Beyond Labeling

Using Clustering to Build Network Behavioral Profiles of Malware Families

Nadeem, A.; Hammerschmidt, C.A.; Hernandez Ganan, C.; Verwer, S.E.

DOI
10.1007/978-3-030-62582-5_15

Publication date
2021

Document Version
Final published version

Published in
Malware Analysis using Artificial Intelligence and Deep Learning

Citation (APA)
Nadeem, A., Hammerschmidt, C. A., Hernandez Ganan, C., & Verwer, S. E. (2021). Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families. In A. Shalaginov, M. Stamp, & M. Alazab (Eds.), *Malware Analysis using Artificial Intelligence and Deep Learning* (pp. 381-409). Springer. https://doi.org/10.1007/978-3-030-62582-5_15

Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families

Azqa Nadeem, Christian Hammerschmidt, Carlos H. Gañán, and Sicco Verwer

Abstract Malware family labels are known to be inconsistent. They are also black-box since they do not represent the capabilities of malware. The current state of the art in malware capability assessment includes mostly manual approaches, which are infeasible due to the ever-increasing volume of discovered malware samples. We propose a novel unsupervised machine learning-based method called MalPaCA, which automates capability assessment by clustering the temporal behavior in malware's network traces. MalPaCA provides meaningful behavioral clusters using only 20 packet headers. Behavioral profiles are generated based on the cluster membership of malware's network traces. A Directed Acyclic Graph shows the relationship between malwares according to their overlapping behaviors. The behavioral profiles together with the DAG provide more insightful characterization of malware than current family designations. We also propose a visualization-based evaluation method for the obtained clusters to assist practitioners in understanding the clustering results. We apply MalPaCA on a financial malware dataset collected in the wild that comprises 1.1 k malware samples resulting in 3.6 M packets. Our experiments show that (i) MalPaCA successfully identifies capabilities, such as port scans and reuse of Command and Control servers; (ii) It uncovers multiple discrepancies between behavioral clusters and malware family labels; and (iii) It demonstrates the effectiveness of clustering traces using temporal features by producing an error rate of 8.3%, compared to 57.5% obtained from statistical features.

A. Nadeem (✉) · C. Hammerschmidt · C. H. Gañán · S. Verwer
Delft University of Technology, Delft, The Netherlands
e-mail: azqa.nadeem@tudelft.nl

C. Hammerschmidt
e-mail: c.a.hammerschmidt@tudelft.nl

C. H. Gañán
e-mail: c.hernandezganan@tudelft.nl

S. Verwer
e-mail: s.e.verwer@tudelft.nl

M. Stamp et al. (eds.), *Malware Analysis Using Artificial Intelligence and Deep Learning*, https://doi.org/10.1007/978-3-030-62582-5_15

1 Introduction

The first malware was discovered over thirty years ago. Yet, it is still one of the leading threats in cybersecurity.[1] AV-test, a security research institute, reported detecting over 1000 Million malware samples in 2019.[2] Anti-Virus (AV) companies play a pivotal role in analyzing malware by assigning labels to newly discovered samples. However, there are several shortcomings of malware family labels: (i) Each vendor has its own way of determining a malware family. Labels obtained from different vendors are often inconsistent [29]. (ii) The precise methods used by each vendor are proprietary and unstandardized [49]. (iii) The current labels are heavily based on static and system-level activity analysis. The problem is that malware family labels do not represent the capabilities of malware samples. The black-box (unexplainable) nature of the labeling methods also makes it impossible to verify assigned family labels, causing the evaluation of newer detection methods to depend on unreliable ground truth [33]. Moreover, network traffic is rarely used to determine family labels because of noisy ground truth and non-stationary data distribution [3]. As a result, malware samples that exhibit identical network behavior but have different code attributes end up in different families, see, e.g., Perdisci et al. [44].

In this chapter, we address the limited interpretability of malware family labels by proposing white-box[3] behavioral profiles for malware samples. Existing research suggests that network traffic shows malware's core behavior by capturing direct interactions with the attacker or C&C server [14]. Network traffic analysis can also be performed remotely, which presents a lower overhead than many popular system-activity solutions. Therefore, we place emphasis in building network behavioral profiles. To this end, we propose MalPaCA (Malware Packet Sequence Clustering and Analysis) for automated capability assessment of malware samples. The goal of *Capability Assessment* is to discover the behaviors a malware sample can exhibit. We investigate the usage of unsupervised machine learning for intelligent capability assessment to tackle the ever-increasing volume of newly discovered malware.

Until now, malware capability assessment has primarily been a manual effort [11, 40, 50], resulting in behavioral profiles that are quickly outdated. Although machine learning-based behavioral analysis approaches exist, they construct a single model that describes either the whole network or each protocol usage individually [47]. However, the network traffic originating from even a single host can be so complex that these models fail to correctly represent malicious behaviors [23]. This is why MalPaCA splits the network traffic between hosts into *uni-directional connections* and considers them as discrete behaviors (or *capabilities*).

MalPaCA clusters similar connections based on their temporal similarity, where each cluster represents a unique capability. A malware sample is then represented by its *Behavioral Profile*—a list of cluster membership of its connections. We represent

[1] https://www.cybersecurity-insiders.com/top-15-cyber-threats-for-2019/.

[2] https://www.av-test.org/en/statistics/malware/.

[3] In white-box ML, all steps are explainable—the input, output and how the output was generated. In contrast, only the input and output are known in black-box ML, e.g., Neural Networks.

malware's behavioral profiles in a Directed Acyclic Graph that shows different samples' overlapping behaviors. The graph also shows malware samples from different families behaving identically, showing potentially incorrect family labels. MalPaCA is novel as it adopts sequential features that keep the temporal nature of the traffic intact. It uses a combination of Dynamic Time Warping and Ngrams to measure the distance between network connections. MalPaCA utilizes only 20 packets to identify the network behavior shown by any given connection. It also utilizes only the packet header features that are available even when traffic is encrypted.

The last step of MalPaCA's pipeline is assigning capability labels to clusters. Each discovered cluster is visualized using *temporal heatmaps* to determine which capability it captures. The temporal heatmaps provide a goal- and data-driven approach to investigate the performance of MalPaCA's clustering, by clearly showing the network connections that are grouped together. This eliminates the need to manually investigate thousands of network traces. Security analysts can also fine-tune MalPaCA's parameters by visualizing the temporal heatmaps. The key advantage of this methodology is its white-box and explainable nature: it provides a visual representation to investigate MalPaCA's rationale for finding behavioral similarity. In doing so, we address the interpretability problem of typical black-box analysis methods, which is an important stepping stone towards better detection methods.

We evaluate MalPaCA's performance on 1.1 k malware samples (resulting in 3.6 M packets) coming from 15 families collected in the wild. We also compare the effectiveness of sequence clustering by comparing with an existing method based on frequently-used statistical (aggregate) features [54].

Results. The results are very promising: (i) MalPaCA's capability assessment works on low quality datasets with as low as 20 packets in each trace, though additional traces result in more thorough profiles; (ii) It successfully discovers several attacking capabilities, such as port scans and reuse of C&C servers; (iii) MalPaCA demonstrates the effectiveness of sequence clustering by producing an error rate of 8.3% compared to 57.5% obtained from statistical features; and (iv) MalPaCA uncovers multiple discrepancies between behavioral clusters and family labels. We believe this happens either because the labels are incorrect or because the overlapping families share significant behavior.

Contributions. We summarize our contributions as follows:

1. We show that short sequences of packet header features are capable of characterizing network behavior;
2. We build *MalPaCA*[4]—a tool to automatically build network behavioral profiles of malware samples collected in the wild;
3. We introduce *temporal heatmaps*—a data-driven and visualization-based cluster evaluation method that requires no ground truth;
4. We show the behavioral relationships between malwares using a Directed Acyclic Graph, which also uncovers discrepancies between behavioral clusters and traditional family labels;

[4] https://github.com/azqa/malpaca-pub.

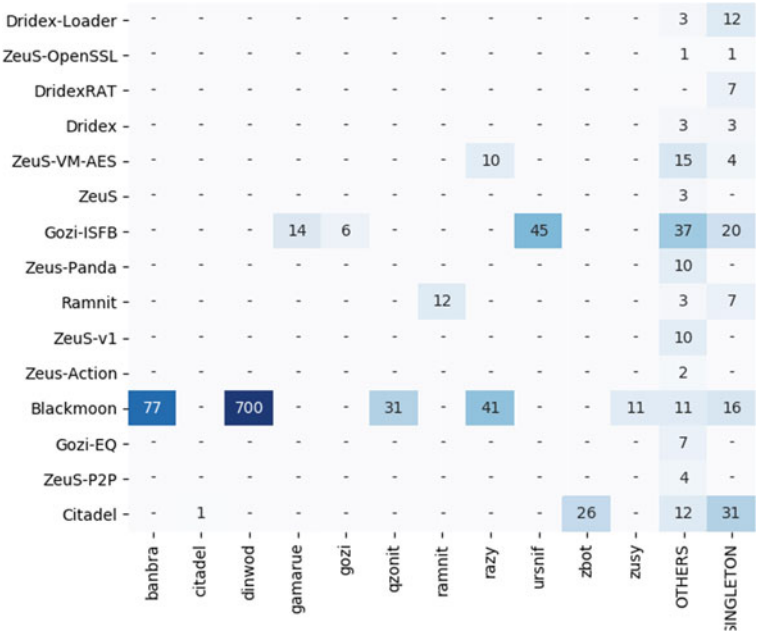


Fig. 1 Disagreements between AV vendors. Rows: YARA labels, Columns: AVClass labels, Counts: # malware binaries

5. We demonstrate the effectiveness of sequence clustering, which shows less errors than an existing solution based on statistical features.

