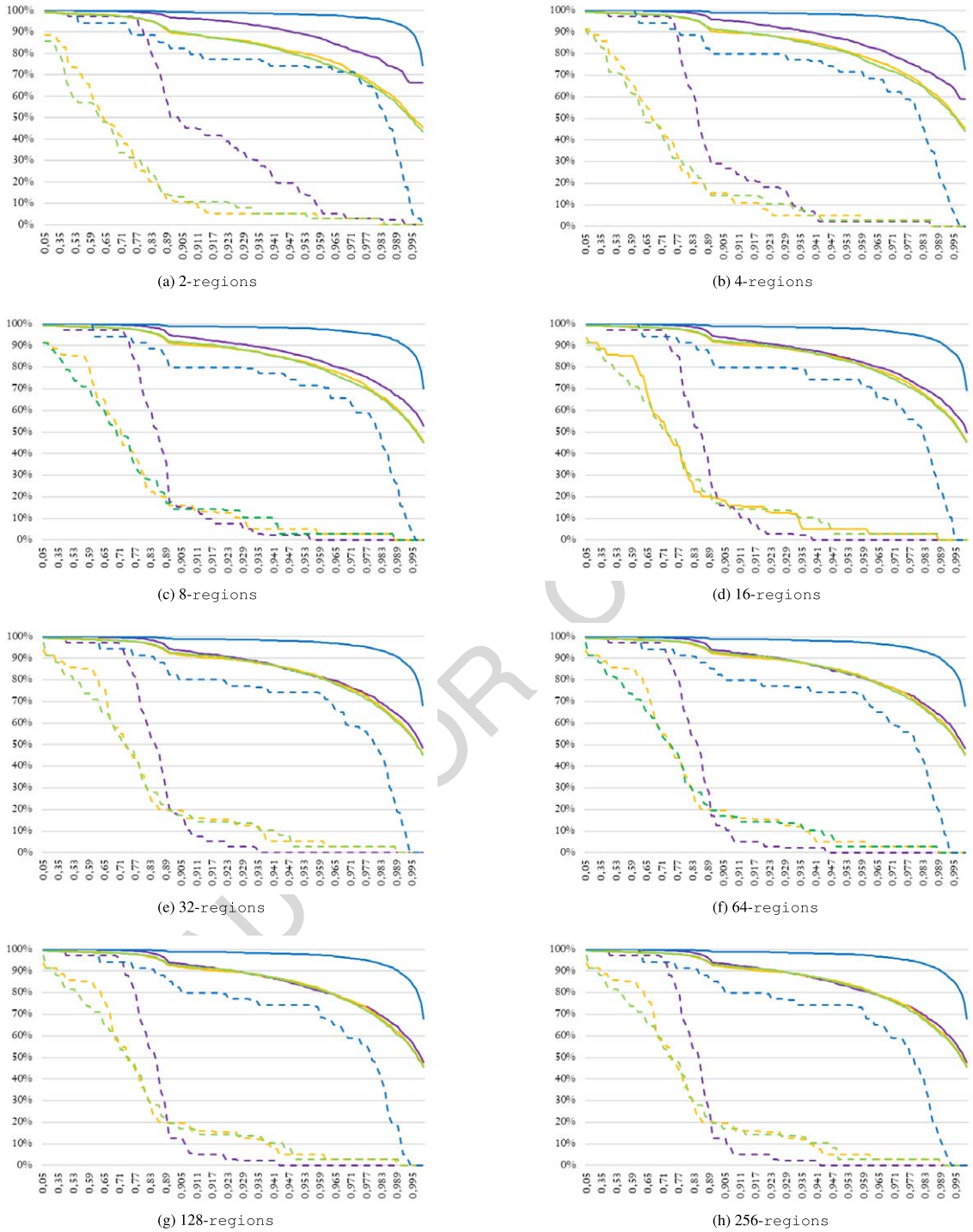
As we can observe from the exhibited results depicted in Fig. 8, the increase of the regions in the construction of the DMTG yields a more fine grained depiction of the temporal evolution of the corresponding Group Relation Graph (GrG) which lead us to deduct that the mutations in terms of structural modifications of a graph that represents, e.g., a malicious sample, intensify the structural alteration in more specific segments (i.e., regions that do capture the system-call invocation performed in particularly in that time interval) of the evolution of the corresponding System-call Dependency Graph (ScDG) during its execution through taint analysis.

5.2.2. Detecting malicious samples utilizing CMTG

In this class of experiments we evaluate the detection potentials of our model, utilizing the cumulative modification temporal graphs (CMTG) for the representation of software samples under consideration in order to represent the temporal evolution of its corresponding GrG graph. The procedure followed for the conduction of the second class of experiments is the same as the one described in the previous subsection. Hence, similarly to the previous class of experiments, there are examined four types of similarity measurements between CMTG graph representations, concerning the measurement of similarity regarding the relational,the quantitative, and the qualitative characteristics, utilizing the Jaccard Similarity, the Δ-Similarity, and the Cosine Similarity metric, respectively. Following the setting of the previous class of experiments, we perform 8 five-fold cross validation experiments, one for any number of regions

that the CMTG graph is partitioned, examining the result that represent the detection potentials of our model with respect to the corresponding settings.

