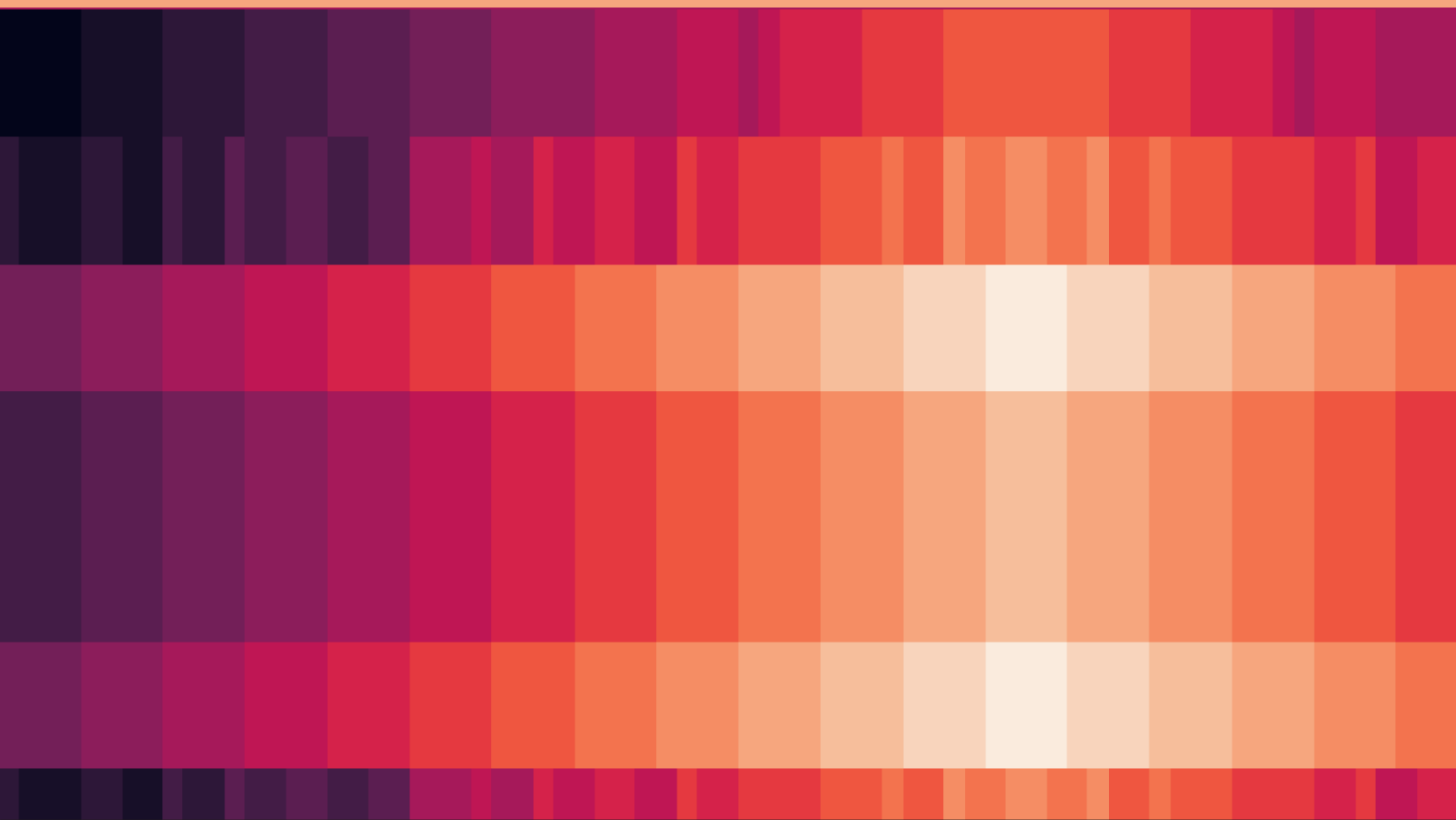
Clustering Malware's Network Behavior using Simple Sequential Features

Azqa Nadeem

Delft University of Technology,
Faculty of EEMCS,
Group: Cyber Security



TU Delft Delft University of Technology

Challenge the future

Clustering Malware's Network Behavior using Simple Sequential Features

by

Azqa Nadeem

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Track: Data Science & Technology
Specialization: Cyber Security

at the Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
to be defended publicly on Monday September 10, 2018 at 10:00 AM.

Student number:	4542606	
Project duration:	November 22, 2017 – September 10, 2018	
Supervisor:	Dr. ir. S. E. Verwer	
Thesis committee:	Prof. dr. P. H. Hartel,	EEMCS, TU Delft
	Dr. ir. C. Hernandez Ganan,	TPM, TU Delft
	Dr. Zaid Al-Ars,	EEMCS, TU Delft

TU Delft Delft University of Technology

ABSTRACT

Developing malware variants is extremely cheap for attackers because of the availability of various obfuscation tools. These variants can be grouped in malware families, based on information retrieved from their static and dynamic analysis. Dynamic, network-level analysis of malware shows its core behavior since it captures the interaction with its developer. Moreover, increasingly more emphasis is given to using Deep Packet Inspection (DPI) in order to cluster malware's network behavior. However, DPI has severe privacy implications, as it involves inspecting payloads of the network traffic.

This report presents an exploratory study, the aim of which is to characterize and cluster malware behavior using high-level, non-privacy-invasive, sequential features extracted from its network activity. The key intuition behind the proposed solution is that if the underlying infrastructure of distinct malware samples is similar, the order in which they perform certain actions should also be similar. The results of this research show that sequence clustering allows flexible and robust clusters, as opposed to using non-sequential features. In addition, incoming and outgoing connections are clustered separately even though IP address is not used as a feature. The clusters themselves reveal interesting attacking capabilities, such as port scans, potentially malicious subnets, and the same Command and Control server responding to different malware families. Lastly, a comparison with clusters obtained from static analysis reveals that dynamic, network-based clustering is far more qualified to determine the many behaviors exhibited by a single malware family, as well as behaviors common across multiple malware families.

PREFACE

I came to TU Delft to pursue a Master degree from a renowned university knowing that it will change the course of my life. The excitement of learning new things every day quickly overcame the realization of being the only female student from my country enrolled in the degree I was pursuing. I had accepted my fate as a trailblazer in many endeavors I would choose to do in the future.

The initial choice of my Master thesis reflected immense ambition in that I wanted to put an end to crime once and for all. As I continued my research, it humbled me by teaching me the importance of baby steps. Hence, this thesis is the first step of many that I intend to take in my journey as a researcher. This report is a result of my over-ambitious ideas, hard work, and many sleepless nights accumulated for the past ten months. It goes without saying that I would not have been able to traverse my way through to the finish line if it wasn't for the intellectual and emotional guidance provided by my supervisors, peers, and friends.

I want to begin the thanking spree by thanking, first and foremost, Allah ta'ala, for giving me the opportunity to pursue my dreams; for bestowing upon me the courage to ask and find answers for the difficult questions; and for being there in times I have needed Him the most.

I am indebted to my supervisors: Sicco Verwer for his constant guidance and for providing invaluable feedback, on things not just related to my thesis. Thank you for believing in me and making me see the positive side of things. I also want to thank Carlos Hernandez Ganan for his support, invaluable feedback, and the hours he took out of his own time to help me with the project. If it weren't for you, I'd probably still be waiting for the dataset. Furthermore, I want to thank Pieter Hartel and Zaid Al-Ars for being on my thesis committee and for taking out the time to read my report.

Many thanks to the TU Delft for deeming me worthy of the Justus & Louise van Effen Excellence Scholarship. I wouldn't have been able to pursue my Master without it.

I also want to express my gratitude towards my parents for their unwavering support in everything I decide to do, and for always standing firm by my side. Last, but not least, I want to thank Junaid for being my family away from home, and for truly being my better-half during this roller coaster journey.

Azqa Nadeem
Delft, September 2018

Contents

1

INTRODUCTION

The number of malware attacks are dramatically increasing and are also becoming more elusive, adaptive and powerful [1]. Malware is one of the leading threats in cybersecurity today, and its growth is still on the rise. According to PandaLabs report, in Q3 of 2016 alone, 18 million new malware samples have been detected [1]. In Q1 of 2017, a new malware specimen emerged every 4.2 seconds [2], and this number is predicted to keep growing [3].

A significant fraction of the malware samples that we see on the day-to-day basis is similar to each other, either in terms of target selection, behavior, or the Command and Control (C&C) server they report to. Available literature suggests that the malware developers follow an iterative approach towards malware development, as opposed to prototyping each variant separately [4]. So, the similarities in these variants can be used to categorize malware samples into *malware families*. Grouping malware into families will allow us to analyze only the unique samples, reducing the number of samples to be analyzed drastically.