2 The Problem with AV Labels

This section presents an analysis of our experimental dataset to emphasize the problem of inconsistent AV labels and motivates the need for explainable behavioral profiles. We compare the *agreement rate* of two popular malware labeling practices, i.e., YARA rules[5] and VirusTotal[6] labels. The malware collection process is given in Sect. 5.1. Table 2 shows the number of binaries in each malware family.

The malware binaries in the dataset are labeled using YARA rules. Each malware binary also has a Virus Total (VT) scan report. On average, there are 61 AV vendors for each malware sample, out of which 25.8% vendors per malware sample return a `null` detection, i.e., unable to detect it as malicious. The rest assign various labels to each malware binary.

[5] https://virustotal.github.io/yara/.

[6] https://www.virustotal.com/.

Since each AV vendor has its own vocabulary, a trivial filtering attempt on a VT report cannot identify the true underlying family label. Sebastian et al. [49] have developed an open source tool, called AVClass, that takes VT reports as input and returns the most likely family label. If, after all the filtering steps, AVClass is unable to identify the family name, it declares the malware as a "SINGLETON". We use AVClass to reduce a VT report into its representative VT family label. In the experimental dataset, AVClass returns "SINGLETON" for 101/1196 (8.4%) VT reports, while assigning 42 unique family labels to the rest 1095 malware binaries.

Figure 1 shows the label agreement rate between the YARA and VT labels. The y-axis shows the YARA labels. The x-axis shows the VT labels as aggregated by AVClass. For brevity, "OTHERS" category contains all samples for which $counts < 10$. Only 3 family names co-exist in both YARA and VT labels, i.e., `Citadel`, `Gozi`, and `Ramnit`. Also, although `Ramnit` is detected under the same name by both YARA and VT, 10 malware samples are still labeled differently. In fact, YARA family labels are assigned 4.2 distinct VT labels on average, while VT labels are assigned 1.5 distinct YARA labels on average. One example demonstrating this is: `YARA: Zeus-VM-AES` (29 samples) are predicted as `VT: razy` (10 samples), `gamarue` (6 samples), `cerber` (3 samples), `upatre` (3 samples), `farfli` (1 samples), `locky` (1 samples), `hpcerber` (1 samples), and `SINGLETON` (4 samples). This makes it very hard to understand the collected malware. One fair conclusion is that some VT labels can be considered as sub-families of the popular YARA malware family. For example, `Dinwod` and `Banbra` seem to be sub-families of `Blackmoon`, but the names alone do not explain which attributes set them apart from each other.

3 Related Work

The field of malware analysis has existed since the first malware was discovered over 30 years ago. Since then, multiple machine learning-based approaches have been proposed to automate malware detection and analysis. In this section, we present a brief survey of the major research challenges targeted by prior work. In doing so, we highlight how our work fills the gaps across various research themes.

3.1 Challenges in Malware Labeling

Existing research has repeatedly shown that malware family labels are noisy and inconsistent. Popular tools, such as VirusTotal, run multiple AV scanners and return an array of labels predicted by each scanner, without any indication as to which is correct. There is also an absence of a common vocabulary that all security companies can follow to label malware samples. Maggi et al. [37] propose a method to find inconsistencies in malware family labels generated by Anti-Virus (AV) scanners. Mohaisen et al. [38] are the first to measure the accuracy, consistency, and com-

pleteness of AV scanners. Their results show that AV vendors produce inconsistent labels 50% of the time, on average. These findings resulted in research that found ways to deal with the inconsistencies in the family labels. Kantchelian et al. [29] proposed an algorithm based on Expectation Maximization and Bayesian models that assign weights to each vendor's trustworthiness. Sebastián et al. [49] developed a useful open source tool, called AVClass that determines the likely family name after performing heavy filtering on all the predicted labels. However, these methods do not address the key underlying issue—malware family labels are black-box with limited interpretability.

Behavioral profiles complement family names in that they also describe the behavior of a sample. *Capability assessment* is done to characterize a malware family, which has primarily been a manual effort resulting in behavioral profiles that are quickly outdated. Also, most of the prior works in capability assessment utilize information extracted from the static analysis of malware executables: Black et al. [11] bridge the semantic gap between low-level API calls and high-level behaviors in order to build a taxonomy of banking malware. They extract API calls by statically analyzing a banking malware dataset, and map them to high-level behaviors manually with the help of domain experts. Sharma et al. [50] recently proposed a method to automatically build behavioral profiles. They select a few high-level capabilities possessed by malware by investigating the literature, and map them to low-level behaviors extracted from the static analysis of 56 malware samples. *In contrast, we propose MalPaCA that automatically builds dynamic (network) behavioral profiles.*

3.2 Research Objectives: Detection Versus Analysis

Existing research on malware comes in two strains: detection-based and analysis-based. Malware detection and signature generation dominates existing literature, with the end-goal of optimizing metrics [1, 2, 7, 10, 17, 24, 35, 36, 39, 44, 46, 54, 60], while only a few of these works also help the readers understand and analyze the obtained results [23, 43]. Recently, however, several malware analysis approaches have been proposed that aim to improve malware understandability rather than optimizing detection rates. These methods provide essential insights that can improve malware detection methods. Black et al. [11] perform an in-depth analysis of the key behaviors of banking malware families and how they have evolved over time. Moubarak et al. [40] discuss malware evolution and the structural relationship between several potentially state-sponsored malware. In [51], the authors cluster Android malware samples and build a dendrogram of the malware families showing overlapping code snippets. Sharma et al. [50] build behavioral profiles of malware samples using static analysis. *In this chapter, we follow a similar approach and build an analysis tool, MalPaCA. MalPaCA uses unsupervised clustering to group network connections that behave similarly and uses them to construct malware's behavioral profiles.*

Although clustering is an unsupervised technique, existing literature has often used some notion of ground truth (family labels) to evaluate the cluster quality. Bayer et al. [7] evaluate their malware clustering approach using labels obtained by the majority voting of 6 AV vendors. Perdisci et al. [44] evaluate their malware clustering approach by introducing a notion of AV graphs that depict the agreement between AV vendors as a measure of cluster cohesion and separation. In [35], the authors report the precision and recall of higher than 0.95 of their malware clustering approach. They use the majority voted family labels from 25 AV vendors as their ground truth. Li et al. [33] have examined the challenges of evaluating malware clustering and have advised caution when deciphering highly accurate clustering results as they can be impacted by spatial bias: performing majority voting on AV-provided labels is hazardous, since if most of the AV vendors are in agreement, it typically indicates that the families are already easy to detect. *In this chapter, we propose a data-driven and visualization-based method to evaluate clusters, without using family labels. Instead of optimizing clustering accuracy, our emphasis is on explainability of the results.*

3.3 Challenges in Malware Behavior Modeling

Modeling software behavior is a challenging task, but modeling malware's behavior is even more challenging since malware authors specifically try to evade detection [15]. Static analysis of malware binaries and disassembled code has been a popular malware analysis approach in the literature [6, 11, 21, 35, 39]. Increasingly more malware uses obfuscation techniques to evade analysis, causing difficulties for statically analyzing malware. The obfuscation attempts gave rise to dynamic analysis of malware that executes a malware sample in a sandbox and collects execution traces from it. Dynamic analysis is generally divided into two strains: System activity and Network traffic analysis. Network traffic analysis collects traces of malware samples remotely using existing network monitoring infrastructures [44], making it much easier to apply. However, the behavioral analysis and signature generation literature is heavily focused on system activity analysis, e.g., see [7, 16, 50, 52]. Research suggests that network traffic shows the core behavior of malware [14]. Although sometimes encrypted, network traffic contains the direct interaction with the attacker. In this section, we discuss three major challenges of modeling malware behavior via traffic analyses.

Feature selection. Network traffic analysis is generally applied when designing Network Intrusion Detection Systems (NIDS), which either detect anomalous traffic [24] or generate signatures for malware families [22, 26, 55]. Deep Packet Inspection (DPI) is one commonly used approach in NIDS to extract information from packet payloads. For example, Rafique et al. [46] use DPI for automatic signature generation of malware families. Although effective, downsides to DPI-based approaches are that they are privacy-intrusive, operationally expensive, and do not work out-of-

the-box for encrypted traffic. There are also approaches that detect specific attacks. For example, HTTP-based malware can be detected using specific features from the Application header [44]. Similar approaches exist for DNS-based malware [32, 45], and HTTPs-based malware [4]. In the absence of the HTTP, DNS, and TLS headers, these approaches seize to work.