- Detection using Relational Characteristics of CMTG
  In Fig. 9, similarly to the procedure followed for the experiments utilizing the CMTG graphs, with purple continuous and dashed lines for TP-rates and FP-rates, repetitively, we cite the results exhibited by the application of Jaccard index in order to measure the similarity regarding the exhibited relational characteristics between malicious and benign samples against known malicious samples represented by their cumulative modification temporal graphs (CMTG). The exhibited difference between TP and FP rates is maximized for 2 `regions` when $\lambda = 0.961$ (i.e., FP $= 86\%$, TP $= 5\%$), for 4 `regions` when $\lambda = 0.943$ (i.e., FP $= 89\%$, TP $= 2\%$), for 8 `regions` when $\lambda = 0.934$ (i.e., FP $= 89\%$, TP $= 3\%$), for 16 `regions` when $\lambda = 0.922$ (i.e., FP $= 91\%$, TP $= 3\%$), for 32 `regions` when $\lambda = 0.936$ (i.e., FP $= 88\%$, TP $= 0\%$), for 64 `regions` when $\lambda = 0.922$ (i.e., FP $= 91\%$, TP $= 3\%$), for 128 `regions` when $\lambda = 0.923$ (i.e., FP $= 91\%$, TP $= 3\%$), and for 256 `regions` when $\lambda = 0.924$ (i.e., FP $= 40\%$, TP $= 3\%$). The exhibited results prove that the FP rates are decreasing by the increase of the `regions` while the TP rates are in most of the cases kept in adequately high levels especially in values after $\lambda = 0.9$. Additionally an interesting observation results from the fact that while the TP rates decay when increasing the number of `regions` from lower values of λ, the detection ability of our proposed model seems not to be affected, since comparing the difference between TP and FP rates across the `regions` it results that it is increased even for lower values of λ.
- Detection using Qualitative Characteristics of CMTG
  In Fig. 9, with orange/green continuous and dashed lines for TP-rates and FP-rates, repetitively, we present the results exhibited measure the similarity regarding the exhibited qualitative characteristics between malicious and benign samples against known malicious samples represented by their cumulative modification temporal graphs (DMTG) by the application of Δ-Similarity metric. As in the corresponding series of experiments on CMTG graphs, two cases are distinguished, where in the first case the experiments are conducted utilizing the Δ-Similarity taking into account how the vertices of GrG graphs are evolved during time (i.e., accumulating modifications performed during a `region` regarding their the out-degree and out-weight, while in the second case regarding their the in-degree and in-weight, deploying hence the Δ^--Similarity and the the Δ^+-Similarity metrics, respectively. Similarly to the exhibited results of the application of Δ^--Similarity and Δ^+-Similarity metrics in DMTG graphs, whose detection results are denoted by the orange and the green lines, respectively, the application of these similarity metrics in CMTG graphs performs quite similar concerning the detection ability of the framework by the application of either Δ^--Similarity or Δ^+-Similarity metrics. For the case of the application of Δ^--Similarity metric, the exhibited difference between TP and FP rates is maximized for 2 `regions` when $\lambda = 0.917$ (i.e., FP $= 87\%$, TP $= 5\%$), for 4 `regions` when $\lambda = 0.924$ (i.e., FP $= 88\%$, TP $= 5\%$), for 8 `regions` when $\lambda = 0.931$ (i.e., FP $= 88\%$, TP $= 5\%$), for 16 `regions` when $\lambda = 0.935$ (i.e., FP $= 87\%$, TP $= 5\%$), for 32 `regions` when $\lambda = 0.938$ (i.e., FP $= 87\%$, TP $= 5\%$), for 64 `regions` when $\lambda = 0.941$ (i.e., FP $= 87\%$, TP $= 5\%$), for 128 `regions` when $\lambda = 0.943$ (i.e., FP $= 86\%$, TP $= 5\%$), and for 256 `regions` when $\lambda = 0.943$ (i.e., FP $= 86\%$, TP $= 5\%$). For the case of the application of Δ^+-Similarity metric, the exhibited difference between TP and FP rates is maximized for 2 `regions` when $\lambda = 0.933$ (i.e., FP $= 85\%$, TP $= 5\%$), for 4 `regions` when $\lambda = 0.943$ (i.e., FP $= 83\%$, TP $= 3\%$), for 8 `regions` when $\lambda = 0.943$ (i.e., FP $= 85\%$, TP $= 3\%$), for 16 `regions` when $\lambda = 0.947$ (i.e., FP $= 84\%$, TP $= 3\%$), for 32 `regions` when $\lambda = 0.948$ (i.e.,



(a) 2-regions (b) 4-regions

(c) 8-regions (d) 16-regions

(e) 32-regions (f) 64-regions

(g) 128-regions (h) 256-regions

Fig. 9. From (a) to (h) we depict the detection rates (continuous line) and the corresponding false positive rates (dashed line) for various regions using the Jaccard similarity (purple color), the Δ^{-}-similarity (orange color), the Δ^{+}-similarity (green color), and the Cosine Similarity (blue color), on CMTG graphs.

FP = 85%, TP = 3%), for 64 `regions` when $\lambda = 0.949$ (i.e., FP = 84%, TP = 3%), for 128 `regions` when $\lambda = 0.949$ (i.e., FP = 85%, TP = 3%), and for 256 `regions` when $\lambda = 0.949$ (i.e., FP = 85%, TP = 3%). A notable observation arises from the fact that the behaviour of the FP rates exhibits a stabilization in low levels for values of λ greater the 90 for the utilization of Δ^+-Similarity metric, while, for the case of Δ^--Similarity metric the FP rates are still decaying.

- Detection using Quantitative Characteristics of CMTG
  In Fig. 9, with blue continuous and dashed lines for TP-rates and FP-rates, repetitively, we cite the results exhibited by the application of Cosine Similarity deployed to measure the similarity regarding the exhibited qualitative characteristics between malicious and benign samples against known malicious samples represented by their discrete modification temporal graphs (DMTG). As we can observe through the exhibited results the increase of the number of `regions` results to a decrease so in the TP rates as also in the FP rates. The exhibited difference between TP and FP rates is maximized for 2 `regions` when $\lambda = 0.996$ (i.e., FP = 88%, TP = 3%), for 4 `regions` when $\lambda = 0.997$ (i.e., FP = 84%, TP = 0%), for 8 `regions` when $\lambda = 0.996$ (i.e., FP = 85%, TP = 0%), for 16 `regions` when $\lambda = 0.995$ (i.e., FP = 85%, TP = 0%), for 32 `regions` when $\lambda = 0.994$ (i.e., FP = 86%, TP = 0%), for 64 `regions` when $\lambda = 0.993$ (i.e., FP = 86%, TP = 0%), for 128 `regions` when $\lambda = 0.994$ (i.e., FP = 85%, TP = 0%), and for 256 `regions` when $\lambda = 0.993$ (i.e., FP = 86%, TP = 0%). Although, an interesting observation results from the fact that the TP rates exhibited a slower decay compared to the FP rates increasing the values of threshold λ across the number of `regions`. More precisely an interesting fact appears when observing that increasing the number of `regions` a more sharp decay is exhibited in the TP rates for lower values of λ, while the TP rates are decaying somehow later for greater values of λ resulting hence to greater differences among the TP and FP rates leading further to inspect promising aspects of that approach.

As we can observe from the exhibited results depicted in Fig. 9, the increase of the `regions` results to the computation of similarity among parts of the graph that include their previous modification (CMTG graphs), which leads to the inference that the average similarity exhibited is on the other hand biased by differences exhibited in previous `regions` and are included to all the next segmentations, but on the other hand however, the achieved detection ability seems to perform better across all the deployed similarity metrics regardless of the type of graph characteristics under investigation. This observation leads us to a further inference, that construction of the CMTG performs adequately well against the mutations of malicious samples across the whole extent of their graphs, capturing system-call invocation that have been performed in different times due the evolution of the GrG by its corresponding ScDG graph.

5.3. Evaluating the classification ability

In order to perform the classification procedure, we primarily conducted an investigation over our data-set, regarding the correlation between the malware families. In particular, an interesting observation results from the fact that samples that have been pre-classified to malware families with “related” names tend to have indeed an increased similarity between them. In Fig. 10 we have depicted the relation between families with at least one common block at their names definitions. In order to perform a prior evaluation of our model we distinguished three type of classification, namely: Exact, Direct, and Partial Matching. Next we present the results exhibited through two series of experiments utilizing both the discrete modification temporal graphs and the cumulative modification temporal graphs in order to

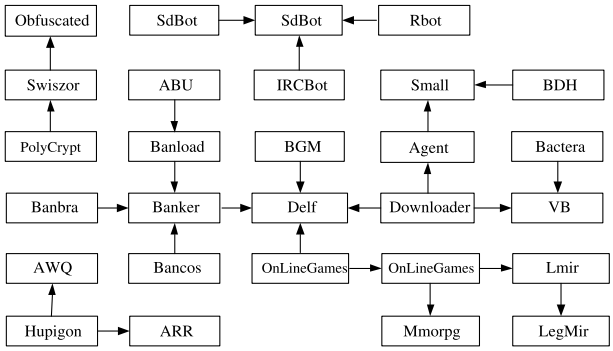


Fig. 10. Malware families connected by their names.