In addition, several researchers have also shown that malware developers reuse code either from old malware variants or other malware families [5]. This often happens as a result of a cat and mouse game where small improvements in a target's system push the attackers to include additional malicious code while keeping the old code intact with the aim of maximizing their chances of success. For example, Sun *et al.* [6] report that to circumvent detection and to deploy malware quickly, hackers usually do not develop new malware from scratch, but rather improve existing logic or add new malicious logic into existing malware. The findings of Li *et al.* [5] and Sun *et al.* [6] suggest that malware developers often work in groups, where they either collaborate using forums or share their code publicly for others to use. Therefore, we expect that similar behavior depicted by samples of different malware families will uncover unknown malware author collaborations. Moreover, samples that report to the same Command and Control server but show different attack vectors may suggest the presence of a single attacker (or a single group of attackers) controlling those samples. Leveraging these insights will lead to the development of proactive and specialized defense tactics against multiple malware families that behave similarly.

Different techniques exist to cluster malware samples together depending on which 'attack vectors' are considered. The earliest technique to dissect a malware is static analysis, which involves decompiling the malicious binary and understanding the code to make sense of what the attack-

[1] https://www.pandasecurity.com/mediacenter/pandalabs/pandalabs-q3/

ers are after. Since the introduction of obfuscation techniques, such as encryption and encoding, it has become more and more challenging to decompile the binary and to understand the malicious code [7]. Although analyzing the decompiled source code provides an overall picture of what the malware is capable of, not all the features present in the source code are utilized when the malware is executed [8]. Hence, static analysis is not the best technique when one wants to analyze the actual behavior exhibited by the malware. Dynamic analysis, on the other hand, involves executing a malware in a sandboxed environment and analyzing the actions it performs. This technique makes the code obfuscation problem almost irrelevant because even if the code is obfuscated, the malware still behaves the way it is programmed to. Therefore, dynamic analysis is the current state-of-the-art mechanism for analyzing malware, though static analysis is also used in a few cases.

There are two sources of malware behavior that can be used to analyze it: system-level behavior (system calls and API usage) and network-level behavior (what data is sent or received over the network). System-level behavioral analysis can become prohibitively expensive given the different system architectures often present within a network and the several low-level operations that a malware performs, which may or may not be related to its final objective. Moreover, extracting such behavioral information from each system in a network is extremely tedious and may create privacy and compatibility issues. On the other hand, network-level activities show the core behavior of malware, since it is used to communicate with the attacker directly. It shows exactly what information is exfiltrated (in case of spyware or a banking malware) and precisely what commands are received by the malware. Deploying a system that operates purely on network behavior induces less overhead on the end hosts, and existing infrastructure for network monitoring can be reused. Moreover, increasingly more malware samples are using the Internet to communicate with their Command and Control servers (or their developers), and since the Internet architecture is mostly uniform in the way devices interact with each other, the same solution can be deployed for different organizations. Even though the network traffic is sometimes encrypted, the way a malware communicates, such as the certificates it uses, may reveal distinguishing patterns that can still potentially be used as signatures. Therefore, clustering malware using the dynamic analysis of its network-level behavior is the goal of this research.

1.1. MOTIVATING EXAMPLE

Clustering malware based on its network-level behavior is not necessarily better than clustering on system-level behavior. In fact, they are two pieces of the same puzzle that, when combined, provide the exhaustive behavior of a malware sample. However, literature has shown that the malware authors may sometimes divide the tasks over multiple malware samples [9]. These samples may perform different system-level activities but produce similar network-level activity because they are controlled by the same attacker or have used the same botnet kit.

One such example is taken from the work of Perdisci *et al.* [9], as shown in Figure 1.1. The authors demonstrate a case of two distinct malware samples that perform significantly different system-level activities. Due to their difference in behavior, the anti-virus they have used assigns different labels to the samples (TR/Dropper.Gen and DR/PCK.Tdss.A.21). However, the network traffic that the two samples generate is identical (e.g., see the IP address contacted, the bytes transferred, and the sequence in which actions are performed). This suggests that either the malware author for both these samples is the same, or the botnet kit used to develop these malware samples is the same, such that even the Command and Control server is the same. If one were to cluster malware samples such as these purely based on their system-level behavior, they would end up in different clusters. Moreover, the anti-virus labels are assigned heavily based on system-level behavior, so similar clus-

tering would be observed with them too. Figure 1.2 shows another example of two malware samples that are assigned different labels because their decompiled code looks different. However, the figure shows similar sequences of packet sizes received from two different IP addresses (185.195.24.6 for Zeus and 173.224.119.212 for Gozi). This similarity in behavior could either indicate mislabelling of these samples as separate families or that the malware samples share some underlying infrastructure. If one were to cluster these samples based on the hosts they communicated with or the assigned malware family, they would be separated, which is undesirable. On the contrary, Figure 1.3 shows two malware samples that are both labeled as Ramnit but exhibit different behavior in terms of the sequence of packet sizes. This may indicate two different attacking capabilities. If one were to cluster these samples using their assigned family names, they would end up together, which is also undesirable.

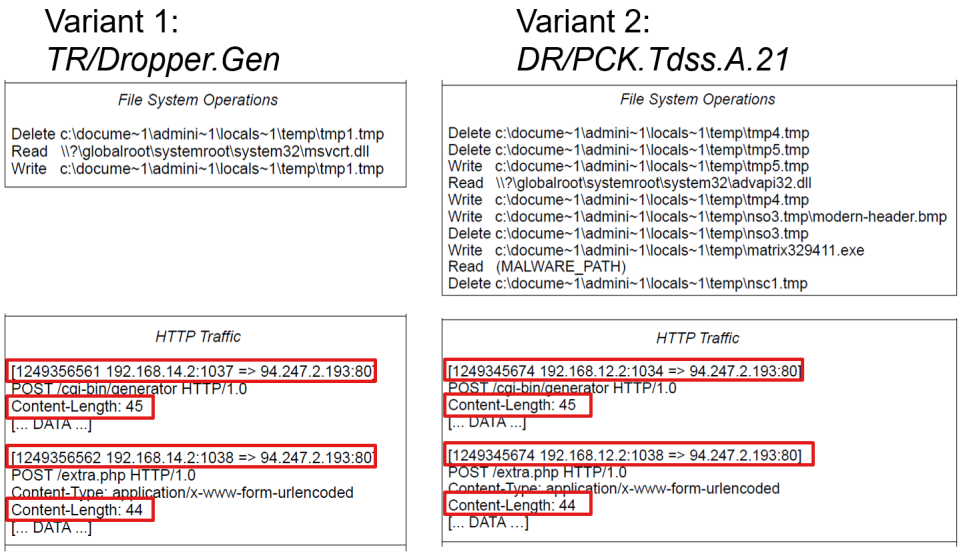


Figure 1.1: Motivating example 1 – two malware samples with different system level activities but identical network behavior

We argue that the network behavior paints a different picture, which is closer to the attacker than system-level behavior since it is the network behavior that captures the interaction with the Command and Control server. Therefore, we see a benefit in clustering malware using their network traffic. We expect that clustering malware samples, such as the ones shown in Figures 1.2 and 1.3, based on their network activity will provide a different way of analyzing similar malware.

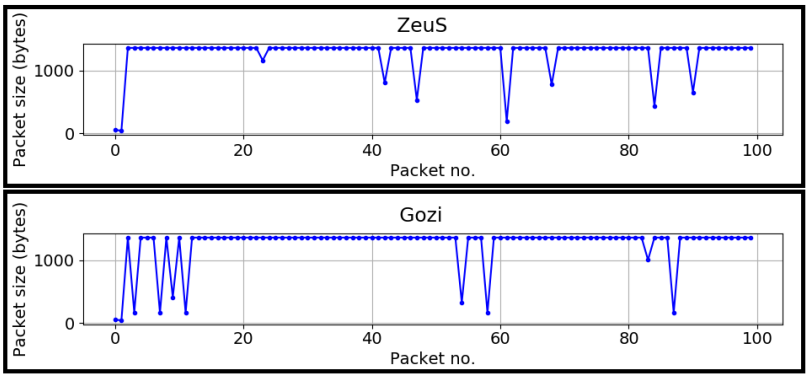


Figure 1.2: Motivating example 2 – two malware samples with similar network activity but different labels

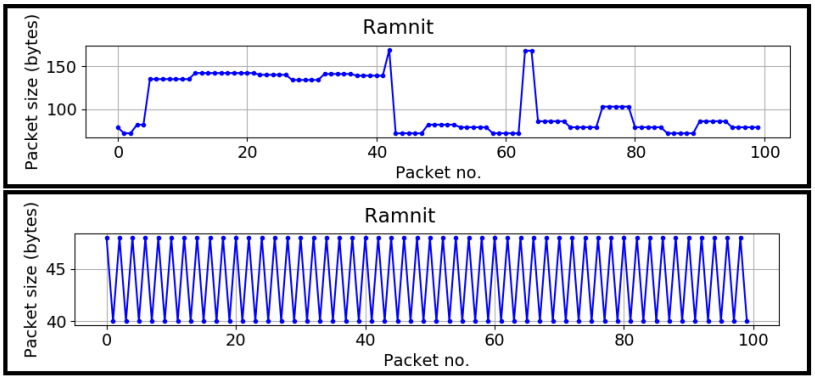


Figure 1.3: Motivating example 3 – two malware samples with different network activity but same label