Several works use *coarse* or high-level features that are protocol-agnostic and work out-of-the-box even with encrypted traffic. For example, Conti et al. [17] use sequences of packet sizes to characterize the network behaviors generated by Android applications. Aiolli et al. [2] use various statistical features computed over packet sizes to detect bitcoin wallet application functionality. Acar et al. [1] use network traffic direction and packet lengths to identify commands issued to smart home IoT devices. These works aim to characterize benign network behaviors. In the malware domain, Tegeler et al. [54] use average packet size, average packet inter-arrival time, average connection duration, and the FFT of C&C communication to detect bot-infected hosts. Garcia [23] builds a behavioral Intrusion Detection System by using the size, duration, and periodicity of Netflows. *In this chapter, we also use high-level features from packet headers to characterize malware's network behavior. To the best of our knowledge, network traffic analysis has not been used in capability assessment or for generating behavioral profiles of malware samples.*

Feature representation. Machine learning methods take a feature vector as input, which can represent anything ranging from a single behavior to a complete malware sample. Multiple observations for a single feature are aggregated into *statistical features*, e.g., mean packet size of a netflow. Existing literature is filled with approaches that use such statistical features, e.g., see [5, 10, 23, 54]. Although they are computationally efficient, they lose local behavioral details, which can be a problem when the goal is to characterize that behavior.

Another approach that is gaining momentum is the use of *sequential features*. Numeric sequential features are typically used in two ways: *Discretized* and *Raw sequences*. A raw sequence (or a continuous sequence) is composed of the original observations, while a discretized sequence encodes the observations into a finite set of bins. Discretizing sequences is typically faster and makes measuring distances easier. Pellegrino et al. [43] learn state machines from discretized netflow data in order to detect bot-infected traffic, while Hammerschmidt et al. [27] use it to cluster host behavior over time. Lin et al. [36] detect anomalies in industrial water treatment plant by using discretized sequences from sensor readings. In practice, malware-related data is often scarce and noisy. In this case, discretization can lose important information.

Raw sequences are rarely used for modeling network traffic because it is non-stationary and contains noise (e.g., empty acknowledgment packets or retransmissions), and delays (due to varying network latency) [3]. Ntlangu et al. [41] provide a brief overview of time-series approaches to model network traffic. As noted in [41], due to the nature of network traffic and their distributions, (auto-)regressive models struggle to accurately capture them. Kim et al. [30] use a multi-variate time-series regression model on host-based resource consumption, such as CPU and memory

usage (not network traffic) to identify Android malware. Conti et al. [17] propose a method to detect the action performed by Android applications using raw sequential features. *To the best our knowledge, MalPaCA is the first method that successfully uses short raw sequential features to characterize malware network behavior.*

Distance measure. The notion of behavioral similarity requires the means to be able to measure distance between two objects. The choice of the distance measure is directly dependent on the data type of the feature set (e.g., numeric or categorical) and the way the features are represented (e.g., statistical or sequential). For statistical features, Euclidean distance is most commonly used. For instance, Chan et al. [16] use Euclidean distance to determine similar Android processes.

Calculating the distance between sequential features is more challenging because they may not always be properly aligned. For categorical (or discretized) sequences, there exist Bioinformatics inspired solutions using sequence alignment [57]. They require pre-computed substitution matrices, which currently do not exist for malware. There also exist String matching solutions frequently used in the Natural Language Processing domain. Baysa et al. [8] use Levenshtein, or edit distance, to measure the similarity between two malware binary files. A sequence can also be broken down into sub-sequences, represented as Ngrams, which have been used to model genomic sequences [58] and to match files [34]. They have also been used to classify malware families in [13]. Longest Common Subsequence (LCS) with k-gaps can also be used to measure distances between sequences. The gaps account for the occasional noise. Chan et al. [16] use LCS to group similar resource-access-patterns (not network traffic) in Android applications.

A few distance measures exist for raw or continuous sequences. Verwer et al. [56] have used Kullback–Leibler divergence to measure the distance between two sequences while learning probabilistic automata. However, it requires substantial amount of data to measure the similarity with a high confidence, which is not always available for malware. Another promising distance measure is Dynamic Time Warping (DTW). DTW has been used in fingerprint verification [31], characterizing DDoS attack dynamics [59], and measuring similarity in android application behavior [17]. *MalPaCA uses a combination of DTW and Ngrams to measure the distance between network connections.*

4 MalPaCA: Malware Packet Sequence Clustering and Analysis

The ultimate goal of MalPaCA is to construct a behavioral profile for each malware sample that is more descriptive than its family label. Research shows that malware belonging to the same family exhibits similar behaviors since malware authors often share code and resources [53]. To this end, MalPaCA automatically identifies the various network behaviors exhibited by malware samples, and groups samples that share common behavior. MalPaCA does not assume any a priori knowledge about

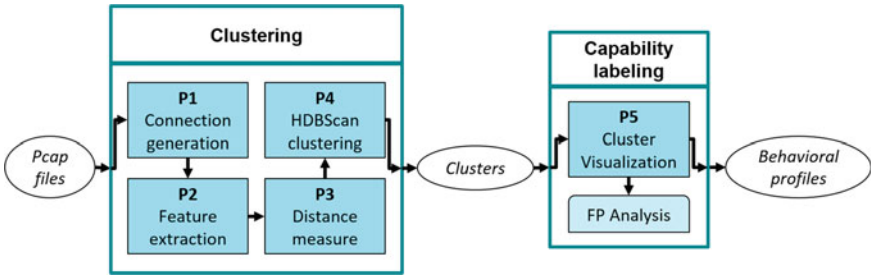


Fig. 2 MalPaCA: Connections clustered on behavioral similarity; malware described using connections' cluster membership

the malware's family name or its capabilities, and hence can be used out-of-the-box for other malware datasets. The profiles are built using observed behavior since only the executed functionality is relevant for behavioral profiling. Profiles for individual families can be enriched further by observing additional traffic. We release MalPaCA to the public.[7]

Figure 2 illustrates the architecture of MalPaCA with its five phases (P1 to P5). Network traces (Pcap files) are given as input to the system, which are split into uni-directional packet streams (or *connections*) that are clustered based on temporal similarities. Each cluster is assigned a capability label by visualizing temporal heatmaps showing connections' feature values. Each malware sample (and its associated Pcap file) is then described by a *Cluster Membership String*, forming a descriptive behavioral profile.

4.1 Connection Generation (P1)

A *connection* is defined as an uninterrupted uni-directional list of all packets sent from source IP to destination IP address. This means 8.8.8.8 → 123.123.123.123 is a different connection than 123.123.123.123 → 8.8.8.8. We refer to these as *Outgoing* and *Incoming* connections based on their direction with respect to the `localhost`. Note that we do not use IP address as a feature, except to create connections.

Ideally, a connection captures one complete capability. The connection length can vary significantly depending upon the behavior and network delays. Since the network delay is an artifact of the network, not of the malware, it is important to reduce its impact when measuring behavioral similarity. MalPaCA does so by capping the sequence length to a fixed threshold, avoiding artifacts that are due to connection length.

[7] https://github.com/azqa/malpaca-pub.

Existing research suggests that it is possible to identify behavioral differences from a *handshake*.[8] Wang et al. [61] use the first 3 to 12 bytes of packet headers in order to identify the different so-called Protocol Format Messages. MalPaCA builds upon this idea and utilizes the first few packets of a connection to identify the capability. This is a fixed threshold denoted by the tunable parameter len. It should be large enough to allow the handshake to be modeled, the length of which is often unknown in network traffic analysis. Larger values of len not only include noise artifacts but also increase the computational resources required to process longer connections.

4.2 Feature-Set Extraction (P2)

The choice of feature-set is crucial for determining the kind of behaviors that are identified by MalPaCA. Two considerations motivate our choice: (1) MalPaCA should be generalizable to more than one type of malware; (2) The feature set is small and easy to extract. Hence, we do not use features extracted from the packet payload itself as they limit the applicability of the method. We also do not use IP addresses as they are easy to spoof and are considered Personally Identifiable Information[9] in countries like the Netherlands. We use four sequential features: (i) packet size, (ii) time interval, (iii) source port, (iv) destination port. All four features are independent of the protocol type, making them available for every connection. Each feature is represented as a sequence of raw observations for subsequent packets. Although these features are simplistic, we demonstrate that their sequential nature captures malware behavior effectively.

Packet size (f_{ps}) measures the size of the *IP datagram* of each packet in bytes. *Time interval* (f_{in}) captures the inter packet arrival time in milliseconds. We use time interval because malware tends to show a periodic behavior, e.g., bots send periodic heartbeat packets[10] to inform the C&C server about the infected host. MalPaCA is meant to be used on a single network at a time since using inter-arrival time makes connections collected on different latency networks incomparable.

We use both *source* (f_{sp}) and *destination* (f_{dp}) *port numbers* because the connections are uni-directional. We particularly use source port so the analysts can limit the use of problematic ports in case of outgoing connections. The usage of certain vulnerable ports can also indicate suspicious activity. Each connection is represented by four sequences, one per feature, $C = (f_{ps}, f_{in}, f_{sp}, f_{dp})$.

[8] *Handshake traffic refers to the introductory few packets of a connection.*

[9] https://www.enterprisetimes.co.uk/2016/10/20/ecj-rules-ip-address-is-pii/.

[10] https://www.ixiacom.com/company/blog/mirai-botnet-things.

4.3 Distance Measure (P3)

Three considerations motivate our choice of distance measure: (1) Different distance measures are applicable on numeric and categorical data types; (2) The distance measure should be intuitive to help understand the results; (3) It must produce results that are resilient to delays and noise, which are common characteristics of network traces. The last consideration was added after observing distance measures producing results that were artifacts of network delays. MalPaCA uses a combination of Dynamic Time Warping (DTW) and Ngram analysis to measure distance between two connections.