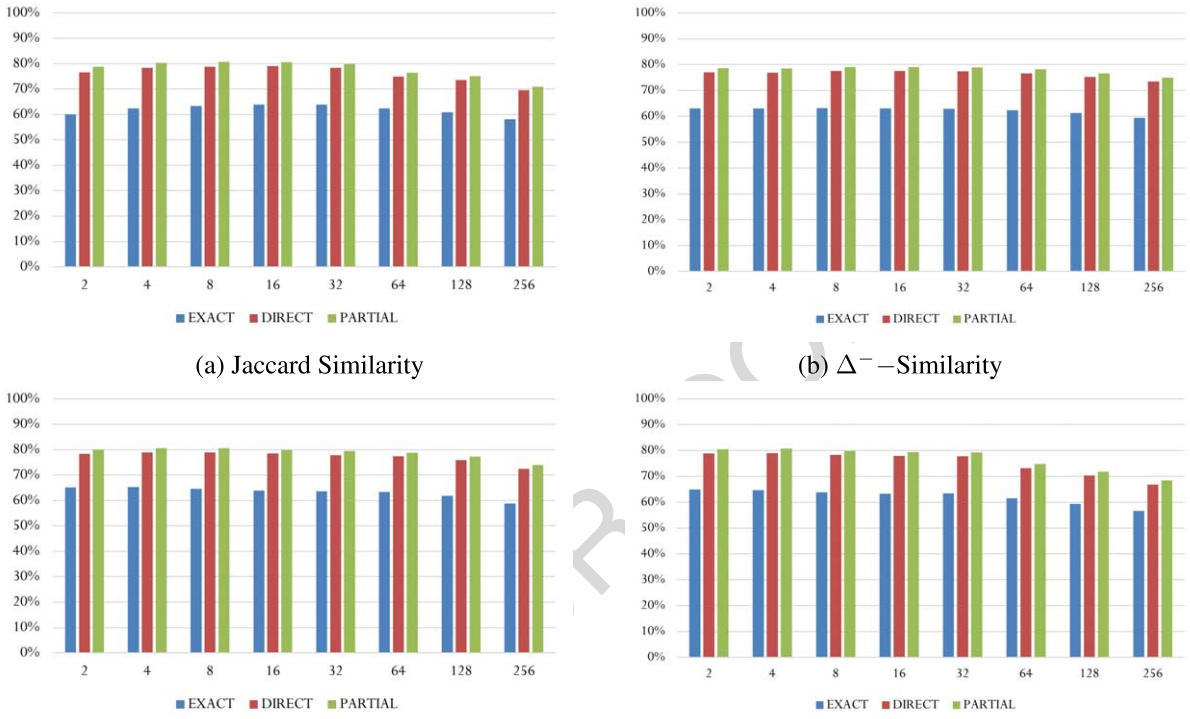
represent the evolution of a GrG graphs over time, and into a further extent leverage the characteristics of these type of temporal graphs in malware classification. The procedure followed for the indexing of software samples that have been previously detected as malicious to known malware families, is the one described in Section 4.3, see Fig. 5. Each of these classification types refers to how we attest the name of the family that our model classified our sample, and are define as follows:

- **Exact Matching** The Exact Matching attests both parts of the names between the name of the family that our system assigned the sample to and the name of the family that exists on the sample's hidden family tag, demanding to be fully equivalent at both parts of their names. More precisely, for a given test sample τ from family X_τ, Y_τ, if it has been classified to the family that s belong to, let X_s, Y_s, then the conditional rule $((X_\tau \equiv X_s) \wedge (Y_\tau \equiv Y_s)) \rightarrow T$ will be valid.
- **Directed Matching** The Directed Matching attests both parts of the names between the name of the family that our system assigned the sample to and the name of the family that exists on the sample's hidden family tag, demanding at least one of the parts of the names to be matched to the exact position (i.e., first or second part of the test sample's family name, or even both of them) in order for a correct classification to be counted. More precisely, for a given test sample τ from family X_τ, Y_τ, if it has been classified to the family that s belong to, let X_s, Y_s, then the conditional rule $((X_\tau \equiv X_s) \vee (Y_\tau \equiv Y_s)) \rightarrow T$ will be valid.
- **Partial Matching** The Partial Matching attests both parts of the names between the name of the family that our system assigned the sample to and the name of the family that exists on the sample's hidden family tag, demanding at least one of the parts of the names to be matched to any part of the name of the family of the test sample (i.e., at least one part of the sample's family name to be matched to at least one part of the test family name) in order for a correct classification to be counted. More precisely, for a given test sample τ from family X_τ, Y_τ, if it has been classified to the family that s belong to, let X_s, Y_s, then the conditional rule $(X_\tau \equiv X_s) \vee (X_\tau \equiv Y_s) \vee (Y_\tau \equiv Y_s) \vee (Y_\tau \equiv X_s) \rightarrow T$ will be valid.

In Table 2, we present an example of our classification evaluation procedure through a simple example explaining how these classification accuracy metrics work. Let as assume that we have a sample from family "Bancos, Banker" that has been detected as malware and we classify it into a malware family that is presented in the first column. So, in columns 2, 3 and 4 we can observe what would be the result (1 for correct or 0 for wrong classification) according to if we demand exact, direct, and partial matching.

Table 2
Classification example utilizing the exact, the direct, and the partial matching

Indexed in:	Exact	Direct	Partial
Banker, Delf	0	0	1
Banbra, Banker	0	1	1
Banker, Banker	0	1	1
Bancos, Banker	1	1	1



(a) Jaccard Similarity (b) Δ^-−Similarity

(c) Δ^+−Similarity (d) Cosine Similarity

Fig. 11. From (a) to (d) we depict the indexing of detected malware samples to known malware families for various `regions` using the Jaccard, the Δ^-, the Δ^+, and the Cosine Similarity metrics on DMTG graphs.

5.3.1. Classifying detected samples utilizing DMTG

In the first series of experiments we utilize the discrete modification temporal graphs to represent the evolution of a GrG graphs over time, and the deployment of similarity metrics that investigate the relational, the qualitative, and the quantitative, characteristics, in order to measure the classification ability of our proposed model. Throughout this series of experiments we investigate the classification rates exhibited by the our model over different types of correct classifications, (i.e., Exact, Direct, and Partial Matching) as described above in order to prove the potentials of our model in the indexing of software samples.

In Fig. 11 we present the classification results exhibited through a series of experiments utilizing the discrete modification temporal graphs alongside the Jaccard Similarity, the Δ^--Similarity and the Δ^+-Similarity metrics, and the Cosine Similarity metric investigating the relational, the qualitative and the quantitative characteristics respectively, note that the exhibited results are depicted w.r.t. the three types

of classification defined above. As we can observe from Fig. 11 despite the type of the underlying characteristics of the temporal graph, the Exact Matching type of classification seems to perform moderately across all the partition sizes of temporal graphs (i.e., `regions`).

On the other hand, as we can observe from Fig. 11(a) the utilization of relational characteristics performs adequately for the types of Direct and Partial Matching especially for lower values of `regions` seems presents a common behaviour on the two types of classification across the `regions`, exhibiting adequately high levels for 8–32 `regions`. The Δ-Similarity metric, in both of its implementations utilizing either the in/out-degree and in/out-weights, as shown in Fig. 11(b),(c) seems that for the cases of Direct and Partial Matching perform adequately well across the greater extent of `regions`, exhibiting a slight decrease only for numbers of `regions` over 64. Finally, for the same ranges of `regions`, i.e., 8 to 32 the Cosine Similarity seems to perform similarly to Jaccard and Δ-Similarity, however for higher numbers of `regions` the effect of the quantitative characteristics seems that does not contribute positively in the indexing of malicious samples regarding a more fine-grained and more detailed comparison.