1.2. PROBLEM STATEMENT

Developing malware variants is extremely cheap for attackers given the many obfuscation tools so readily available. These variants have different hashes that make it difficult for traditional signature-based solutions, such as anti-viruses, to detect them. The state-of-the-art analysis techniques involve clustering malware samples based on similarity to identify whether a malware sample is unique and should be analyzed [10]. Clustering based on network behavior is a relatively novel technique that produces low overhead on end hosts and captures the communication between the infected host and the Command and Control server. There is an increasing interest in *Deep Packet Inspection* (DPI) [11–14], which involves inspecting payloads of the network traffic to detect suspicious content. While DPI is extremely effective in detecting malicious content, it has severe privacy implications because it can easily be misused as a surveillance tool [15]. Besides, because of the increasing privacy awareness among the masses, many countries have adopted data and information privacy protection regulations. For example, the European Union enforced the General Data Protection Regulation (GDPR) in May of 2018. Hence, the privacy concerns associated with DPI do not render it future-proof. Therefore, we are interested in finding solutions for malware detection and clustering that are privacy-sensitive. This would entail developing a technique that does not access the actual content of the packet, but relies on high-level features extracted from packet headers, which do not come under Personally Identifiable Information (PII) [2]. For example, IP address is considered as PII [3], so it cannot be used for this research.

The concise problem statement that is addressed in this research is as follows:

> *Developing a technique to cluster malware's network behavior without using Deep Packet Inspection.*

1.3. DATASET

This project aims to develop a clustering technique that is applicable to the real world. Therefore, real-world data was collected with the help of a security company based in the Netherlands, whose

[2] https://www.csoonline.com/article/3215864/privacy/how-to-protect-personally-identifiable-information-pii-under-gdpr.html
[3] https://www.enterprisetimes.co.uk/2016/10/20/ecj-rules-ip-address-is-pii/

name we cannot disclose. Hence, the company has been referred to as ABC in this report.

The dataset consists of zipped packet capture (Pcap) files, which stores network traces of malware samples that they have detected in 2017. The dataset is composed of only banking malware because ABC's main clientèle are financial institutions. An example of the directory structure of the files is given in Figure 1.4 and what the Pcap files look like is provided in Figure 1.5. More details regarding the actual content of the dataset are given in Chapter 4.

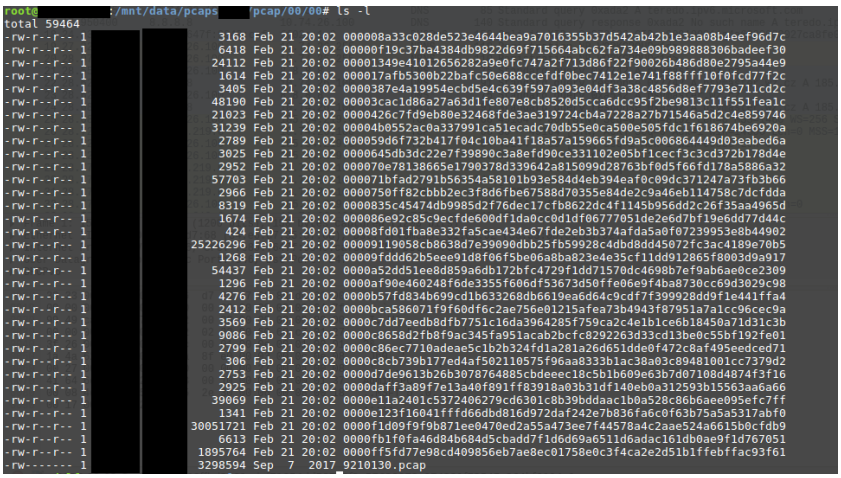


Figure 1.4: Screenshot of the directory structure showing the provided dataset

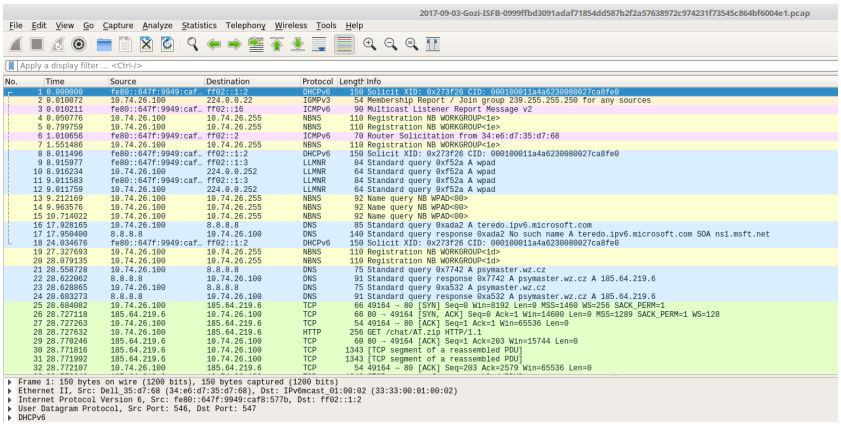


Figure 1.5: An example Pcap file from the provided dataset opened in Wireshark

1.4. PROPOSED SOLUTION

The key intuition behind the proposed solution is that if the underlying infrastructure of distinct malware samples is similar (e.g., as a result of being controlled by the same attacker, or by reusing code from another author), the order in which they perform certain actions would be similar. However, if we were to look at individual data points rather than sequences, we might lose similarity shown by the order of event occurrence. Figure 1.6 summarizes each step of the project flow. The blue boxes show project phases, while the purple ones show the different sub-sections of that phase. We briefly introduce each phase in this section.

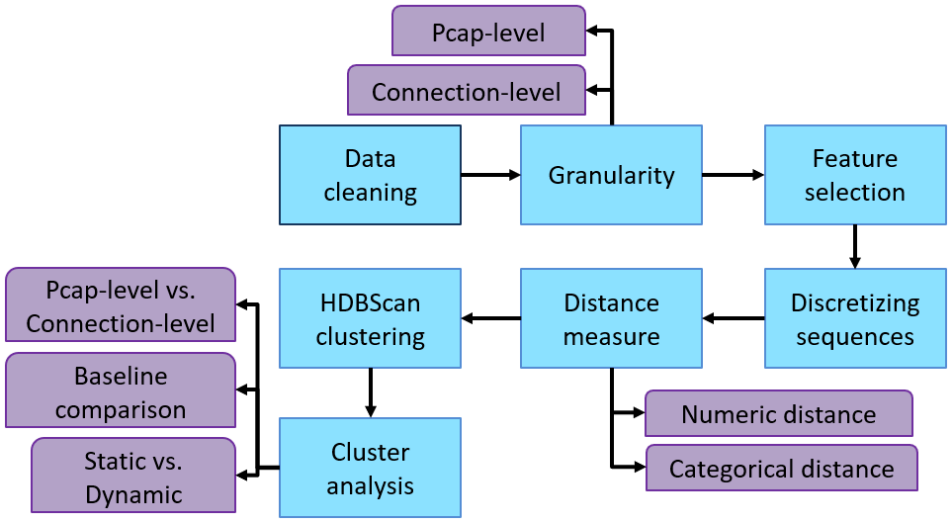
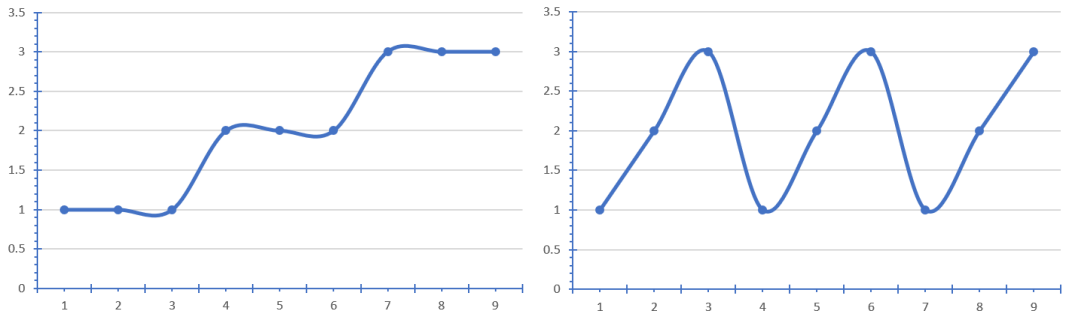


Figure 1.6: Flowchart showing the various project phases

1.4.1. SEQUENCE CLUSTERING

A sequence is an ordered list of items. The order captures temporal information, which is useful for behavioral modeling. On the other hand, a single data point captures only the summary of the dataset. Consider an example of a computer program that randomly generates a list of 9 numbers from 1 to 3, and we want to model the different behaviors exhibited by this program. Suppose that it generates the following two sequences: [1,2,3,1,2,3,1,2,3] and [1,1,1,2,2,2,3,3,3]. A visualization of these sequences is given in Figure 1.7. It is evident that the behavior exhibited by the program was categorically different when it generated the two sequences. The aggregate (average) of both these sequences is 2. If we were to model the program using a single data point, it would incorrectly group these two behaviors.