Dynamic Time Warping. DTW [9] is used to measure distances between numeric sequences (packet size and time interval) due to its robustness to delays and noise. It aligns two time-series that may contain distortions (or warps) in the time-axis. It maps local substructures in one sequence to those of the other sequence. For two sequences $a = [a_0, a_1, \ldots, a_n]$ and $b = [b_0, b_1, \ldots, b_m]$ the DTW distance $d_{dtw}(a, b)$ is

$$d_{dtw}(a,b) = \sum_{i=1}^{n+1}\sum_{j=1}^{m+1} ||a_i - b_j|| + \min \begin{cases} d(a_{i-1}, b_j), \\ d(a_i, b_{j-1}), \\ d(a_{i-1}, b_{j-1})) \end{cases} \quad (1)$$

The output is a *similarity* score, which we normalize using:

$$d_{ndtw}(a,b) = \frac{d_{dtw}(a,b) - \min_{x,y}\{d_{dtw}(x,y)\}}{\max_{x,y}\{d_{dtw}(x,y)\} - \min_{x,y}\{d_{dtw}(x,y)\}} \quad (2)$$

Ngram analysis. An *Ngram* is defined as the set of n (called $order$) consecutive items in a given sequence. The larger the value of $order$, the more sequence structure is captured. A sequence of port numbers is converted into a set of Ngrams, called its *Ngram profile* using a sliding window of length $order$. An example for $order = 2$ is shown in Table 1, where A, B, C, D are hypothetical port numbers. Let G be the set of all unique Ngrams occurring in the dataset. For each packet sequence a, a vector $a_g = [f(g_1, a), f(g_2, a), \ldots, f(g_{|G|}, a)]$ is generated, containing the occurrence frequencies $f(g_i, a)$ in a of each Ngram $g_i \in G$.

We measure the distance between two Ngram profiles using Cosine distance. Other distance measures exist for Ngrams, but Cosine has shown promise in measuring

Table 1 Example—Distance measurement using Ngram analysis

Input	Ngram profiles	G = [AB, BC, CB, DA, CA]	Cosine distance
$ABCBC$	AB, BC, CB, BC	[1, 2, 1, 0, 0]	0.3876
$DABCA$	DA, AB, BC, CA	[1, 1, 0, 1, 1]	

similarity between categorical sequences [63]. It is determined by the angle between two non-zero vectors. The similarity value lies between 0 and 1, where 1 means that the two vectors are the same (parallel to each other) and 0 means they are completely different (orthogonal to each other). For two sequences in their vector representations $a = [v_1, \ldots, v_{|G|}]$ and $b = [v'_1, \ldots, v'_{|G|}]$, the Cosine distance $d_{cos}(a, b)$ is

$$d_{cos}(a, b) = 1 - \frac{\sum_{i=1}^{|G|} a_i \times b_i}{\sqrt{\sum_{i=1}^{|G|} a_i^2} \times \sqrt{\sum_{i=1}^{|G|} b_i^2}} \tag{3}$$

Finally, the DTW and cosine distances are combined to calculate the final distance between two connections:

$$d_{conn}(A, B) = \frac{d_{ndtw}(a_{ps}, b_{ps}) + d_{ndtw}(a_{in}, b_{in}) + d_{cos}(a_{sp}, b_{sp}) + d_{cos}(a_{dp}, b_{dp})}{4} \tag{4}$$

where $A = (a_{ps}, a_{in}, a_{sp}, a_{dp})$ and $B = (b_{ps}, b_{in}, b_{sp}, b_{dp})$ are connections and their features: packet sizes $\{a|b\}_{ps}$, intervals $\{a|b\}_{in}$, source port Ngram profiles $\{a|b\}_{ps}$, and destination port Ngram profiles $\{a|b\}_{dp}$.

4.4 HDBScan Clustering (P4)

A key strength of MalPaCA is the clustering algorithm it uses. There exists a familial structure among malware behaviors [51, 55]. Therefore, it makes sense to use hierarchical clustering to model the relationships between them. We have used Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBScan) [12] for this purpose. The key strengths of HDBScan are twofold: it automatically determines the optimal number of clusters, and it generates high-quality clusters that remain stable over time. It also has minimal tunable parameters, which allow configurations to be generalizable.

HDBScan requires a pairwise distance matrix as input. It does not force data points to become part of clusters—all data points whose membership to a cluster cannot be determined are considered to be *noise*. In our context, *noise* refers to behaviors that are either too different from all the others or cannot be clearly assigned to one cluster. An ideal dataset with clear cluster boundaries will have no noise. Hence, in the presence of a less ideal dataset, noise is discarded to extract high-quality clusters. Keep in mind that discarding excessive connections as noise can also be counterproductive. We discuss this limitation in Sect. 8.

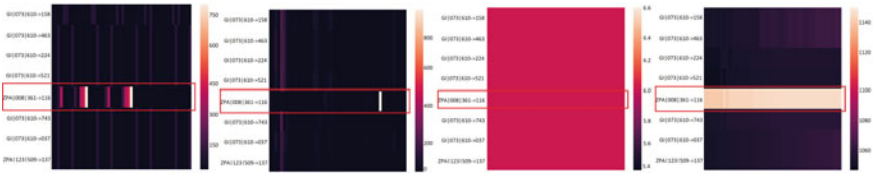
4.5 Cluster Visualization (P5)

Formalizing cluster quality without ground truth is a fundamental challenge in clustering. Although some metrics exist that capture cluster quality (i.e., Silhouette index [48] and DB Index [18]), they require a notion of distance from a cluster centroid, which is difficult to obtain for sequences. In MalPaCA, each connection is represented by four sequences and collapsing these into a single cluster quality measure loses important local behavior. Instead, we define the following properties to be indicative of good clustering: (1) Cluster homogeneity is high—a cluster contains only similar connections. (2) Cluster separation is high—each cluster captures a unique capability. (3) Clusters are small and specific so they only capture the core capability. The first two properties ensure that we obtain meaningful capability-based clusters, the third ensures that only the core capabilities are captured.

We use *temporal heatmaps* for a white-box cluster analysis. We graphically show the connection features and rely on human visualization skills to determine cluster quality. Analysts can inspect heatmaps to determine which behavior is captured in a cluster. This gives them control over the clustering results. We leave the automation of this process as future work.

Four temporal heatmaps are associated with each cluster, one corresponding to each feature. Each row in a heatmap shows the corresponding feature sequence of the first *len* packets in a connection. Figure 3 shows example temporal heatmaps. The figure highlights one dissimilar connection among the eight in the cluster, clearly highlighted in red.

Clustering Error Analysis. Visualizing the cluster content helps to identify which connections belong in a cluster. A *Clustering Error* (CE) is defined as a connection that is placed in cluster `X` despite half of its features being different from the remaining connections in the cluster. Since each feature holds equal weight, we only consider a connection as CE if more than two features *differ*. We consider two features *different* if more than 50% of their sequences differ so significantly that a different color appears on the temporal heatmap. This is where human visualization skills play a key role in determining feature similarity. Figure 3 shows a cluster containing one CE, highlighted in red. It shows that three out of four feature values of this connection are different from other connections in the same cluster. The clustering error rate is



(a) Packet sizes (b) Interval (c) Source Port (d) Destination Port

Fig. 3 A clustering error: one connection does not belong in the cluster it is assigned

calculated as $\frac{CEs}{Clustersize}$, i.e., $\frac{1}{8}$. We measure the error rate of each cluster similarly and calculate the *average percentage of errors per cluster* as a notion of clustering quality.

In practice, we first establish the common majority by finding two or more connections that are most similar to one another, i.e., the ones that have the least mutual distance. The pairwise distance matrix computed during clustering is used as a lookup table for finding such connections. Figure 3 shows a simple case where the *rightful owners of a cluster* are easily visible since 7 out of 8 connections are very similar. The rest of the connections are compared with the rightful owners and are either considered as true positives or clustering errors, depending on how many feature sequences differ.

5 Experimental Setup

In this section, we describe the dataset used for the experiments and the configuration details of MalPaCA's parameters.

5.1 Experimental Dataset

MalPaCA was evaluated on financial malware samples collected in the wild. We worked in collaboration with a security company that specializes in malware analysis and threat intelligence. They collected the dataset independently. The dataset contained 1196 malware samples that were collected over one year. Each malware sample was executed in a sandboxed environment containing several virtual machines. The resulting network traffic was stored in a Pcap file. Some samples showed sandbox evasion. They were re-executed in a VM with different settings. This resulted in a total of 1196 Pcap files. Uni-directional connections were extracted, resulting a total of 8997 connections containing 3.6 M packets.

The dataset contains 15 famous financial malware families. They were labeled by the security company using their proprietary YARA rules. Additionally, each sample was submitted to VirusTotal (VT), which hosts 68 AV vendors. For each sample, VT returns a report containing detection results from each vendor. Table 2 summarizes the dataset.