5.3.2. Classifying detected samples utilizing CMTG

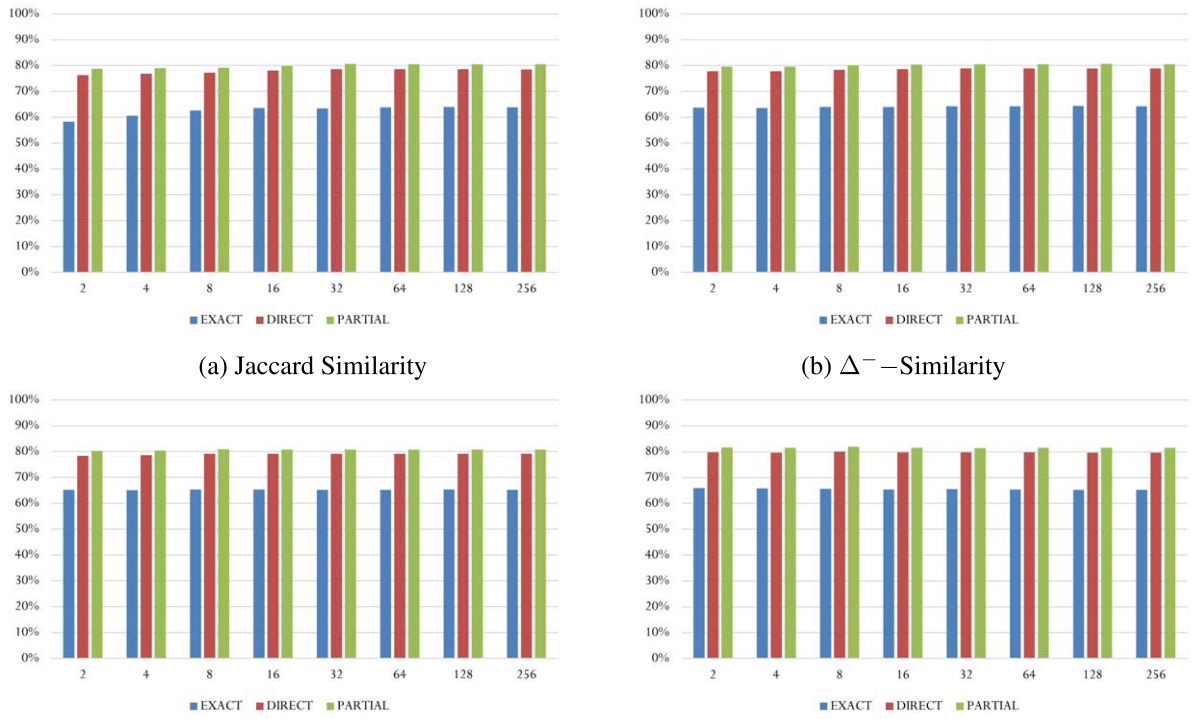
In the second series of experiments, similarly we utilize the cumulative modification temporal graphs to represent the evolution of a GrG graphs over time, and the deployment of similarity metrics that investigate the relational, the qualitative, and the quantitative, characteristics, in order to measure the classification ability of our proposed model. Correspondingly, our main goal is to investigate the classification rates exhibited by the our model over different types of correct classifications, (i.e., Exact, Direct, and Partial Matching) as described above in order to prove the potentials of our model in the indexing of software samples.

The classification results exhibited through the second series of experiments, utilizing the cumulative modification temporal graphs alongside the Jaccard Similarity, the Δ^- and the Δ^+-Similarity metrics, and the Cosine Similarity metric investigating the relational, the qualitative and the quantitative characteristics respectively, are presented in Fig. 12. The exhibited results concern the three types of classification, similarly to the previous experiment series for the evaluation of the classification ability utilizing DMTG graphs. A major insight that results through the observation of Fig. 12, is the fact that the Exact Matching type of classification seems to perform moderately across all the partition sizes of temporal graphs (i.e., `regions`), regardless the underlying characteristics of the temporal graph, as also mentioned in the previous experiments.

On the other hand, as we can observe from Fig. 12 (a)–(d), an interesting observation comes to light, when it results that the increase on the number of `regions` results to an increase of the classification ability across all the deployed similarity metrics. Additionally, as we can see in Fig. 12 (d) there is exhibited a slightly increased classification ability leading to the deduction that the application of quantitative characteristics for the computation of similarity between CMTG graphs performs better than the other characteristics in terms of classification ability.

6. Conclusion

In this work we designed and developed a graph-based framework for malware detection and classification. The core component of our work are the temporal graphs that depict the structural evolution of a graph through time. The object we operate on, are the Group Relation Graphs (GrG graphs), that are constructed after grouping specific disjoint vertices and the corresponding edges by the system-calls



(a) Jaccard Similarity

(b) Δ^-−Similarity

(c) Δ^+−Similarity

(d) Cosine Similarity

Fig. 12. From (a) to (d) we depict the indexing of detected malware samples to known malware families for various `regions` using the Jaccard, the Δ^-, the Δ^+, and the Cosine Similarity metrics on CMTG graphs.

invoked during the execution time of a software sample. In our approach we distinguish two types of temporal graphs, namely, the discrete modification temporal graphs and the cumulative modification temporal graphs. In the first type of temporal graphs, i.e., the discrete modification temporal graphs, the structural modifications of a GrG graphs through time (i.e., `regions`) are registered containing only the specific modification performed in a specific period, while, in the second type of temporal graphs, i.e., the cumulative modification temporal graphs, the structural modifications of a GrG graphs through the `regions` are registered accumulating all the modification performed by the initial state of a GrG graph until a specific period. Finally, the detection and classification results exhibited over a series of experiments of our approach provided us with insights about the potentials of our model regarding the effect of mutation of malicious software in the detection and classification procedures.

6.1. Discussion

Through the evaluation of our model we performed two distinct series of experiments regarding the utilization of the two types of temporal graphs, namely, the discrete modification temporal graphs and the cumulative modification temporal graphs, regarding their potentials in detecting and classifying malicious samples. Throughout the experimental study for the evaluation of our proposed framework, we investigated three types of graph characteristics that could be utilized by the corresponding similarity metrics in order to measure the similarity between a test and any known malicious sample. Namely, commonalities exhibited by the relational, the qualitative and the quantitative characteristics between

temporal graphs are measure by the Jaccard Similarity, the Δ-Similarity, and the Cosine Similarity, respectively, in order to deduce whether a software sample under consideration is malicious or benign, and to a further extent to classify it to a family of known malware samples.

The experimental results exhibited over the evaluation of detection and classification ability of our model utilizing the discrete modification of temporal graphs showed that the performance of our framework on detection and classifying malicious samples performs adequately well. Observing the corresponding results it is figured out that the framework utilizes efficiently the commonalities exhibited over the various types of graph characteristics where as we can see a further increase on the number of `regions` results to a more fine grained depiction of the temporal evolution of the corresponding Group Relation Graph (GrG) that may be affected by the mutation of a sample, making it more vulnerable to be easily altered, decreasing hence the probability of high similarity when computing it in very specific part of a graph.