(a) Seq A: 1,1,1,2,2,2,3,3,3

(b) Seq B: 1,2,3,1,2,3,1,2,3

Figure 1.7: Motivating example for Sequence clustering

In this project, the aim is to model a Pcap file in a way that captures its behavior. Hence, based on our intuition, using sequences to model features instead of a single data point representing a feature is a natural design choice.

1.4.2. GRANULARITY

The level of granularity of a sequence defines which packets are used in the generation of a sequence. For Pcap files, a design choice is to identify whether the whole Pcap file should be considered as one long sequence, should it be broken down to incoming and outgoing traffic sequences, or should it be broken further down to connection level sequences. A *connection*, in this context, is defined as the traffic sent from source IP to destination IP address. This means 8.8.8.8 → 123.123.123.123 will be a different connection than 123.123.123.123 → 8.8.8.8. We hypothesize that looking at different granularities will result in different patterns being found in the sequences. Hence, we consider two levels of granularity for the sake of this project:

1) **Pcap-level granularity:** A feature X of a malware is generated at the Pcap-level by concatenating each value of the feature X that occurs in the Pcap.

2) **Connection-level granularity:** A feature X of a connection from A to B is generated at the Connection-level by concatenating only the values of X that occur during the connection $A \rightarrow B$.

1.4.3. FEATURE SELECTION AND DISTANCE MEASURE

Feature selection is a crucial step towards modeling a malware's network behavior. As mentioned before, our intuition is that malware related to each other will perform actions in a similar order. We hypothesize that the similarity in the sequence of actions will also be reflected in high-level features, such as packet sizes or the interval between sending or receiving the next packet. Therefore, the feature set used is a number of sequences of simple, high-level features, extracted from the headers of Network and Transport layer of the OSI model [4]. The feature set selected for each level of granularity is different.

The features are either categorical (a.k.a nominal) or numerical (a.k.a ratio) sequences in nature. In order to compute the distance between two sequences, multiple distance measures for each type of sequence are experimented with. For numerical sequences, Dynamic Time Warping, and Euclidean (point-to-point) distance measure are considered. For categorical sequences, Sequence alignment, Longest Common Subsequence, and Ngram analysis are considered. Each of these methods are explained in Chapter 3 Section 3.2 in detail.

1.4.4. DISCRETIZATION

Sequence discretization is a technique used to convert continuous sequences into discrete ones to limit the number of values a sequence can be represented with. Discretizing a sequence helps suppress noise and represent the sequence in a customized way. However, the real challenge is defining where the thresholds exist that map the categories to the numerical data values. To this end, we introduce three discretization techniques that define the thresholds:

1) **Local Percentile**: The thresholds are selected using percentiles calculated from the sequence under consideration.

2) **Peak Analysis:** The thresholds are selected using a frequency analysis of the sequence under consideration, and taking the average between the most frequent values.

3) **Global Set-Percentile:** The thresholds are selected by 1) doing an initial pass over the whole

[4] https://www.webopedia.com/quick_ref/OSI_Layers.asp

dataset (each packet of every Pcap), 2) selecting only the unique values across the entire dataset, 3) and then using percentiles calculated from the global set of values.

For each level of granularity, we conduct a study on which discretization technique is best suited, and whether sequences should be discretized at all.

1.4.5. CLUSTERING AND CLUSTER VALIDATION

Clustering is a well-studied field in Machine Learning. Many different clustering algorithms exist for different goals and data types. Several limitations, such as the presence of categorical and numerical features and the need for robust clusters filtered the clustering algorithms that were applicable to our case. Two algorithms that are used most commonly in a problem similar to ours were considered: Single-linkage hierarchical clustering and HDBScan clustering. The explanation of these algorithms is present in Chapter 3, Section 3.3. Each level of granularity was clustered separately, with its own set of parameters.

A manual cluster validation approach was utilized because of the absence of any applicable metrics. Clusters were visualized using heatmaps and scatter plots. A false positive analysis was performed to identify Pcaps/connections that were too different from the rest of the Pcaps/connections in that cluster.

A NOTE ON CLUSTERING VERSUS CLASSIFICATION

Clustering and classification are two machine learning problems that may seem similar but are very different. Classification is a supervised learning problem, where a mathematical model is fitted on the dataset under consideration. What makes it a supervised learning problem is the presence of ground truth data labels that can be used to classify future samples in one of the available categories. The accuracy of a method can be checked by counting the number of times the model assigns the same label to a future sample as the ground truth label. On the other hand, clustering is an unsupervised learning method. This technique also categorizes the dataset into groups, but their labels are not known beforehand, because of the absence of ground truth labels. Therefore, the result of this technique is items of the dataset grouped based on some similar aspects (while being different from other groups at the same time), without the prior knowledge of what these groups actually mean. Hence, although there exist metrics to quantify cluster quality, there is no straightforward way to assess the accuracy of the resultant clusters.

In this project, we are developing a clustering technique as opposed to a classification technique because of the absence of reliable ground truth labels. Each Pcap file that we have is assigned a label, which is one of many malware families. However, they are primarily assigned after the static analysis of the malware samples' binaries, which may not always translate to the same categorization in the dynamic behavior shown by the malware sample. Therefore, the clustering also serves as a test regarding this assumption – whether the categorization done using static analysis is the same as that done using dynamic analysis of malware samples.

1.4.6. CLUSTER ANALYSIS

Three types of cluster analysis are performed on the resulting clusters:

1) **Pcap-level versus Connection-level clustering:** The traffic of each cluster is analyzed to understand the kind of behaviors that map to the resulting clusters. This analysis is performed for each

level of granularity. Based on the results, recommendations are presented regarding which level of granularity to use in order to cluster malware's network behavior.

2) **Static versus Dynamic analysis:** As mentioned earlier, the family labels associated to each malware sample were assigned using static analysis of the binary executable. Since we did not have the expertise or the resources to conduct a clustering based on static analysis of the executables ourselves, the family labels themselves were considered as clusters resulting from static analysis – Each malware family is a distinct cluster and all Pcaps/connections associated to it are considered as part of the cluster. On the other hand, the clusters extracted from our technique are considered as dynamic analysis clusters. For both levels of granularity, the location of each Pcap/connection is compared between static-analysis and dynamic-analysis clusters.

3) **Comparison with Baseline:** To the best of our knowledge, there exists no baseline with which the results of this project can be compared directly. Hence, a baseline is constructed from scratch, where instead of representing a feature with a sequence, an aggregate of the sequence is used instead. In particular, we represent a feature with the average of its corresponding sequence. Every other setting is kept constant, in order to emphasize the significance of using sequence-as-features. This analysis is performed at both levels of granularity. The resulting baseline clusters are then compared with sequence-as-features clusters. In particular, each Pcap/connection in sequence-as-features clusters is compared to its new location in the baseline clusters.

1.5. RESEARCH QUESTIONS

This report presents an exploratory study to see whether it is even possible to characterize and cluster malware behavior using high-level non-privacy-invasive, sequential features extracted from its network activity. Hence, the main research question that we aim to answer in this research, and that reflects the problem statement is:

> *RQ: Are high-level sequential features effective in characterizing and clustering malware families' network behavior?*

The first problem that needs to be addressed when using network traffic as the source of data is the granularity at which to construct sequences. In this project, we consider two levels of granularity: Pcap-level and Connection-level. Hence, the first research question answers which level of granularity best characterizes malware's network behavior, and which behaviors are captured in the considered granularities.

> *RQ1: What level of granularity is best to characterize malware families' network behavior?*

Once the granularity has been decided, the next question is: which features characterize the malware sample's behavior? Since the focus is on using sequences, the results from available literature (see Chapter 2) may not be directly applicable. Moreover, the features need to be privacy-preserving. Therefore, header information available at different layers of the OSI model is explored to find the sweet-spot between behavior-characterization and privacy. In addition, the next part of the research question quantifies the benefit of using sequences instead of using singular data points by comparing the clusters resulting from using sequence-as-features, versus averaging those sequences into singular data points.

RQ2 (a): What sequences of high-level feature-set characterize a malware's network traffic?

RQ2 (b): What is the difference between using features that are represented by sequences versus those that are not?

The features considered may have different data types. For example, packet sizes are numerical, while port numbers are categorical in nature. Since the features are represented as sequences, there are either numerical sequences or categorical sequences. The data type of a feature would dictate the distance metric used. Hence, the next research question focuses on identifying the best method to measure the distance between sequences:

RQ3: Which distance captures the (dis)similarity in sequences at the various granularities considered?