5.2 MalPaCA Parameters

MalPaCA has four parameters, i.e., $order$ of the Ngrams used for port numbers, len of packet sequences for features, and the two parameters of HDBScan clustering algorithm: $Minimum\_Cluster\_Size$ and $K\_nearest\_neighbors$. In our experi-

Table 2 Experimental dataset: malware binaries and their associated YARA family labels

Family name (YARA)	# Malware binaries
Blackmoon (B)	887 (74.10%)
Gozi-ISFB (GI)	122 (10.19%)
Citadel (C)	70 (5.85%)
Zeus-VM-AES (ZVA)	29 (2.42%)
Ramnit (R)	22 (1.83%)
Dridex-Loader (DL)	15 (1.25%)
Zeus-v1 (Zv1)	10 (0.83%)
Zeus-Panda (ZPa)	10 (0.83%)
Gozi-EQ (GE)	7 (0.58%)
Dridex RAT Fake Pin	7 (0.58%)
Dridex (D)	6 (0.50%)
Zeus-P2P (ZP)	4 (0.33%)
Zeus (Z)	3 (0.25%)
Zeus OpenSSL	2 (0.17%)
Zeus Action	2 (0.16%)
Total	**1,196** (100%)

ments, we have used trigrams ($order = 3$) for port numbers, because they form a good trade-off between performance and data sparsity [28]. In the experimental dataset, the length of connections is highly skewed towards shorter sequences, with a mean of 20 packets. We use this mean as len.[11] Out of 8997 connections in the dataset, 733 connections are longer than len. The HDBScan algorithm uses $Minimum\_Cluster\_Size = 7$ and $K\_nearest\_neighbors = 7$. These parameters were selected by tuning MalPaCA on a configuration dataset (5% of the usable data). The experiments were run on a machine with Intel Xeon E3-12xx v2 processor, 8 cores and 64 GB RAM.

The specificity of the identified behaviors is highly dependent on the length of sequences, i.e., len. Based on preliminary experiments with $len = \{5, 10, 20, 50\}$, we found that $len = 20$ provided the optimal trade-off between behavior characterization and the amount of connections that were discarded. For smaller values, the connections were too generic. For larger values, connections with slight behavioral differences were considered very different. For example, at $len = 50$ several clusters capture slightly different variations of port scans, while at $len = 20$ those variations merge to form a few strong clusters.

[11] *len can be adjusted based on the required behavioral specificity.*

6 Malware Capability Assessment

MalPaCA produces 18 clusters from the dataset. There are, on average, 25 connections in each cluster. The algorithm discards 284 connections as noise. The remaining 449 connections originate from 216 Pcap files. Each cluster captures a unique behavior, listed in Table 3 along with the malware families that show that behavior. We describe a few of the interesting behaviors obtained by MalPaCA. We also discuss how host-based blacklisting [25, 54], which is a very common practice in security companies, will fail to detect these behaviors.

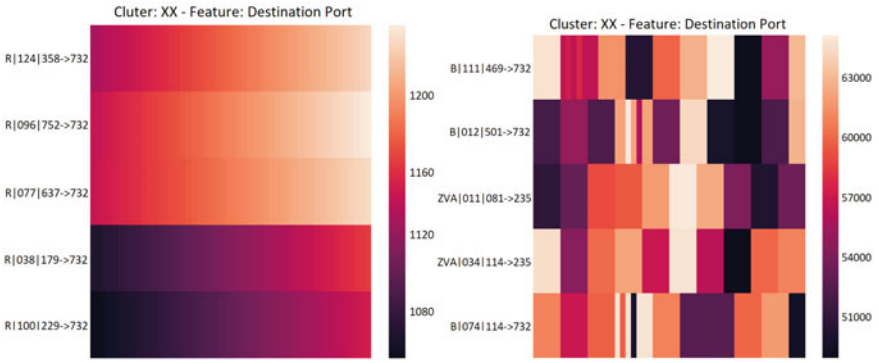
1. **Connection Direction Identification.** MalPaCA successfully identifies the direction of traffic flow even though no such feature is used. The clusters and their traffic direction are listed in Table 3. Interestingly, we continue to see this pattern even when port-related features are removed from the clustering. Hence, the sequence of packet sizes and their inter-arrival time are collectively indicative of the flow direction. This important trait identifies whether the suspicious behavior is originating from inside the network or from outside it.

Table 3 For each cluster, (i) # connections, (ii) # malware families, (iii) Capability label, and (iv) Traffic direction

Cluster	# Conns	# Families	Behavior	Direction
c1	39	***9 (Common)***	SSDP traffic	Out
c2	90	***9 (Common)***	Broadcast traffic	Out
c3	9	4	LLMNR traffic	Out
c4	49	5	Systematic port scan	In
c5	56	5	Randomized port scan	Out
c6	25	***1 (Rare)***	Connection spam	In
c7	23	***1 (Rare)***	Connection spam	Out
c8	16	***1 (Rare)***	Malicious subnet	Out
c9	11	***1 (Rare)***	Connection spam	Out
c10	9	2	HTTPs traffic	Out
c11	8	2	C&C Reuse	In
c12	18	4	HTTPs traffic	In
c13	25	5	Misc.	In
c14	10	3	Misc.	In
c15	20	3	Misc.	In
c16	12	3	Misc.	Out
c17	19	3	Misc.	Out
c18	10	4	Misc.	Out

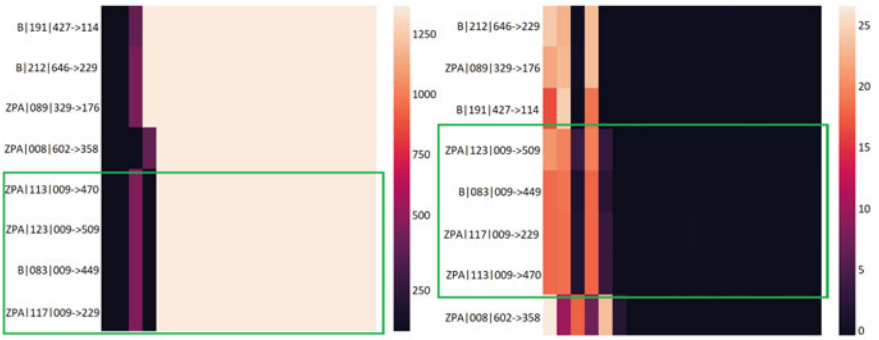
2. **Device Probing.** Some clusters capture connections that connect to the same host. For example, one cluster contains all connections broadcasting to 239.255.255.250, which is used by the SSDP protocol to find Plug and Play devices. Another cluster captures all connections broadcasting to 224.0.0.252, which is used by the Link-Local Multicast Name Resolution (LLMNR) protocol to find local network computers. These clusters could easily have been obtained by using IP-based blacklist, but they would not have clustered behaviorally similar hosts with different IP addresses.
3. **Split-personality C&C Servers.** In several instances, an infected host was observed responding differently to the same request, so much so that the resulting connections ended up in different clusters. For example, two connections of `Gozi-ISFB` contact 46.38.238.XX, which has been reported as a malicious server located in Germany. The outgoing connections are identical as they both request for the same resource. However, the responses received are very different—the first response contains a small packet followed by a series of 1200-byte packets, while the second one contains a periodic list of small and large packets in the range of 600–1800 bytes. This insight portrays a better picture of the behavior of said C&C server. In contrast, a blacklist would have grouped these connections since they belong to the same host.
4. **Port Scan Detection.** Some clusters capture a *Port Scan*,[12] which is a method for determining open ports on a device in a network. Port scans are usually a part of the reconnaissance phase in the attack kill chain [62]. Utilizing sequences of port numbers enables us to detect any suspicious temporal behavior before an attack happens. The clusters identify two types of port scans: (i) *Systematic port scan* where ports are swept incrementally, which is seen as a gradient in the corresponding temporal heatmap; and (ii) *Randomized port scan* where ports are contacted randomly, which shows up in the heatmap as a checkered pattern. See Fig. 4. Port scans carried out by different connections are clustered together if they contact the same range of port numbers, which increases their mutual similarity. This result is in direct contrast with Mohaisen et al. [39] who conclude that port numbers are the least useful features in distinguishing malware families.
5. **C&C Reuse by Multiple Families.** One cluster contains connections from different families that contact the same C&C server, and their temporal heatmaps look behaviorally identical. The cluster includes three `Zeus-Panda` (ZPA) connections and one `Blackmoon` (B) connection who contact a single IP address (encoded as 009), which has been reported as malicious. Figure 5 shows the temporal heatmaps of this cluster. The said connections are highlighted in green. This result suggests that either the YARA rules mislabeled one of the samples or that authors share C&C servers.
6. **Malicious Subnet Identification.** In some instances, several connections contact IP addresses that fall in the same subnet. For example, two `Zeus-VM-AES` connections contact one host from 62.113.203.XX subnet, while another connection detected 15 days later contacts another host in the said subnet. Similarly, two

[12] https://whatismyipaddress.com/port-scan.



(a) Systematic port scan. (b) Randomized port scan.

Fig. 4 Clusters showing systematic and randomized port scans



(a) Packet sizes (b) Interval

Fig. 5 Similar Zeus-Panda and Blackmoon connections

`Zeus-Panda` connections and one Blackmoon connection contact two hosts in 88.221.14.XX subnet. This gives actionable intelligence to ISPs to investigate if other IPs in these subnets are also hosting C&C servers.