On the other hand, the experimental results exhibited over the evaluation of detection and classification ability of our model utilizing the cumulative modification of temporal graphs showed that the performance of our framework on detection and classifying malicious samples performs in very high rates exhibiting high detection rates followed by very low false positive rates, alongside with high classification rates. Throughout the corresponding results it is clear to see that the commonalities over all the types of graph-characteristics perform equally well regarding the detection and classification potentials exhibited by our proposed framework. The experimental results show in contrast that the number of `regions` has less decrease on the detection and classification abilities of our model since the increase of the `regions` results to the computation of similarity among parts of the graph that include their previous modification, biasing by the differences exhibited in previous `regions` and are included to all the next segmentations the average similarity exhibited.

As we depict in Fig. 13, there is a straight distinction among the detection results exhibited over the utilization of discrete modification and the cumulative modification temporal graphs alongside the underlying graph-characteristics the commonalities of them computed by the corresponding similarity metrics. In Fig. 13, we present the detection results exhibited over the deducted series of experiments and evaluate the potentials of each case by measuring the maximum difference presented among the TP and the FP rates across a number of `regions`. In the corresponding plot there are presented the detection results of the Jaccard Similarity, the Δ^{-}- and Δ^{+}-Similarity, and Cosine Similarity metrics when deployed over discrete modification and cumulative modification temporal graphs.

Observing these results it is easy to see that there is a set combinations of similarity metrics and underlying temporal graphs that over the point of 32 `regions` the maximization of the difference of TP and FP rates that depict their performance seems to decrease to very low levels. This fact comes to attest our initial conjecture that the fine grained and very detailed information that discrete modification temporal graphs are able to capture is vulnerable to mutations located into smaller segments of the graph and hence the different types of graph-characteristics are unable to perform as effectively as in case of smaller `regions`. The exhibited performance in each case (i.e., detection/classification) exhibits a slight decrease due to the utilization of temporal graphs as by their construction they are intended to capture specific information discretely over a sequence of periods. That is, when the partitioning on the temporal evolution of a graph is performed in a more fine-grained approach, i.e., the more the `regions` a graph is partitioned into the smaller is the extent of each graph instance, then the information that depicts a specific behavioral pattern of a malicious sample may be distributed into more sequential graph instances (i.e., parts of the graph constructed over each `region`).

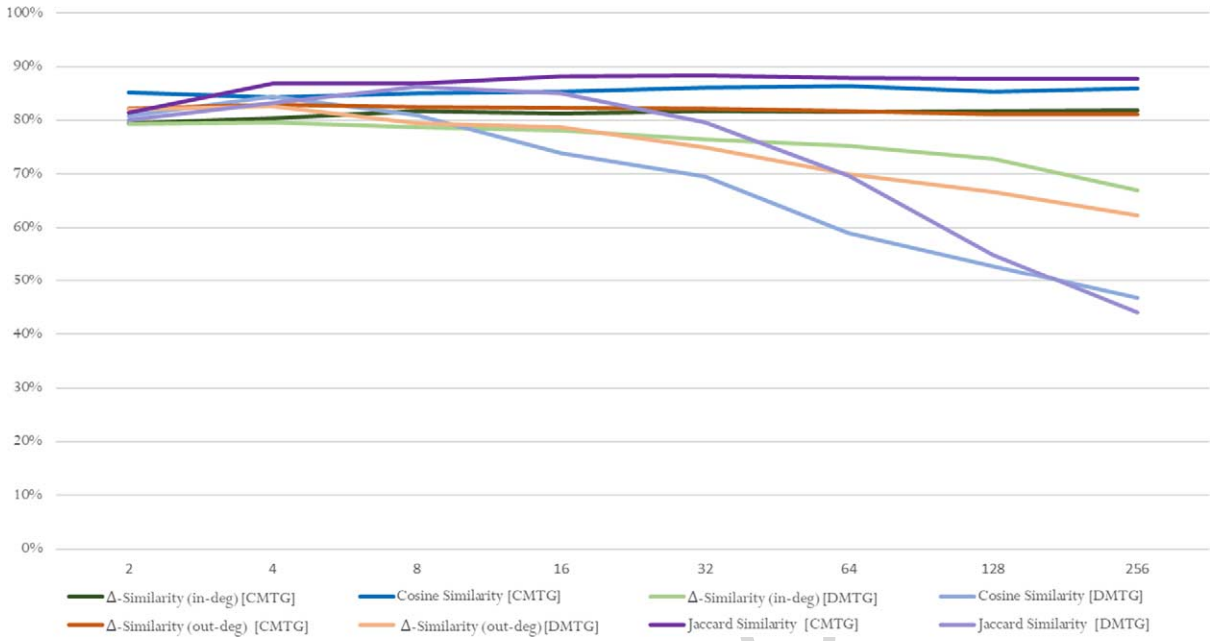


Fig. 13. Comparison of detection results utilizing different similarity metrics over DMTG and CMTG graphs.

On the other hand, the similarity metrics deployed on top of the cumulative modification temporal graphs, even for greater numbers of `regions` seem to perform similarly exhibiting high TP rates with low FP rates regardless the underlying graph-characteristics utilized for the computation of the similarity metric, with exceptional case the utilization of relational characteristics through Jaccard Similarity that in the whole range of `regions` seems to outperform the Δ^-, Δ^+, and Cosine Similarity metrics.

In order to attest the potentials of the detection ability of our proposed detection model we measure its effectiveness on detection malicious samples consigning four measurements, namely the precision, the recall, its accuracy and the F-measure, which their computation is listed below:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \qquad \text{F-Measure} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Throughout our evaluation experiments regarding the detection potentials of our proposed model, for the case of the utilization of Discrete modification Temporal Graphs (DMTG) to represent the structural evolution of a Group Relation Graphs (GrG) graph over time, the corresponding four factors that prove the detection potentials of our model are maximized for the case of the deployment of Jaccard similarity metric partitioning the graph into 4 `regions` with $\lambda = 0.72$, where the Precision is 97.7%, the Recall is 97.0%, the Accuracy is 95.0%, and F-Measure is 97.3%. On the other hand, or the case of the utilization of Cumulative modification Temporal Graphs (CMTG) to represent the structural evolution of a GrG graph over time, the corresponding four factors that prove the detection potentials of our model are maximized for the case of the deployment of Jaccard similarity metric partitioning the graph into 64 or 256 `regions` with $\lambda = 0.88$ and $\lambda = 0.87$, respectively, where the Precision is 97.9%, the Recall is 97.0%, the Accuracy is 95.3%, and F-Measure is 97.5%.