Finally, we are interested in observing the kind of behaviors that the proposed clustering technique is able to capture. Specifically, we want to know whether certain type of attacks are reflected in the network traffic of malware samples, or does this technique completely fail in that we do not see any interesting attack behaviors getting clustered. In addition, the second part of the next research question compares the clustering done with using malware family labels versus that done with features extracted from the network traffic of the malware.

RQ4 (a): What kind of behaviors are visible in the clustered malware samples resulting from the proposed clustering approach?

RQ4 (b): How different is the malware samples' membership to clusters resulting from network analysis versus static analysis?

1.6. RESEARCH SCOPE

This project develops a clustering technique to group similar network behaviors exhibited by malware. A comparison with benign network traffic is outside the scope of this project. In addition, the research presents a clustering algorithm, which means that we assume that the malware samples have already been collected. Hence, malware detection is also outside the scope of this project.

Because of the exploratory nature of this study, there does not exist a baseline method to compare the presented solution with. To the best of our knowledge, sequential features extracted from network traffic have not been utilized to cluster malware families. Existing literature (see Chapter 2) varies so much on the feature set used, which behavior is clustered, and the utilized clustering algorithm itself, that they simply cannot be considered as baselines. Therefore, a synthetic baseline has been constructed, which removes the sequence-aspect of the features. We do not perform a detailed analysis of what behaviors are clustered in the baseline version, but only compare the location of samples in clusters with respect to sequence-as-features clusters. Additionally, this technique was not optimized for performance. Hence, a basic performance evaluation is performed but is not

meant to be compared with other existing clustering techniques.

The scope of this project only covers Pcap-level and Connection-level granularity. Moreover, cluster validation is performed manually because of the absence of an applicable metric. Hence, development of a cluster validation metric, as well as a metric for False positive analysis has been left as future work.

1.7. CONTRIBUTIONS

State-of-the-art dynamic analysis techniques rely heavily on system-level activities, so much so that even the labels assigned to malware families are based on the binary executable information. Network-level behavior, on the other hand, presents a different behavioral aspect of malware families, the benefit of which is often overlooked. In addition, the techniques that do exist for network-behavioral analysis focus increasingly on Deep Packet Inspection, which is a controversial topic because of its privacy implications. The proposed solution in this research is novel because of the special emphasis on using high-level features to cluster malware's network behavior. We compensate the 'abstractness' of the features by also exploiting the order in which they occur. Following are the contributions of this project:

1. **Appropriate level of granularity:** We identify the kind of behaviors that are clustered if we make sequences out of one connection at a time (source IP, destination IP) pair versus making sequences out of the whole Pcap.
2. **Feature-set exploration:** We identify the high-level feature-set that characterizes the malware's network behavior based on the level of granularity of the sequences.
3. **Discretization technique:** We have proposed three discretization techniques to convert numeric sequences into categorical ones if one wishes to use distance measures applicable to categorical sequences.
4. **Distance measure identification:** We identify interpretable distance measures to measure the distance between both, numeric and categorical sequences.
5. **Clustering:** We identify the applicability of HDBScan clustering algorithm to cluster malware samples' network behavior robustly. In addition, we present a systematic way of clustering and analyzing malware samples.
6. **Cluster analysis:** We perform visual cluster analysis to identify the different attacking capabilities of malware families. We also compare the resulting clusters with a baseline that does not utilize sequence-as-features. Furthermore, we perform a comparison of resulting clusters with their corresponding family labels.

1.8. SUMMARY OF RESULTS

Sequence clustering allows flexible and robust clusters, as opposed to using non-sequential features. We found that sequences of simple features, such as packet sizes and port numbers can characterize a malware's network behavior. In addition, Pcap-level and Connection-level granularities were evaluated for malware characterization. We found that at the Connection-level, behavioral patterns are clearer and a lot more information is available to infer about the kind of behaviors captured in the clusters, as opposed to Pcap-level granularity. The resulting clusters show different

malicious Internet-usage behaviors. Incoming and outgoing connections are clustered separately even though IP address is not used as a feature. The clusters themselves reveal interesting attacking capabilities, such as systematic and randomized port scans, potentially malicious subnets, and the same C&C server reporting to different malware families. Lastly, a comparison with clusters obtained from static analysis revealed that dynamic, network-based clustering is far more qualified to determine the many behaviors exhibited by a single malware family, as well as behaviors common among multiple malware families.

1.9. PROJECT WORKFLOW AND REPORT ORGANIZATION

Chapter 2 summarizes the literature that was studied in order to identify the research gap. Chapter 3 explains the necessary background concepts required to understand the rest of the report. Each phase of the proposed methodology is explained in the subsequent chapters. In the data collection phase, a few banking malware families are chosen, and their corresponding Pcaps are collected. This is explained in Chapter 4. Then, the Pcap files are explored, and the feature set best characterizing malware families is chosen. At this step, we also explore the level of granularity at which the sequences of features should be generated. In addition, we also identify the distance measures that best represent the distances between numeric and categorical sequences. These steps are explained in Chapter 5. In the sequence discretization step, we propose three discretization techniques to convert numeric sequences to categorical ones. This is explained in Chapter 6. Once we have the granularity at which the sequences should be constructed, the feature set represented as a set of numeric and categorical sequences, and the distance measure that represents an interpretable distance between sequences, we build an `nxn` pairwise distance matrix, where `n` is the dataset size. In the clustering phase, we provide the pre-computed distance matrix to the clustering algorithm. Then, we measure the cluster quality by visual analysis of the resulting clusters at different parameter settings. This step is explained in Chapter 7. After choosing the optimal parameters, we extract the samples from each cluster and further analyze them to answer the research questions posed in this project. The insights obtained from the cluster analysis are explained in Chapter 8. Next, Chapter 9 discusses the limitations of this project and touches upon the future work. Finally, we conclude in Chapter 10.

2

LITERATURE REVIEW

In this chapter, the relevant literature upon which the research is built is discussed. We also show what the state-of-the-art is and identify the research gap, which our study aims to fill.

There is rich literature on clustering malicious software (malware) based on various attributes. However, because of the highly evolving threat landscape, some research is not applicable anymore, while some aspects have not been explored yet. In this section, we summarize the literature that is most relevant to the problem we are addressing in this project. In particular, we discuss works on clustering/classification malware using symbols obtained from 1) Static analysis (Binary information [4, 16–19], Metadata analysis [20, 21], and 2) Dynamic analysis (System activity [6, 22, 23], Network activity [9, 24–31]).

2.1. STATIC ANALYSIS

The first theme of research that we discuss is clustering or classification of malware using static analysis. There are two problems addressed by the following scientific literature: classifying/clustering malicious and benign samples, and classifying/clustering malware families. The latter problem consists of a dataset that is entirely malicious and directly falls under this project's scope.

2.1.1. BINARY ANALYSIS

In the following papers, the authors perform the classification/clustering on decompiled code obtained from disassembling the binary executable.

For example, Abou-Assaleh *et al.* [16] generate signatures that differentiate malicious code from benign code. They use I-Worm and Win32 virus executables as their dataset (dataset not available anymore), but it is unclear whether they use the decompiled source code as features or the binary executable itself. Their intuition is that malware authors sometimes use common tools to compile malware, which leaves signatures in the compiled binary file. They aim to detect such patterns and use them as signatures, which will also be able to detect never-before-seen malware compiled under similar conditions. They apply Ngrams (n=10) for classification, which is a highly common technique used in text classification. The distance metric they use is a straightforward k-nearest neighbor algorithm, where the samples having the highest overlap in Ngrams are classified as the

same category. Their method is 91% accurate in distinguishing between malicious and benign code. However, we believe that the results they show are unreliable because they present the numbers for classification by evaluating the classifier on the training data itself, which is known to yield high results. Secondly, when calculating the distance metric, although the metric is simple, they do not filter out the sequential data for common terms (by applying Term Frequency–Inverse Document Frequency (tf-idf), for example), which might skew their results.

Zhang and Reeves [17] aim to detect software variants, such as obfuscated malware or evolved versions of a software. They perform static analysis on the disassembled code of Windows executables. Their method is applicable to both malware and benign software. They generate patterns based on the flow of data and system calls in the disassembled code and measure their similarity. The similarity metric is a weighted function of the value of operands and operators present in the instructions of the software being compared. The intuition is that patterns coming from similar malware will be similar. In a dataset of 200 malware variants, their method is able to detect 90% of the similar pairs. However, their technique requires a predefined threshold value of the similarity measure above which samples are considered variants of each other. This parameter is unintuitive and highly dependent on the underlying dataset.

Suarez-Tangil *et al.* [18] aim to classify malware samples into families by performing statistical analysis of malware code. They decompile android apps into their Dalvik bytecode. Each malware sample is represented by a set of code chunks, which contain one or more methods. The idea is to identify common chunks across multiple malware samples. These common blocks serve as a signature for that malware family. They use a text-mining approach to classify these malware samples – by converting the code chunks into vector space and measuring distance between them using cosine distance. They report that their method is fast, scalable and produces good results having a classification error of mere 5.74%.