6.1 Cluster Characterization

We analyze the temporal heatmaps for the behavioral trend of each cluster in order to label it. MalPaCA's goal is to identify different behaviors in the network traffic and it does so regardless of their maliciousness and origin. Hence, the resulting clusters contain both, benign and malicious behaviors. The common clusters can be discarded if they contain known-benign behaviors, drastically reducing the number of connections to analyze.

We successfully assigned labels to 12 clusters. For example, in the case of connection spam, the whole cluster is filled with almost identical connections originating from the same host. We validate this observation by specifically looking at the network traffic of these connections to see exactly what behavior is shown. Six clusters were left unlabeled since we could not identify the captured capability simply by exploring temporal heatmaps. These particular clusters were also the source of clustering errors. Table 3 shows that *SSDP* and *Broadcast traffic* are the most common behaviors and are both specific to Windows OS. Since the dataset is composed of Windows-based malware, it explains why 9 out of 12 families have connections in these two clusters. On the contrary, *Connection Spam* and *Malicious Subnet* are the rarest behaviors. *Malicious Subnet* only captures `Zeus-VM-AES`. `Gozi-ISFB` opens numerous connections, creating a *Connection Spam*. The incoming connections are stored in one cluster, while the outgoing traffic is split into two clusters due to the difference in the type of requests. This detailed behavioral analysis enables the identification of interesting clusters to analyze further.

Performance Analysis. The temporal heatmaps show that on average, 8.3% connections per cluster are CEs—their feature sequences are different from their fellow connections in a cluster. The majority of the errors originate from the last six clusters. Note that this error rate is low for an unsupervised setting since not all connections require manual revision.

6.2 Constructing Behavioral Profiles

MalPaCA identifies 18 distinct behaviors in the dataset. Hence, each malware sample (and its associated Pcap file) can be described as a binary string of 18 characters, known as *Cluster Membership String (CMS)*, where each character signifies whether the Pcap's connections were found in that cluster. Precisely, for a malware sample x, $CMS_x = b^n$, where $b \in \{0, 1\}$, n is the number of behavioral clusters, and b^i indicates whether x's connections are present in the ith cluster. The Cluster Membership String can be regarded as the behavioral profile of a given malware sample. In this work, we consider binary CMSs because we are only interested in the behavior overlap of different malware samples. Non-binary $CMS_x = z^n$, for connection counts $z \in \mathbb{Z}$, is an interesting avenue to investigate.

Table 4 lists the composite behavioral profiles for each YARA malware family in the dataset—each YARA family is represented as the union of all its samples' CMSs. `Dridex`, `Gozi-EQ`, `Zeus-P2P` and `Zeus-v1` only generate either *SSDP* or *Broadcast traffic*. Since this traffic is obtained from standard Windows services, it is likely that the malware was not activated when the associated Pcap files were recorded. Hence, the only connections observed from these families seem benign. `Gozi-ISFB` has the most diverse profile, with connection in 16 out of 18 clusters, which exhibit attacking capabilities such as *Port Scans* and *Connection Spamming*. Specifically, the *Connection Spamming* behavior is never exhibited by any other

Table 4 Composite behavioral profiles of malware families. Columns: YARA labels, Rows: Cluster labels by MalPaCA

	B	C	D	DL	GE	GI	R	Z	ZP	ZPa	Zv1	ZVA
SSDP traffic	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	–	✓
Broadcast traffic	✓	✓	–	✓	–	✓	✓	–	✓	–	✓	✓
LLMNR traffic	✓	✓	–	✓	–	✓	–	–	–	–	–	–
System. port scan	✓	✓	–	–	–	✓	✓	–	–	–	–	✓
Random. port scan	✓	✓	–	–	–	✓	✓	–	–	–	–	✓
In conn spam	–	–	–	–	–	✓	–	–	–	–	–	–
Out conn spam	–	–	–	–	–	✓	–	–	–	–	–	–
Malicious Subnet	–	–	–	–	–	–	–	–	–	–	–	✓
In HTTPs	–	✓	–	✓	–	✓	–	–	–	✓	–	–
Out HTTPs	–	–	–	–	–	✓	–	–	–	✓	–	–
C&C reuse	✓	–	–	–	–	–	–	–	–	✓	–	–
Misc.	✓	✓	–	✓	–	✓	–	✓	–	✓	–	✓
# Clusters	7	11	***1***	8	***1***	***16***	4	2	***1***	7	***1***	7

malware family in the dataset. There are two reasons for `Gozi-ISFB`'s diversity: (i) `Gozi-ISFB` is the largest family under consideration, so many of its behavioral aspects are captured; and (ii) `Gozi-ISFB` opens more connections per sample compared to other families. For example, one sample of `Gozi-ISFB` opens 111 connections, while the average number of connections for other malware samples is 3.

6.3 Showing Relationships Using DAG

We extract the behavioral relationships between the 216 Cluster Membership Strings by considering it a *Set Membership* problem. It dictates that, e.g., `Set A= {0,1,1}` is a *subset* of `Set B={1,1,1}` because `Set B` encapsulates all of `Set A`'s behaviors and more. Similarly, `Set C= {0,0,0}` is a subset of every other set in this domain. `Set C` represents Pcaps where all connections were discarded as Noise due to significant differences in behavior.

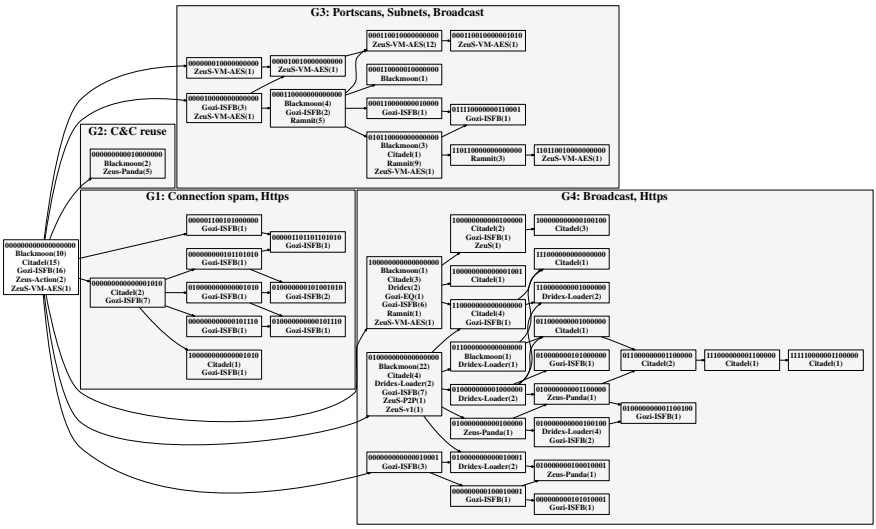


Fig. 6 MalPaCA's behavioral profiles: The DAG shows the behavioral relationships between malware samples. Each node shows a CMS and compares with YARA family labels (+ # Pcaps)

We represent the relationships between Pcap files using a Directed Acyclic Graph (DAG), shown in Fig. 6. Each node represents a unique Cluster Membership String. Multiple Pcaps can share a single CMS IFF their behaviors overlap. The nodes with minimum Hamming distance are connected using edges. This method allows multiple parents, i.e., a CMS of `"111"` may be reached by both `"110"` and `"101"`. Note that this graph is constructed purely from a data-driven approach without using any knowledge of family labels. In combination with human intelligence, we believe that it can serve as a powerful tool in understanding malware's network behavior.

7 Comparative Analysis

We show MalPaCA's results in relation to existing work by conducting two comparative analyses: (i) Comparing MalPaCA's behavioral profiles with YARA family labels, and (ii) Comparing MalPaCA's cluster quality with an existing approach that uses statistical features.

7.1 Comparison with Traditional Family Labels

We use the DAG from Fig. 6 to contrast between YARA labels and MalPaCA's behavioral profiles. Each node shows a unique CMS, and the number of malware families

that share it. For example, the node with the CMS of "0000000000000001010" is labeled as "Citadel(2), Gozi-ISFB(7)" because 2 Citadel Pcaps and 7 Gozi-ISFB Pcaps show the same behavior—their connections are co-located in the clusters 15 and 17. The root (on the left most side) contains the Pcaps for which all connections were discarded as Noise. Pcaps showing subsequently more behaviors are placed towards the right of the graph, with the right most node "11111000000110000 Citadel(1)" containing one Citadel Pcap that shows the most diverse number of behaviors. Note that observing additional network traffic will enrich this graph even further.

The graph shows four major partitions (denoted by G1-G4), indicating that there are four high-level behavioral sub-groups present in the dataset. The G2 group containing only one node stands out. It contains Pcaps from Zeus-Panda and Blackmoon, and are the only malware samples that share a C&C server. This observation makes a strong case that these particular Pcap files, albeit originating from two families, are behaviorally alike. The G3 group contains Pcaps from various families that are observed doing port scans and broadcasting behaviors. Some servers from this group also form malicious subnets. The G4 group, on the other hand, is the largest group that uses HTTPs traffic along with broadcasting behaviors. The G1 group is highly dominated by Gozi-ISFB and is observed doing Connection spamming, along with using HTTPs traffic. Some connections from these Gozi-ISFB Pcaps were placed in the behavioral clusters that we failed to identify (c13-c18).