Table 3
Detection rates (%) exhibited on the factors of precision, recall, accuracy, and F-measure, by relative works

In:	Method	Precision	Recall	Accuracy	F-Measure
[51]	API-Call Mining	93.9	–	94	94
[29]	Integrated Feature Set of PE	95.5	94.4	94.9	94.9
[14]	Audio Signal Processing	91.5	93.1	92.2	92.2
[45]	Group Relation Graph	99.1	94	93.5	96.5
[10]	Family Behavior Graph	–	–	96.4	–
[15]	API-Calls Usage Frequency	90.2	–	87.1	86.6
[11]	Fuzzy and Fast Fuzzy Pattern Tree	94.3	89.7	96.4	89
[61]	Deep Learning of Behavior Graphs	98.6	99.2	–	98.9
[26]	API Call Sequence Alignment	95	93.6	94.9	94.3
[49]	Structural Specification of PE – RF	95.7	95.4	95.5	95.5
[56]	API Call Sequences	92.6	–	90.8	90.6
[30]	System-call Dependency Sequences	97.6	–	98.2	98.2
This work	Discrete Modification Temporal Graphs	97.7	97	95	97.3
This work	Cumulative Modification Temporal Graphs	97.9	97	95.3	97.5

In Table 3 we cite the results from relative approaches concerning the reported measurements on the detection rates (%) on the exhibited Precision, Recall, Accuracy and F-Measure, where they are reported. The proposed approach, when compared to the baseline where the GrG graphs are deployed without the representation of their structural evolution during time [45] it is observable that only in the Precision measurement the proposed approach exhibits results less than the ones achieved by the baseline, where in the other three factors, i.e., Recall, Accuracy, and F-Measure, the proposed approach utilizing either the DMTG or the CMTG outperforms the approach where the Temporal Graphs are not deployed. Moreover, for the case of the utilization of CMTG for $\lambda = 0.909$ and partitioning the graph into 64 or 256 `regions`, and for $\lambda = 0.91$ and partitioning the graph into 128 `regions` it is achieved a 93% TP-rate over a 5% FP-rate, that compared to the baseline (94% TP-rate over a 13% FP-rate) it exhibits a higher true-negative rate i.e., 95% over an 87% which indeed verifies our initial intuition for the improvements achieved by the utilization of Temporal graphs. On the other hand, regarding the Precision measurement, the results exhibited by our work achieves from the highest rates, while for the Recall measurements the proposed approach still exhibits high results even if in some works there is not reported the corresponding measurement. Additionally, the Accuracy and the F-Measure achieved by the proposed approach in both the utilization of DMTG and CMTG exhibits also high rates compared to the previous approach proving the potentials of our model in distinguishing malicious from benign samples.

Regarding the comparison of the results exhibited by our proposed method it is observable that the potentials depicted by the achieved detection rates show that our technique is comparable to the ones proposed during the latest years, i.e., the utilization of Fuzzy and Fast Fuzzy Pattern Tree [11], API Call Sequence Alignment [26], API Call Sequences [56], and System-call Dependency Sequences [30], where our proposed approach exhibits better results against some of them, with the System-call Dependency Sequence method proposed in [30] exhibiting the maximum values regarding the Accuracy, F-Measure, and Precision–Recall measurements.

Similarly, throughout our evaluation experiments regarding the classification potentials of our proposed model, for the case of the utilization of Discrete modification Temporal Graphs (DMTG) to represent the structural evolution of a Group Relation Graphs (GrG) graph over time we verified the classifica-

Table 4
Classification accuracy (%) exhibited by relative works

In:	Method	Accuracy
[50]	Discriminative behaviors	88
[22]	K-nearest neighbors Function-call Graphs	69.9
[64]	One-class SVM APIs, strings and basic blocks	78
[7]	Performance monitor, System-calls and System-call sequences	66.8
[37]	SVM N-gram feature of the network artifacts	80
[47]	Runtime artifacts, IDS signatures, and important API calls	92.5
[45]	Group Relation Graphs	83.42
This work	Discrete Modification Temporal Graphs	81
This work	Cumulative Modification Temporal Graphs	82

tion ability of our model comparing it with the ones achieved by relative approaches. For the case of the deployment of Cossine similarity metric partitioning the evolution of the GrG graph into 4 `regions`, utilizing the Discrete Modification Temporal Graphs our proposed model achieves an 81% classification accuracy, while for the case where the GrG graph is partitioned into 8 `regions` and the Cumulative Modification Temporal Graphs are utilized, the proposed model achieves an 83% classification accuracy. Respectively, in Table 4 we cite the results from relative approaches concerning the reported measurements on the classification accuracy rates (%). The proposed approach, when compared to the baseline [45] where the GrG graphs are deployed without the representation of their structural evolution during time [45] it is observable that it performs similarly to the initial approach where the Temporal Graphs are not utilized, while on the other hand the proposed approach of the utilization of Temporal Graphs in the depiction of the structural evolution of the GrG graphs seems to perform adequately well regarding the results exhibited by other graph-based techniques or API/System-call based approaches, proving thus the potentials of our proposed model in indexing malicious samples to known malware families.

6.2. *Potentials and limitations*

Several modeling alternates have been arise during the theoretical construction of our graph-based proposed model regarding the temporal evolution of behavioral graphs that represent software samples, regarding their structural modification during time. Our approaches that we discuss briefly next, mostly concern the representation of the structural modifications on the GrG graphs during time, and how they could also be represented with other structures that do not cooperate graphs, and consequently deserve the application of different manipulation methods.

In the first alternate approach, we could denote the structural evolution of a given by plotting by a discrete distribution of the addition of edges over the graph on specific time buckets and create patterns that could be utilized in order to perform pattern-matching over the plot of any given pair of samples. These plots should be construct for the temporal evolution of each corresponding edge pair of two given graphs in order for the patterns to be comparable. On the other hand, in the second approach of our model, we need to simulate the structural modification of a given graph during time. Similarly to our approach, rather than constructing several graph instances equal to the number of the defined `regions` and structurally relevant to the applied method regarding the discrete or cumulative modification approach, we could also represent these structural modification over the time for each edge of GrG graph. More precisely, we could define a binary sequence for each edge, where 0 denotes absence and 1 denote addition of this edge on the overall graph, and the length of the sequence equals the size of the ScDG. Then,

various alignment algorithms could be adopted in order to retrieve similarity patterns among any pair of such sequences, that represent corresponding edges on the graphs of the test and the known malicious samples.

Regarding the limitations of our model, the main issue encountered regarding the implementation design concerns the spatial complexity of our approach. More precisely, defining a fine-grained or a coarse-grained quantization of time would affect to a great extent the space required to store the corresponding Temporal Graph instances. As easily someone can understand, an implementation of our proposed model on a fine-grained time quantization scheme, would be more precise against a more coarse-grained once. Additionally, further tuning issues arise over the trade-off between the precision on temporal structural modifications and the construction of more distinguishing patterns. However, more sophisticated approaches, such an implementation that utilizes the maximum length of a binary tree in order to bound the quantization would lead to a more stable, rational, effective and efficient approach.

6.3. Further research

In the context of extending our work we set our future research aims over the investigation of comparison of temporal graphs, and mainly focusing on the `regions` defined by the discrete modification temporal graphs (DMTG), regarding the measurement of similarity between `regions` that are in different temporal position. The main aspect of our future research, approached the computation of similarity between the `regions` of DMTG graphs, by overcoming the drawback caused by the shift of specific patterns exhibited in the corresponding Group Relation Graphs due to malware mutations. Our primarily concern is the development of an algorithmic approach in order to manage the computational and storage complexity that will be caused by the deployment of the extension discussed above. Finally, an interesting perspective would be to investigate additional types of characteristics exhibited across the structure of cumulative modification temporal graphs (CMTG) alongside the utilization of other similarity metrics, as also further information investigation the relations between malware families.

Acknowledgment

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning 2014-2020" in the context of the project "Malicious Software Detection and Classification utilizing Temporal–Graphs of Discrete and Cumulative Structural Evolution" (MIS 5047642).