Sun *et al.* [19] aim to find repackaged apps and malware variants in android apps. Similar to the work of Suarez-Tangil *et al.* [18], the technique performs a static analysis of decompiled Dalvik bytecode of android apps. However, their approach utilizes Component-based Control Flow Graphs (CBCFG) to find the maximum number of overlapping code blocks among apps. Each node in a CBCFG represents an API call. Each app is then identified by its signature, which is composed of the author information in the META-INF file and multiple CBCFGs. The more similar the CBCFGs are, the higher chance there is that the two apps are variants of each other. Although they report a detection ratio of 96.6%, the similarity metric seems too stringent to detect variants that have a significant amount of code revised.

The study presented by Tajalizadehkhoob *et al.* [4] focuses on inject code evolution in banking malware. Inject codes are pieces of code, which a malware (specifically, banking malware) injects in the browser of an infected host with the goal of stealing credentials entered in a form or adding extra fields on the bank's web page to make the victim enter more information than what is actually required. This study is performed on a dataset provided by our current data provider – ABC. The authors analyze 1.2 Million inject codes and find that only 1% of the code is unique across the entire dataset. This suggests code reuse and code stealing among attackers. They also look at the evolution of inject code over time and find most of the versions to be exactly the same. Whenever they see differences in code, it suggests that the target's User Interface (UI) changed, so the attacker adapted the code to the change, while the rest of the code remained the same. Our research is different because we study the dynamic network behavior of the malware. However, our work is partially motivated by the results of this study – most of the attack code remains the same among variants of a malware family. Whether one can identify this similarity in the network behavior of the malware too is the question we aim to answer.

2.1.2. METADATA ANALYSIS

The work of Gupta *et al.* [20] is innovative since they characterize a malware from its description written in natural language. They collect their dataset from McAfee's threat library database. From what they have described, it seems like they have applied some text-mining algorithm, such as Ngram analysis to get their feature vector, based on which they generate a graph. They extract the actual features used by their system from the generated graph – fan-out of nodes, number of spawning children, the lifetime of the malware, and code sharing suggested by similarity. This can be considered static analysis since the malware sample itself has not been executed.

In addition, the work of Johnson [21] clusters DNS-based malware samples using the dynamic execution trace metadata provided by VirusTotal. Their feature set includes code size, imports, and DNS requests made. This study can also be considered as static analysis because the feature set they use does not incorporate any dynamic aspect of the malware. They also acknowledge that their dataset is not representative and that the applicability of a particular feature is dependent on the kind of data set. Finally, they use Agglomerative linkage-based algorithm to perform the clustering.

SUMMARY

In summary, the studies mentioned above either use disassembled code of the binary executable or metadata of the malware to classify it as malicious or benign. The techniques they utilize range from building CFGs to using text mining algorithms. Since all the static-analysis techniques rely on accessing the code, they suffer from code obfuscation problem. On the other hand, analyzing the metadata may be effective for exploratory analysis, but it does not say anything about the actual behavior of the malware sample.

2.2. DYNAMIC ANALYSIS

In this section, we discuss behavioral analysis approaches for clustering and classification. Generally, collecting a malware's behavioral information on an operational device is a challenging task since several other services are also running in the background, which can add noise to the collected data. The most common workaround is to execute the malware sample in a controlled environment and to collect the traces generated by that sample. Most of the research work analyzed further in this section collects behavioral data in this way.

2.2.1. SYSTEM-TRACE ANALYSIS

Bayer *et al.* [22] want to cluster malware variants together so that one does not have to analyze a malware that has been seen before. They collect system-level symbols, such as OS objects and operations, and network API calls. The main objective of this approach is to keep it scalable, which they achieve by utilizing Jaccard index – a set-specific distance measure and Locality Sensitive Hashing [32] – a technique that maps input items to their hashes with the goal of achieving hash collisions for similar items.

Sun *et al.* [6] build a malware variant detection system for Android applications. It combines the static and dynamic analysis by collecting features from both sources. The authors collect information from the static symbols in the manifest file. They also generate a run-time behavioral graph (a Control Flow Graph made from static analysis of code), which is enriched with run-time symbols collected from executing the malware. They compare the resulting behavioral graphs using a

simple *edit distance* (number of additions, deletions, and replacements required to convert graph A into graph B). If the similarity is higher than a predefined threshold, the two samples are considered variants of each other. They report a classification accuracy of 99%, which seems a little too good to be true. One negative aspect of this paper is that they have developed their own malware sample for the evaluation. They also apply obfuscation techniques themselves to make malware variants. Developing self-developed synthetic malware samples can potentially skew results in their favor. This is also why we resort to collecting real-world malware samples.

The Master thesis presented by Wong Hon Chan [23] is one of the examples of using sequences-as-features for clustering. The author utilizes the sequence of accessed resources in Android applications to group applications that behave similarly. He uses the length of the Longest Common Subsequence (LCS) between two access sequences as the measure of similarity. Then, he uses the K-means algorithm to cluster the behaviors based on a pre-computed distance matrix. He also defines a number of cluster validation techniques for quantifying the cluster quality. His results motivate the use of sequences-as-features and show LCS to be a promising metric to cluster similar-behaving resources.

2.2.2. NETWORK-TRACE ANALYSIS

There is a variety of work that clusters malware samples based on their network behavior, and they all rely on the labels provided by the Anti-Virus (AV) vendors to calculate the accuracy of their approach. On the contrary, research has shown that AV vendors do not use a standardized naming convention for malware samples [33]. In addition, the labels that AV vendors assign to malware samples are heavily based on the system-level behavioral analysis rather than the network-level behavioral analysis. Hence, it is essential to think about the applicability of such labels if one aims to utilize only the network behavior of the malware to perform the classification/clustering.

Providing an alternative to the system-level behavioral clustering of Bayer *et al.* [22], Perdisci *et al.* [9] develop a 3-step clustering algorithm, which operates purely on network traffic. Their intuition is that if the structure of the request query of multiple malware samples is similar, then it is more likely that they belong to the same family, that the developers may be related, and even that code reuse might have occurred. The 3-step algorithm works by first coarsely clustering the traffic based on statistical features, then splitting those clusters further by utilizing the query structure, and finally merging the clusters that are most similar (by comparing their centroids represented by regular expressions). They check the cluster quality by introducing their own way of calculating cluster cohesion and cluster separation. They build graphs from the labels (family names) assigned by AV vendors to the samples. These labels form the nodes of the graph, and the edges between nodes are drawn when two AVs label the sample as names in the corresponding nodes. Essentially, the graph shows the agreement between labels assigned by the AVs. Clusters are cohesive if all the samples in a cluster have the same label. Clusters are well-separated if there is no overlap between the labels of distinct clusters. Figure 2.1 shows what cohesive and well-separated clusters look like. This algorithm can cluster 12M HTTP requests in a few minutes while producing very low false positives and is able to achieve higher accuracy than the work of Bayer *et al.* [22]. One negative point is that the authors do not justify most of the decisions regarding parameter values of their system. Secondly, since the main feature of their approach is the request URL of the malware, their method is only applicable for clustering HTTP-based malware. Similarly, multiple approaches exist specifically for DNS-based malware [24, 34], and HTTPs-based malware [35, 36].

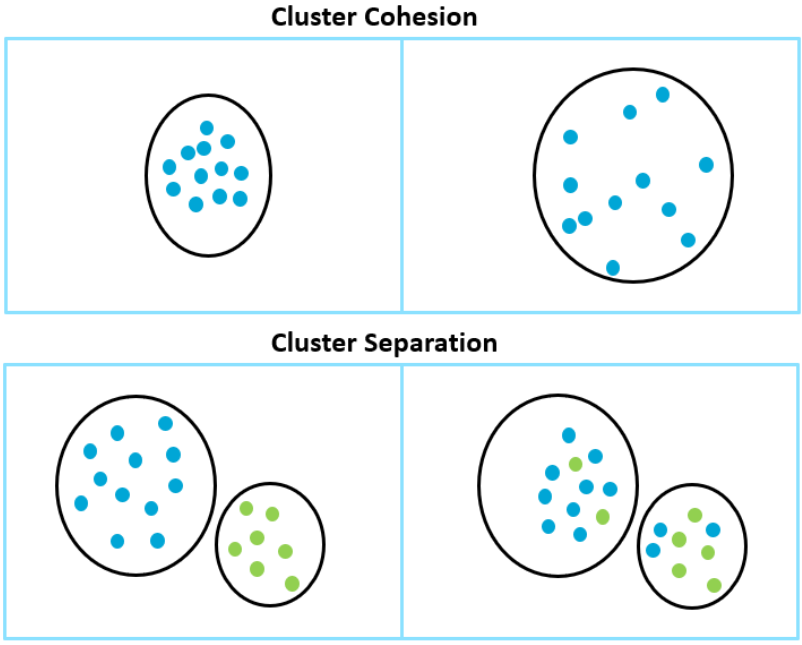


Figure 2.1: High (Left) and Low (Right) cluster cohesion and separation

In the study presented by Tegeler *et al.* [25], they build a classification system to detect network traffic generated by bot-infected hosts. Similar to our work, their emphasis is on using high-level features. Unlike us though, they use statistical features from netflows (aggregated packet transfer based on connections), such as average time between the start times of two subsequent flows, the average duration of a connection, the number of bytes transferred to the source, etc. They report a detection accuracy close to 100%, while only producing a few false positives. This study provides evidence that a high-level feature set can indeed be used to characterize a malware's network behavior.