The node location for some malware families is intriguing. For example, most of the Zeus-VM-AES Pcaps that are associated with malicious subnets are located in the G3 group, together with Ramnit files that are associated with port scans. Dridex-Loader is only observed in group G4, while most of the Citadel Pcaps are also seen in the same. Blackmoon and Gozi-ISFB have Pcaps that are distributed over all of the behavioral sub-groups. However, Gozi-ISFB is seen dominating the G1 group, while Backmoon dominates the G4 group. Furthermore, as observed from Table 4, Gozi-ISFB's Pcaps collectively show 18 discrete behaviors and Citadel's Pcaps show 11 behaviors. However, Citadel shows more discrete behaviors in a single Pcap compared to Gozi-ISFB, as Gozi-ISFB's Pcaps contain more (behaviorally similar) connections on average. Also, each of Gozi-ISFB's Pcaps is more behaviorally dissimilar than Citadel's Pcaps.

Zeus-Panda's Pcaps are clearly divided into two behavioral sub-groups—one in G2 group with Blackmoon samples and the other in the G4 group. Zeus-v1, Zeus-P2P, ZeuS, Gozi-EQ, and Dridex are only seen at the left side of the graph, indicating that none of their distinguishing behaviors were present in the dataset.

To conclude, the DAG clearly identifies the discrepancies in the malware's behavioral profiles and their traditional family names. A significant portion of the analysis pipeline is automated and unsupervised. The temporal heatmaps together with the DAG are intended for human-in-the-loop exploration—they actively support malware behavior analysis and provide more insightful characterization of malware than current family labels.

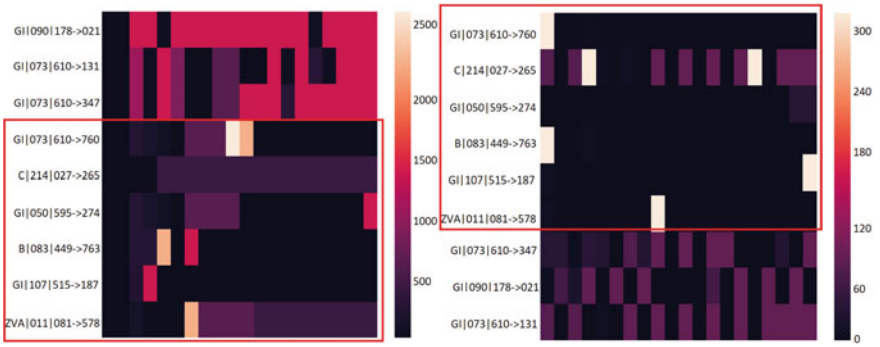
7.2 Comparison with Statistical Features

Baseline Setup. We compare the cluster quality of using sequential versus statistical features. We use the existing method by Tegeler et al. [54] (called baseline, henceforth) to compare our results since they not only use statistical features, but also incorporate periodic behavior using Fourier transform to detect bot-infected network traffic. Although the goal of their study diverges from ours, their feature selection approach is aligned with ours. For objectivity, we keep the rest of the pipeline as explained in Sect. 4. Taking guidelines from Tegeler et al. [54] and adapting them to our problem statement, each connection in the baseline is characterized by (1) average packet size, (2) average interval between packets, (3) average duration of a connection, and (4) the maximum Power Spectral Density (PSD) of the FFT obtained by the binary sampling approach by Tegeler et al. [54]—the signal is 1 when a packet is present in the connection and is 0 in between.

Cluster quality comparison. The baseline method results in 22 clusters, with an average of 21.2 connections per cluster. 265 connections are discarded as noise. These results are in comparison with sequence clustering—18 clusters; on average 25 connections per cluster; 284 connections discarded as noise.

Baseline seems to perform better with smaller cluster size on average and discarding fewer connections as noise. However, a deeper analysis shows the obtained clusters lack quality.

1. With statistical features, connections present in most clusters appear very different from their fellow connections. On average, 57.5% connections per cluster have visually different temporal heatmaps, compared to 8.3% for sequential features. Figure 7 shows a cluster from the baseline. It has nine connections, out of which six are errors based on their behavior. The *rightful owners* of the cluster are the connections that have the least mutual distance, i.e., $\texttt{GI|090|178} \rightarrow \texttt{021}, \texttt{GI|073|610} \rightarrow \texttt{131}, \texttt{GI|073|610} \rightarrow \texttt{346}$. The other six connections have minor differences in all features, except the source port which is 6 for all. They were clustered together because their statistical features had the least mutual distance, i.e., $average\_time\_interval = 19.77 \pm 3.11$; $fft = 0.07 \pm 0.05$; $average\_duration = 397.7 \pm 61.7$; $average\_bytes = 573.3 \pm 113.8$. The temporal heatmaps clearly show behavioral differences in nearly all clusters.
2. Statistical features are also unable to identify the direction of network traffic. In the cluster shown in Fig. 7, there is one incoming connection in the cluster along with eight outgoing ones. A similar trend is observed for 19 out of 22 clusters. In contrast, sequences of packet size and inter-arrival time are enough to identify traffic direction in sequence clustering.



(a) Packet sizes (b) Interval

Fig. 7 Baseline clusters: Six out of nine behaviorally different connections clustered together in baseline version

In summary, while statistical features may be simple to use, they lose behavioral information that plays a crucial role in accurately determining similarities in network behavior. Sequence clustering obtains significantly better clusters. Given that modeling behavioral profiles is already challenging for short sequences, it is remarkable that MalPaCA can identify network behaviors using only 20 packets and 4 coarse features.

8 Limitations and Future Work

Limitations. Performance optimizations are needed to make sequence clustering more efficient and scalable. In MalPaCA, DTW forms the main bottleneck as the length of sequences grows longer. There exist streaming versions of DTW that compute results in real time. One such technique is presented by Oregi et al. [42]. Moreover, using Locality Sensitive Hashing [6, 7] can make MalPaCA more scalable.

Density-based clustering discards rare events as noise. This makes sense if the dataset is noisy. However, in the presence of a purely malicious dataset, the connections that lie in lower density regions may represent rare attacking capabilities, which may be discarded in the current implementation.

Malware authors can try to evade detection by modifying malware's code. A common assumption is that malware can easily evade detection by adding random delays and padding to packets. However, there is a limit to what an attacker can change. For example, a TCP handshake needs to happen in a certain way because this is how the protocol dictates it. Also, padding-related provisions are already standardized by some commonly used protocols, such as TLS making it difficult to hide "coarse" features like packet sizes and inter-arrival times [19]. We expect that MalPaCA is evasion resilient, e.g., since MalPaCA only uses coarse features,

evading it is not a trivial task. Moreover, the usage of Dynamic Time Warping distance makes it resilient to random delays [20] and due to the relative distance measures used in HDBScan, randomized port numbers are already clustered together, as shown in Sect. 6. If, after all this, attackers still manage to evade MalPaCA, the malware sample will end up with a new behavioral profile, making analysts more prone to analyze it. More study is needed to strengthen these claims.

Future work. There are several research directions this work can take: (i) We will work on fully automating the capability assessment of malware by building a directory of observed behaviors, which will be used for cluster labeling. (ii) We will test and improve MalPaCA's adversarial evasion resilience. (iii) We will integrate additional behavioral data sources in MalPaCA so the profiles are based on all static, system-level, and network behavior. (iv) Since MalPaCA is a generic technique, we will test its applicability in building behavioral profiles for everyday-use software.

9 Conclusions

In this chapter, we propose MalPaCA, an intuitive network traffic-based tool to perform malware capability assessment: It groups capabilities using sequence clustering and uses the cluster membership to build network behavioral profiles. We also propose a visualization-based cluster evaluation method whose key advantage is its white-box nature, allowing malware analysts to investigate, understand, and even correct labels, if necessary. We implement MalPaCA and evaluate it on real-world financial malware samples collected in the wild. MalPaCA independently identifies attacking capabilities. We build a DAG to show overlapping malware behaviors and discover a number of samples that do not adhere to their family names, either because of incorrect labeling by black-box solutions or extensive overlap in the families' behavior. We also show that sequence clustering outperforms existing statistical features-based methods by making only 8.3% errors, as opposed to 57.5%. MalPaCA, with its visualizations and capability assessment, can actively support the understanding of malware samples. The resulting behavioral profiles give malware researchers a more informative and actionable characterization of malware than current family designations.

References

1. Acar, Abbas, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and A. Selcuk Uluagac. 2018. Peek-a-boo: I see your smart home activities, even encrypted! *arXiv*.
2. Aiolli, Fabio, Mauro Conti, Ankit Gangwal, and Mirko Polato. 2019. Mind your wallet's privacy: Identifying bitcoin wallet apps and user's actions through network traffic analysis. In *SIGAPP*, 1484–1491. ACM.