References

[1] B. Alsulami, A. Srinivasan, H. Dong and S. Mancoridis, Lightweight behavioral malware detection for windows platforms, in: *International Conference on Malicious and Unwanted Software (MALWARE)*, IEEE, 2017, pp. 75–81. doi:10.1109/MALWARE.2017.8323959.

[2] L. Aneja and S. Babbar, Research trends in malware detection on Android devices, in: *International Conference on Recent Developments in Science, Engineering and Technology*, Springer, 2017, pp. 629–642.

[3] D. Babic, D. Reynaud and D. Song, Malware analysis with tree automata inference, in: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, 2011, pp. 116–131.
[4] S. Basole, F. Di Troia and M. Stamp, Multifamily malware models, *Journal of Computer Virology and Hacking Techniques* **1**(14) (2020).
[5] M.L. Bernardi, M. Cimitile, D. Distante, F. Martinelli and F. Mercaldo, Dynamic malware detection and phylogeny analysis using process mining, *International Journal of Information Security* **1**(28) (2018).
[6] A. Bulazel and B. Yener, A survey on automated dynamic malware analysis evasion and counter-evasion: PC, mobile, and web, in: *Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium*, ACM, 2017, pp. 1–21.
[7] R. Canzanese, M. Kam and S. Mancoridis, Toward an automatic, online behavioral malware classification system, in: *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, IEEE, 2013, pp. 111–120.
[8] A. Damodaran, F. Di Troia, C.A. Visaggio, T.H. Austin and M. Stamp, A comparison of static, dynamic, and hybrid analysis for malware detection, *Journal of Computer Virology and Hacking Techniques* **13**(1) (2017), 1–12. doi:10.1007/s11416-015-0261-z.
[9] B. David, E. Filiol and K. Gallienne, Structural analysis of binary executable headers for malware detection optimization, *Journal of Computer Virology and Hacking Techniques* **13**(2) (2017), 87–93. doi:10.1007/s11416-016-0274-2.
[10] Y. Ding, X. Xia, S. Chen and Y. Li, A malware detection method based on family behavior graph, in: *Computers and Security*, Vol. 73, Elsevier, 2018, pp. 73–86.
[11] E.M. Dovom, A. Azmoodeh, A. Dehghantanha, D.E. Newton, R.M. Parizi and H. Karimipour, Fuzzy pattern tree for edge malware detection and categorization in IoT, *Journal of Systems Architecture* **97** (2019), 1–7. doi:10.1016/j.sysarc.2019.01.017.
[12] M. Elingiusti, L. Aniello, L. Querzoni and R. Baldoni, Malware detection: A survey and taxonomy of current techniques, *Cyber Threat Intelligence* (2018), 169–191. doi:10.1007/978-3-319-73951-9_9.
[13] R. Eskandari, M. Shajari and M.M. Ghahfarokhi, ERES: An extended regular expression signature for polymorphic worm detection, *Journal of Computer Virology and Hacking Techniques* **15**(3) (2019), 177–194. doi:10.1007/s11416-019-00330-1.
[14] M. Farrokhmanesh and A. Hamzeh, A novel method for malware detection using audio signal processing techniques, in: *2016 Artificial Intelligence and Robotics (IRANOPEN)*, IEEE, 2016, pp. 85–91. doi:10.1109/RIOS.2016.7529495.
[15] V. Garg and R.K. Yadav, Malware detection based on API calls frequency, in: *2019 4th International Conference on Information Systems and Computer Networks*, (ISCON), IEEE, 2019, pp. 400–404.
[16] V. Ghanaei, C.S. Iliopoulos and R.E. Overill, Statistical approach towards malware classification and detection, in: *2016 SAI Computing Conference (SAI)*, IEEE, 2016, pp. 1093–1099. doi:10.1109/SAI.2016.7556114.
[17] L.S. Grini, A. Shalaginov and K. Franke, Study of soft computing methods for large-scale multinomial malware types and families detection, in: *Recent Developments and the New Direction in Soft-Computing Foundations and Applications*, Springer, 2018, pp. 337–350. doi:10.1007/978-3-319-75408-6_26.
[18] K. Grosse, N. Papernot, P. Manoharan, M. Backes and P. McDaniel, Adversarial examples for malware detection, in: *European Symposium on Research in Computer Security*, Springer, Cham, 2017, pp. 62–79.
[19] H. Hashemi, A. Azmoodeh, A. Hamzeh and S. Hashemi, Graph embedding as a new approach for unknown malware detection, *Journal of Computer Virology and Hacking Techniques* **13**(3) (2017), 153–166. doi:10.1007/s11416-016-0278-y.
[20] H. Hashemi and A. Hamzeh, Visual malware detection using local malicious pattern, *Journal of Computer Virology and Hacking Techniques* **15**(1) (2019), 1–14. doi:10.1007/s11416-018-0314-1.
[21] M. Hassen and P.K. Chan, Scalable function call graph-based malware classification, in: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ACM, 2017, pp. 239–248. doi:10.1145/3029806.3029824.
[22] X. Hu, T. Chiueh and K.G. Shin, Large-scale malware indexing using function-call graphs, in: *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*, 2009, pp. 611–620.
[23] R. Islam, R. Tian, L. Batten and S. Versteeg, Classification of malware based on string and function feature selection, in: *Proceedings of the Cybercrime and Trustworthy Computing and Workshop (CTC'10)*, 2010, pp. 9–17.
[24] G. Jacob, H. Debar and E. Filiol, Behavioral detection of malware: From a survey towards an established taxonomy, *Journal in computer Virology* **4**(3) (2008), 251–266. doi:10.1007/s11416-008-0086-0.
[25] T.S. John, T. Thomas, Emmanuel and S. Graph, Convolutional networks for Android malware detection with system call graphs, in: *ISEA Conference on Security and Privacy (ISEA-ISAP)*, IEEE, 2020, pp. 162–170. doi:10.1109/ISEA-ISAP49340.2020.235015.
[26] H. Kim, J. Kim, Y. Kim, I. Kim, K.J. Kim and H. Kim, Improvement of malware detection and classification using API call sequence alignment and visualization, *Cluster Computing* **22**(1) (2019), 921–929. doi:10.1007/s10586-017-1110-2.
[27] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras and C. Eckert, Empowering convolutional networks for malware classification and analysis, in: *Neural Networks (IJCNN), 2017 International Joint Conference on*, IEEE, 2017, pp. 3838–3845.
[28] A.V. Kozachok and V.I. Kozachok, Construction and evaluation of the new heuristic malware detection mechanism based on executable files static analysis, *Journal of Computer Virology and Hacking Techniques* **14**(3) (2018), 225–231. doi:10.1007/s11416-017-0309-3.