In the work of Mohaisen *et al.* [26], the authors use an ensemble of multiple clustering algorithms to classify malware families. They specifically use the Zeus botnet's network activity and have used domain experts for manually labeling it as ground truth. They also provide feature ranking, where they conclude that port information is the least useful in characterizing malware variants.

Network traffic analysis does not only have applications in malware analysis. The work by Korczyński and Duda [27] studies the traffic generated by 12 everyday-use software such as Skype, Paypal, and Twitter to see if they can fingerprint their encrypted traffic using Markov chains. They study single-directional traffic coming from a server to a client and use Secure Socket Layer/Transport Layer Security (SSL/TLS) header information (message types, in particular) as features. They provide a framework for encoding types of TLS messages in categories, which makes the handling of sequential data much more manageable. This idea has been borrowed when we discretize numerical sequences. They also provide the insight that their method does not model how the software itself performs but rather, how it utilizes the TLS protocol. This insight also applies to our work as the clusters would not model the full working of the malware, but rather how they utilize the Internet to carry out their objective. Specifically, it models the behavior that is directly associated with the chosen features. Therefore, it is of the utmost importance that the feature set characterizes as much of the malware behavior as possible.

Bilge *et al.* [28] use netflows to detect botnet Command and Control servers. They use the fol-

lowing features: 1) Flow-size-based features (number of bytes transferred, auto-correlation features that help see patterns in time series, unique flow sizes based on the assumption that malicious entities transfer similar amounts of data with their respective Command and Control servers), 2) User-behavior-based features (access patterns that look at the regularity in time difference between flows because bots communicate with their Command and Control servers in regular periods, unmatched flow to detect zombies where no return flow is detected), and 3) Temporal patterns (time when the traffic is generated, e.g., traffic generated at night time with respect to the corresponding time zone is suspicious). Finally, they use Random Forests to perform the classification. Their tool reaches an accuracy of 65%.

Ghorbani and Nari [29] also use netflows to classify malware samples as malicious or benign. They build graphs from netflow data, where the nodes are protocols, and directed edges represent flows grouped by IP addresses. Then, they extract the following features out of the graphs: graph size, average fan-out of nodes, and the number of specific protocol nodes (limited to SSL, DNS, FTP). Using these features and WEKA classifier, they classify the samples as malicious or benign. Their intuition is that similar malware samples will have similar graphs and hence, similar features. They recommend using application-level features to improve accuracy because the behavior at network-level seems similar for some samples.

SUMMARY

In summary, the studies mentioned above utilize the system and network traces generated by executing malware to cluster and classify it. However, there are a number of problems with these approaches. First, comparing with network-level activities, collecting system-level activities is not always possible as it is quite privacy-intrusive and takes a lot of effort to access each end host. Second, because of the feature set used, most of the techniques mentioned above are applicable only to one type of protocol. Additional research and tweaking are required to make them work with other types of malware. Third, although netflows are more privacy-preserving than Pcap files, they are often sampled, which limits their applicability for building sequential behavioral models. Therefore, in this research, instead of using statistical and privacy-intrusive features as recommended by Ghorbani and Nari [29] and Tegeler *et al.* [25], we combine the use of Pcaps and sequences of high-level features extracted from a malware's network traffic to cluster malware families.

2.3. MALWARE CHARACTERIZATION

Many papers were studied to get an idea of the feature set that would best characterize a malware's network activity.

For example, the work by Jiang and Zhou [33] presents a theoretical study on the characterization and evolution of Android malware. The authors create a dataset of android apps and then categorize them by reading blogs, news, etc. They also perform some behavioral analysis regarding how the different malware families install themselves, what they exploit, how they spread and what they do (specifically: Installation, Activation, and Payloads). They discuss some useful features, e.g., the encryption keys used and naming convention of malicious embedded apps. They also identify that different AV vendors name the same malware differently, which shows the lack of consistency and coordination among the AV vendors. This is why we believe that there is a possibility of mislabelling malware samples in the provided dataset.

In the work of Black *et al.* [37], the authors present a detailed survey of seven Windows-based banking malware. It identifies 1) the common malware behaviors among the samples that are stud-

ied, 2) an almost-exhaustive list of ways those behaviors can be achieved in Windows OS, and 3) how each of the malware family performs those actions. In addition, because of the difficulty in characterizing malware behaviors, the authors conclude that this task will remain somewhat manual in nature.

Unfortunately, in the aforementioned studies, the authors only characterize system-level behavior, which cannot be used for our project.

Cui *et al.* [30] utilize the network activity of everyday-use applications to reverse engineer their communication protocol, which is essentially the core behavior of the software. Their results provide evidence that the software's Internet-usage behavior is visible in its network traffic. Hence, we can conjecture that the same would hold true for malware and that the clusters obtained would reflect different types of Internet-usage behaviors. Whether different malware families exhibit distinguishing Internet-usage behaviors is one of the questions that we aim to address in this project.

Secondly, a challenge in using sequences-as-features is determining the reasonable length of a sequence. For example, longer sequences would be better-suited for building state machines, since they need to compute probabilities of transitions, which only makes sense on a large enough sample size. On the other hand, if one wants to detect handshakes, a small chunk of sequence from the beginning would be enough. For example, Wang *et al.* [31] aim to reverse engineer the protocol used by a software using its network traces. They identify that the first 3-bytes extracted from the application header of each packet are a distinguishing pattern in determining the 'message format'. Hence, they use those small strings to cluster different types of network messages, which they eventually use in determining the protocol. In this project, we use this insight and take only the first few packets to construct sequences.

2.4. DISCRETIZATION TECHNIQUES

In Sequence Clustering, it is important to define the notion of 'similar sequences'. This, in turn, depends on many factors, such as the data type and distribution of the sequences. Some distance measures are only applicable to categorical sequences. For example, the distance between the numeric sequences [1,2,3,60,70,80,900,1000] and [2,3,4,61,71,82,901,1001] should not be very large, since the point-to-point values are very close together. On the other hand, if these sequences are considered as categorical in nature, each distinct value is considered a different category. Hence, the distance between these two sequences will be large. Moreover, the key challenge is determining the thresholds/boundaries that discretize the numerical sequence into categories. Detailed surveys on discretization techniques can be found in [38], [39], [40]. We only summarize the most commonly used techniques in this section. There are two main types of discretization techniques: 1) Supervised discretization, which knows the class labels, and 2) Unsupervised discretization, which does not have class information. In this project, we deal with numeric sequences without assuming any class information. Following are some of most commonly used Unsupervised discretization techniques:

1. **Equal-Width Interval Discretization:** This method calculates the boundaries based on the minimum and maximum value of a sequence, and the number of user-defined categories. The interval between each boundary is fixed, irrespective of the number of items that fall in each category. For example, for a minimum value of 10, a maximum value of 50 and 2 categories, the interval of each class is 20 (`calculation:` $\frac{50-10}{2}$), so the boundary lies at 30 (`calculation: 10+20`). If most of the values in the sequence are below 30, the first

category will have a lot more samples than the second category.

2. **Equal-Frequency Discretization:** This method finds the minimum and maximum values of a sequence, arranges the data items in ascending order, and then assigns the sorted values to categories such that the number of items in each category remains the same.

3. **Entropy-based Discretization:** This method recursively evaluates a number of boundaries and chooses the best one based on minimum Shannon's entropy [1]. However, performing this process for a larger number of sequences quickly becomes infeasible.

4. **Percentile-based discretization:** This method divides a sequence into categories based on pre-defined percentiles as the boundaries. For example, assuming three categories, in sequence [1,1,1,1,4,4,5,7,7,7,1], the 33^{rd} percentile is 1 and the 66^{th} percentile is 5. Hence, all values lower than 1 fall in the 1^{st} category, all values lower than 5 fall in the 2^{nd} category, and all other values fall in the 3^{rd} category. This approach is used by Pellegrino *et al.* [41] to discretize numeric sequences such as duration, bytes, and number of packets by assigning them a cluster based on which percentile they lie in.

The above-mentioned techniques are either too computationally heavy, do not take into account the underlying data distribution, or operate locally on a single sequence at a time. The particular challenge that we have is: knowing the number of categories, which boundary selection makes sense for the data set under consideration. This problem is both, data distribution- and context-dependent.

2.5. RESEARCH GAP

There are a number of gaps in the research that we have reviewed above.