3. Anderson, Blake, and David McGrew. 2017. Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD*, 1723–1732.
4. Anderson, Blake, Subharthi Paul, and David McGrew. 2017. Deciphering malware's use of TLS (without decryption). *CVHT Journal* 14 (3).
5. Azab, Ahmad, Mamoun Alazab, and Mahdi Aiash. 2016. Machine learning based botnet identification traffic. In *IEEE Trustcom/BigDataSE/ISPA*, 1788–1794. IEEE.
6. Azab, Ahmad Robert Layton, Mamoun Alazab, and Jonathan Oliver. 2014. Mining malware to detect variants. In *Cybercrime and trustworthy computing conference*, 44–53. IEEE.
7. Bayer, Ulrich, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, behavior-based malware clustering. In *NDSS*, vol. 9, 8–11. Citeseer.
8. Baysa, Donabelle, Richard M. Low, and Mark Stamp. 2013. Structural entropy and metamorphic malware. *CVHT Journal* 9 (4): 179–192.
9. Berndt, Donald J., and James Clifford. 1994. Using dynamic time warping to find patterns in time series. *KDD* 10: 359–370
10. Bilge, Leyla, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *ACSAC*, 129–138. ACM.
11. Black, Paul, Iqbal Gondal, and Robert Layton. 2017. A survey of similarities in banking malware behaviours. *Computers and Security*.
12. Campello, Ricardo J.G.B., Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *PAKDD*, 160–172. Springer
13. Canfora, Gerardo, Andrea De Lorenzo, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. 2015. Effectiveness of opcode ngrams for detection of multi family android malware. In *ARES*, 333–340. IEEE.
14. Cavallaro, Lorenzo, Christopher Kruegel, Giovanni Vigna, Fang Yu, Muath Alkhalaf, Tevfik Bultan, Lili Cao, Lei Yang, Heather Zheng, Christopher C. Cipriano, et al. 2009. Mining the network behavior of bots. Technical report 2009-12.
15. Chakkaravarthy, S. Sibi, D. Sangeetha, and V. Vaidehi. 2019. A survey on malware analysis and mitigation techniques. *Computer Science Review* 32: 1–23.
16. Chan, Neil Wong Hon, and Shanchieh Jay Yang. 2017. Scanner: Sequence clustering of android resource accesses. In *IEEE DSC 2017*.
17. Conti, Mauro, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *CODASPY*, 297–304. ACM.
18. Davies, David L. and Donald W. Bouldin. 1979. A cluster separation measure. In *TPAMI 1979*.
19. Dyer, Kevin P., Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *S&P*, 332–346. IEEE.
20. Elfeky, Mohamed G., Walid G. Aref, and Ahmed K. Elmagarmid. 2005. Warp: Time warping for periodicity detection. In *Data Mining*, 8–pp. IEEE.
21. Feng, Yu, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of android malware through static analysis. In *SIGSOFT*, 576–587. ACM.
22. Gandotra, Ekta, Divya Bansal, and Sanjeev Sofat. 2014. Malware analysis and classification: A survey. *Information Security Journal* 5 (02): 56.
23. Garcia, Sebastian. 2015. Modelling the network behaviour of malware to block malicious patterns. the stratosphere project: A behavioural IPS. *VB*.
24. Garcia-Teodoro, Pedro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security* 28 (1–2): 18–28.
25. Ghafir, Ibrahim and Vaclav Prenosil. 2015. Blacklist-based malicious IP traffic detection. In *GCCT*, 229–233. IEEE.
26. Ghorbani, Ali A., and Saeed Nari. 2013. Automated malware classification based on network behavior. In *ICNC*, 642–647. IEEE.

27. Hammerschmidt, Christian, Samuel Marchal, Radu State, and Sicco Verwer. 2016. Behavioral clustering of non-stationary IP flow record data. In *CNSM*, 297–301. IEEE.
28. Kalgutkar, Vaibhavi, Natalia Stakhanova, Paul Cook, and Alina Matyukhina. 2018. Android authorship attribution through string analysis. In *ARES*, 4. ACM.
29. Kantchelian, Alex, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J Doug Tygar. 2015. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *AISec*.
30. Kim, Ki-Hyeon and Mi-Jung Choi. 2015. Android malware detection using multivariate time-series technique. In *APNOMS*, 198–202.
31. Kovacs-Vajna, Zsolt Miklos. 2000. A fingerprint verification system based on triangular matching and dynamic time warping. *TPAMI* 22 (11): 1266–1276.
32. Lee, Jehyun, and Heejo Lee. 2014. Gmad: Graph-based malware activity detection by DNS traffic analysis. *Computer Communications* 49.
33. Li, Peng, Limin Liu, Debin Gao, and Michael K. Reiter. 2010. On challenges in evaluating malware clustering. In *RAID*, 238–255. Springer.
34. Li, Wei-Jen, Ke Wang, Salvatore J. Stolfo, and Benjamin Herzog. 2005. Fileprints: Identifying file types by n-gram analysis. In *IEEE SMC information assurance workshop*, 64–71. IEEE.
35. Li, Yuping, Jiyong Jang, Xin Hu, and Xinming Ou. 2017. Android malware clustering through malicious payload mining. In *RAID*, 192–214. Springer.
36. Lin, Qin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. 2018. Tabor: a graphical model-based approach for anomaly detection in industrial control systems. In *Asia CCS*, 525–536. ACM.
37. Maggi, Federico, Andrea Bellini, Guido Salvaneschi, and Stefano Zanero. 2011. Finding non-trivial malware naming inconsistencies. In *ICISS*, 144–159
38. Mohaisen, Aziz, Omar Alrawi, Matt Larson, and Danny McPherson. 2013. Towards a methodical evaluation of antivirus scans and labels. In *ISA workshop*, 231–241. Springer.
39. Mohaisen, Aziz, Omar Alrawi, and Manar Mohaisen. 2015. Amal: High-fidelity, behavior-based automated malware analysis and classification. *Computers and Security* 52.
40. Moubarak, Joanna, Maroun Chamoun, and Eric Filiol. 2017. Comparative study of recent mea malware phylogeny. In *ICCCS*, 16–20. IEEE.
41. Ntlangu, Mbulelo Brenwen, and Alireza Baghai-Wadji. 2017. Modelling network traffic using time series analysis: A review. In *IoTBDS*, 209–215.
42. Oregi, Izaskun, Aritz Pérez, Javier Del Ser, and José A Lozano. 2017. On-line dynamic time warping for streaming time series. In *ECML-PKDD*, 591–605. Springer.
43. Pellegrino, Gaetano, Qin Lin, Christian Hammerschmidt, and Sicco Verwer. 2017. Learning behavioral fingerprints from netflows using timed automata. In *IFIP*, 308–316. IEEE.
44. Perdisci, Roberto, Wenke Lee, and Nick Feamster. 2010. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, vol. 10.
45. Pomorova, Oksana, Oleg Savenko, Sergii Lysenko, Andrii Kryshchuk, and Kira Bobrovnikova. 2015. A technique for the botnet detection based on DNS-traffic analysis. In *CN*, 127–138. Springer.
46. Rafique, M. Zubair, and Juan Caballero. 2013. Firma: Malware clustering and network signature generation with mixed network behaviors. In *RAID*, 144–163. Springer.
47. Rieck, Konrad, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19 (4): 639–668.
48. Rousseeuw, Peter J. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *CAM Journal* 20.
49. Sebastián, Marcos, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A tool for massive malware labeling. In *RAID*, 230–253. Springer.
50. Sharma, Arushi, Ekta Gandotra, Divya Bansal, and Deepak Gupta. 2019. Malware capability assessment using fuzzy logic. *Cybernetics and Systems* 1–16.
51. Suarez-Tangil, Guillermo, Juan E. Tapiador, Pedro Peris-Lopez, and Jorge Blasco. 2014. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications* 41 (4).

52. Sun, Mingshen, Xiaolei Li, John C.S. Lui, Richard T.B. Ma, and Zhenkai Liang. 2017. Monet: a user-oriented behavior-based malware variants detection system for android. *TIFS* 12 (5).
53. Tajalizadehkhoob, S.T., Hadi Asghari, Carlos Gañán, and M.J.G. Van Eeten. 2014. Why them? extracting intelligence about target selection from zeus financial malware. In *WEIS*.
54. Tegeler, Florian, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. Botfinder: Finding bots in network traffic without deep packet inspection. In *CoNEXT*, 349–360. ACM.
55. Tian, Ronghua, Lynn Batten, Rafiqul Islam, and Steve Versteeg. 2009. An automated classification system based on the strings of trojan and virus families. In *MALWARE*. IEEE.
56. Verwer, Sicco, Rémi Eyraud, and Colin De La Higuera. 2014. Pautomac: A probabilistic automata and hidden Markov models learning competition. *Machine Learning* 96 (1–2): 129–154.
57. Vinod, P., V. Laxmi, M.S. Gaur, and Grijesh Chauhan. 2012. Momentum: Metamorphic malware exploration techniques using MSA signatures. In *IIT*, 232–237. IEEE.
58. Volis, George, Christos Makris, and Andreas Kanavos. 2016. Two novel techniques for space compaction on biological sequences. *WEBIST*.
59. Wang, An, Aziz Mohaisen, Wentao Chang, and Songqing Chen. 2015. Capturing DDoS attack dynamics behind the scenes. In *DIMVA*, 205–215. Springer.
60. Wang, Wei, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. 2017. Malware traffic classification using convolutional neural network for representation learning. In *ICOIN*, 712–717.
61. Wang, Yipeng, Zhibin Zhang, Danfeng Daphne Yao, Buyun Qu, and Li Guo. 2011. Inferring protocol state machine from network traces: a probabilistic approach. In *ACNS*, 1–18. Springer.
62. Yadav, Tarun and Arvind Mallari Rao. 2015. Technical aspects of cyber kill chain. In *SSCC*.
63. Zahrotun, Lisna. 2016. Comparison jaccard similarity, cosine similarity and combined both of the data clustering with shared nearest neighbor method. *CE&AJ* 5 (1): 11–18.