[29] A. Kumar, K.S. Kuppusamy and G. Aghila, A learning model to detect maliciousness of portable executable using integrated feature set, *Journal of King Saud University-Computer and Information Sciences* **31**(2) (2019), 252–265. doi:10.1016/j.jksuci.2017.01.003.
[30] A.M. Lajevardi, S. Parsa and M.J. Amiri, Markhor: malware detection using fuzzy similarity of system call dependency sequences, *Journal of Computer Virology and Hacking Techniques* **1**(10) (2021).
[31] C.H. Lin, H.K. Pao and J.W. Liao, Efficient dynamic malware analysis using virtual time control mechanics, *Computers and Security* **73** (2018), 359–373. doi:10.1016/j.cose.2017.11.010.
[32] J. Liu, P. Dai Xie, M.Z. Liu and Y.J. Wang, Having an insight into malware phylogeny: Building persistent phylogeny tree of families, *IEICE TRANSACTIONS on Information and Systems* **101**(4) (2018), 1199–1202. doi:10.1587/transinf.2017EDL8172.
[33] J. Liu, Y. Wang, P. Dai Xie and Y.J. Wang, Inferring phylogenetic network of malware families based on splits graph, *IEICE TRANSACTIONS on Information and Systems* **100**(6) (2017), 1368–1371. doi:10.1587/transinf.2016EDL8230.
[34] A. Makandar and A. Patrot, Trojan malware image pattern classification, in: *Proceedings of International Conference on Cognition and Recognition*, Springer, Singapore, 2018, pp. 253–262. doi:10.1007/978-981-10-5146-3_24.
[35] K. Mathur and S. Hiranwal, A survey on techniques in detection and analyzing malware executables, *Journal of Advanced Research in Computer Science and Software Engineering* **3** (2013), 22–428.
[36] J. Ming, D. Xu and D. Wu, MalwareHunt: Semantics-based malware diffing speedup by normalized basic block memoization, *Journal of Computer Virology and Hacking Techniques* **13**(3) (2017), 167–178. doi:10.1007/s11416-016-0279-x.
[37] A. Mohaisen, A.G. West, A. Mankin and O. Alrawi, Chatter: Classifying malware families using system event ordering, in: *2014 IEEE Conference on Communications and Network, Security*, IEEE, 2014, pp. 283–291.
[38] J. Moubarak, M. Chamoun and E. Filiol, Comparative study of recent MEA malware phylogeny, in: *Computer and Communication Systems (ICCCS), 2017 2nd International Conference on*, IEEE, 2017, pp. 16–20.
[39] A. Mpanti, S.D. Nikolopoulos and I. Polenakis, A graph-based model for malicious software detection exploiting domination relations between system-call groups, in: *Proceedings of the 19th Int'l Conference on Computer Systems and Technologies*, ACM, 2018.
[40] S.D. Mukesh, J.A. Raval and H. Upadhyay, Real-time framework for malware detection using machine learning technique, in: *International Conference on Information and Communication Technology for Intelligent Systems*, Springer, 2017, pp. 173–182.
[41] U. Narra, F. Di Troia, V.A. Corrado, T.H. Austin and M. Stamp, Clustering versus SVM for malware detection, *Journal of Computer Virology and Hacking Techniques* **12**(4) (2016), 213–224. doi:10.1007/s11416-015-0253-z.
[42] L. Nataraj, S. Karthikeyan, G. Jacob and B.S. Manjunath, Malware images: Visualization and automatic classification, in: *Proceedings of the 8th Int'l Symposium on Visualization for Cyber Security (VizSec'11)*, 2011, pp. 4–11.
[43] L. Nataraj, S. Karthikeyan, G. Jacob and B.S. Manjunath, A comparative assessment of malware classification using binary texture analysis and dynamic analysis, in: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 21–30. doi:10.1145/2046684.2046689.
[44] S.D. Nikolopoulos and I. Polenakis, A graph-based model for malicious code detection exploiting dependencies of system-call groups, in: *Proceedings of the 16th International Conference on Computer Systems and Technologies*, 2015, pp. 228–235. doi:10.1145/2812428.2812432.
[45] S.D. Nikolopoulos and I. Polenakis, A graph-based model for malware detection and classification using system-call groups, *Journal of Computer Virology and Hacking Techniques* **13**(1) (2017), 29–46. doi:10.1007/s11416-016-0267-1.
[46] Y. Park, D. Reeves, V. Mulukutla and B. Sundaravel, Fast malware classification by automated behavioral graph matching, in: *Proceedings of the 6th ACM Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW'10)*, 2010, pp. 45–49.
[47] A. Pektaş and T. Acarman, Classification of malware families based on runtime behaviors, *Journal of information security and applications* **37** (2017), 91–100. doi:10.1016/j.jisa.2017.10.005.
[48] B.B. Rad, M. Maslin and I. Suhaimi, Camouflage in malware: From encryption to metamorphism, *Journal of Computer Science and Network Security* **12** (2012), 74–83.
[49] T. Rezaei and A. Hamze, An efficient approach for malware detection using PE header specifications, in: *2020 6th International Conference on Web Research (ICWR)*, IEEE, 2020, pp. 234–239. doi:10.1109/ICWR49608.2020.9122312.
[50] K. Rieck, H. Thorsten, W. Carsten, D. Patrick and P. Laskov, Learning and classification of malware behavior, in: *Proceedings of the 5th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA'08)*, 2008, pp. 108–125. doi:10.1007/978-3-540-70542-0_6.
[51] 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*, 2010, pp. 1020–1025.
[52] A. Saracino, D. Sgandurra, G. Dini and F. Martinelli, Madam: Effective and efficient behavior-based Android malware detection and prevention, *IEEE Transactions on Dependable and Secure Computing* **15**(1) (2018), 83–97. doi:10.1109/TDSC.2016.2536605.

[53] G. Severi, T. Leek and B. Dolan-Gavitt, Malrec: Compact full-trace malware recording for retrospective deep analysis, in: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2018, pp. 3–23. doi:10.1007/978-3-319-93411-2_1.
[54] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*, No Starch Press, 2012.
[55] A. Souri and R. Hosseini, A state-of-the-art survey of malware detection approaches using data mining techniques, *Human-centric Computing and Information Sciences* **8**(1) (2018), 3. doi:10.1186/s13673-018-0125-x.
[56] J. Suaboot, Z. Tari, A. Mahmood, A.Y. Zomaya and W. Li, Sub-curve HMM: A malware detection approach based on partial analysis of API call sequences, *Computers & Security* **92** (2020), 101773. doi:10.1016/j.cose.2020.101773.
[57] G. Sun and Q. Qian, Deep learning and visualization for identifying malware families, *IEEE Transactions on Dependable and Secure Computing* (2018).
[58] Y.S. Sun, C.C. Chen, S.W. Hsiao and M.C. Chen, ANTSdroid: Automatic malware family behaviour generation and analysis for Android apps, in: *Australasian Conference on Information Security and Privacy*, Springer, 2018, pp. 796–804. doi:10.1007/978-3-319-93638-3_48.
[59] T. Wüchner, M. Ochoa and A. Pretschner, Malware detection with quantitative data flow graphs, in: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, 2014, pp. 271–282. doi:10.1145/2590296.2590319.
[60] T. Wüchner, M. Ochoa and A. Pretschner, Robust and effective malware detection through quantitative data flow graph metrics, in: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Cham, 2015, pp. 98–118. doi:10.1007/978-3-319-20550-2_6.
[61] F. Xiao, Z. Lin, Y. Sun and Y. Ma, Malware detection based on deep learning of behavior graphs, *Mathematical Problems in Engineering* (2019).
[62] F. Xiao, Y. Sun, D. Du, X. Li and M. Luo, A novel malware classification method based on crucial behaviour, *Mathematical Problems in Engineering* (2020).
[63] I. You and K. Yim, Malware obfuscation techniques: A brief survey, in: *Proceedings of the 5th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA'10)*, 2010, pp. 297–300.
[64] Y. Zhong, H. Yamaki and H. Takakura, A malware classification method based on similarity of function structure, in: *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, IEEE, 2012, pp. 256–261.