Firstly, there is increasingly more emphasis on Deep Packet Inspection to extract good features that characterize a malware's network behavior. Although such features give a better view of the behavior, because of privacy concerns, these tactics may not be applicable in the near-future. On the other hand, using high-level features do not provide an exact behavioral profile, which we want to build. Hence, the first research gap we fill is to use sequences of high-level features, instead of the previously-used single-valued features to evaluate the kind of behaviors they are able to capture. We also compare clusters generated with sequences-as-features with a baseline version that does not incorporate sequence data. Finally, we also test various distance measures to define the notion of 'similarity' when dealing with sequences-as-features.

Secondly, network traffic analysis has been used to solve a number of problems. However, to the best of our knowledge, there does not exist an analysis of which level of granularity is appropriate to achieve certain goals – whether to use all traffic as one sequence, whether to split incoming and outgoing traffic as two sequences, or whether to split the traffic into connections to generate sequences. Hence, we fill the second research gap by performing clustering on two levels of granularity – the whole Pcap file considered as a sequence (Pcap-level) and individual connections considered a sequence (Connection-level). We provide insight into what kind of behaviors are visible at each level.

Thirdly, to the best of our knowledge, there does not exist any work that compares malware families' system-level behavioral aspect with their network-behavioral aspect – how many attacking

[1] https://en.wiktionary.org/wiki/Shannon_entropy

capabilities are exhibited by a particular malware, and how many malware families have common attacking capabilities. We fill this gap by considering malware family labels as clusters on the static-analysis level and compare these clusters with those generated using a malware's network activity.

Lastly, discretizing sequences is a well-studied topic. However, to the best of our knowledge, we have not found a technique that is both context-dependent and is simple to use at the same time. Hence, we evaluate three simple discretization techniques and show under what circumstances they are applicable.

In summary, we combine different aspects of network traffic analysis from various papers and consolidate it all in one master thesis with the goal of filling all the aforementioned research gaps.

3

PRELIMINARIES

This chapter briefly explains the algorithms and concepts required to understand the rest of the report. It also outlines the design decisions for choosing the particular techniques.

3.1. SEQUENTIAL DATA

A sequence is an ordered list of items. Sequences capture subsequent values of a variable and therefore, have the ability to capture behavioral changes in that variable over a period of time. The 'order' of the items is important, because, for example, keeping the order intact in a recipe is what allows one to bake the perfect cake, and a particular order of actions allows one to predict the next moves of a person. On the contrary, a single value of a variable captures only the current state or only the aggregate of multiple states, which may not be appropriate to represent the intricacies of evolving behavior.

In this project, we are interested in clustering the network behavior of malware samples. In order to do that, we need to capture each malware sample's behavior in one or more features. Because of the implicit temporal aspect of sequences, they seem to be the intuitive choice for features.

There are two types of sequences that we work within this project:

1. **Numeric sequences:** Numeric sequences can be considered as time series. Techniques from time series analysis can be used to measure the distance between two numeric sequences. Some examples of numeric sequences are sizes of the alphabets written when writing your name, and the number of bytes transferred in ten subsequent packets.
2. **Categorical sequences:** Categorical sequences are much difficult to deal with because of their non-generalizable and non-standard notions of similarity. Bio-informatics is a field of Computer Science that frequently deals with categorical sequences, such as DNA and RNA strains. They align DNA sequences to identify which species might be related to each other. Moreover, text matching can also be considered as a categorical sequence-matching problem – an alphabet represents its own category and the more overlap there is between alphabets of two words, the higher the similarity score is.

Sequences can be represented in many ways, e.g., as time-series or by transforming them into

another form using dimensionality reduction techniques. We represent the sequences used in this project in two ways:

1. **Sequences-as-lists:** Values at different time steps are collected in a list to form one long sequence of events. For example, the number of coffees I drink in a week can be represented as: [1,2,3,4,1,1,1], which shows a partially increasing trend as the week goes on and drops as the weekend approaches.
2. **Ngrams:** In Computational Linguistics, *ngram* is defined as the set of `n` consecutive items in a given sequence. n=1 is called a unigram, n=2 is called a bigram, n=3 is called a trigram, and so on. Ngrams can capture the structure of a sequence from a statistical point of view. The larger the value of n is, the more structure is captured. A sequence is converted into a set of ngrams by using a sliding window of length n. For example, the sequence `A = [1,1,2,3,4,2,3]` will be converted into the following bigrams: `[(1,1),(1,2),(2,3),(3,4),(4,2),(2,3)]`. The order in which the ngrams are stored now becomes fluid since each ngram captures a little piece of the whole temporal behavior within itself.

A packet capture (Pcap) file consists of the packets sent and received to/from a host during the interval the capture was in effect for. Each packet is composed of many fields, which can be used as features depending on which OSI layer we look at. For example, at the Network layer, we get access to the IP address of the domain a user is trying to contact, such as that of a mail server or a malware trying to contact its Command and Control server. At the Transport layer, we get access to the port number through which the service is communicating to and being communicated with. Port numbers sometimes also indicate the protocol being used for the communication, such as port 80 for HTTP and port 22 for SSH. At the highest level – the Application layer, direct interaction with the service takes place, which makes the exact message sent or received visible (either in plain-text or encrypted), as well as the URL of the request sent to that service. For example, it could be the credentials to a service a user is logging in on or personal pictures that a user is archiving to the cloud. Each layer provides a slightly different view of the network behavior. However, one thing is constant – a sequence of individual actions capture the long-term behavior of a host much better than aggregated, singular values.

3.2. MEASURING DISTANCE BETWEEN SEQUENCES

Characterizing behavior using sequences is an intuitive design choice. However, it is not straightforward to define the notion of 'similar sequences' in order to measure the distance between them. A number of techniques exist to measure the distance/similarity depending on the type of sequence and how the context defines 'similarity'. In this section, we describe a few distance measures that have been evaluated during this project. There are two measures for numeric sequences, i.e., Euclidean (point-to-point) distance, and Dynamic Time Warping; and there are three measures for categorical sequences, i.e., Sequence alignment, Longest Common Subsequence, and Ngram analysis.

A NOTE ON DISTANCE VERSUS SIMILARITY

On a scale of 0.0 to 1.0, when two items are exactly alike, the similarity is 1.0, while the distance between them is 0.0. In contrast, when two items are completely different, the similarity is 0.0 while the distance is 1.0. This means that similarity and distance are inverses of each other. In the rest of the report, we define distance as:

```
distance = 1.0 - similarity
```
(3.1)

; where the similarity score has been scaled to [0-1].

PROPERTIES OF DISTANCES

Distances need to fulfill four conditions to be considered metric:

1. `Non-negative: d(a,b) >= 0`
2. `Symmetric: d(a,b) == d(b,a)`
3. `Exact match: d(a,b) == 0 iff a == b`
4. `Triangular inequality: d(a,b) <= d(a,c) + d(b,c)`

Metric distances are especially required when comparing distances computed from different methods. Metric distances always calculate the same distance, which makes different distance measures comparable.

3.2.1. EUCLIDEAN (POINT-TO-POINT) DISTANCE

One standard technique used to measure the distance between two numeric time series is to map each point of one sequence to each point of the other sequence and then to calculate the Euclidean distance (or Manhattan distance, etc.) between each point-pair and summing it all up. An example of this technique is shown in Figure 3.1. The blue and orange series represent two sequences, and the dotted line between the series represents point-pairs between which distance is computed. This technique is easy to use, but cannot handle sequences of different lengths. It also does not take into account out-of-phase sequences involving delays but otherwise having the same structure.

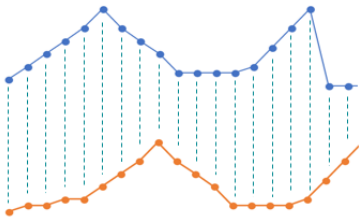


Figure 3.1: Point-to-point distance measure

3.2.2. DYNAMIC TIME WARPING

Dynamic Time Warping (DTW) is an old algorithm, developed in 1983, which was meant to calculate the distance between two curves in the presence of distortions (or warps) in the time-axis. DTW considers the distance calculation between two numeric sequences as an optimization problem where the objective function is to map points from one sequence to that of the other sequence in a way that minimizes the overall distance between the two sequences. DTW is a more robust technique than the standard point-to-point mapping mechanism because it is able to find local substructures in one sequence and map them to that of the other sequence.

DTW iteratively builds the solution by computing the distance between points of the two sequences and summing it up with the minimum solution found so far in the immediate neighborhood. The neighborhood is defined as the values of 1) taking one step in the x-direction, 2) one step in the y-direction, and 3) one step in both x- and y- directions. This is formally written in the following equation:

$$D(i,j) = d(A_i, B_j) + \min(D(i-1,j), D(i,j-1), D(i-1,j-1)) \tag{3.2}$$

Where `d(_,_)` is the distance measure to compute the distance between two sample points, e.g., Euclidean distance or Manhattan distance. The three parameters of `min(_,_,_)` are the values of the neighborhood, which are essentially sub-problems of the main problem. A graphical representation of distance computation is shown in Figure 3.2, where the two sequences are written on the x- and y-axes.

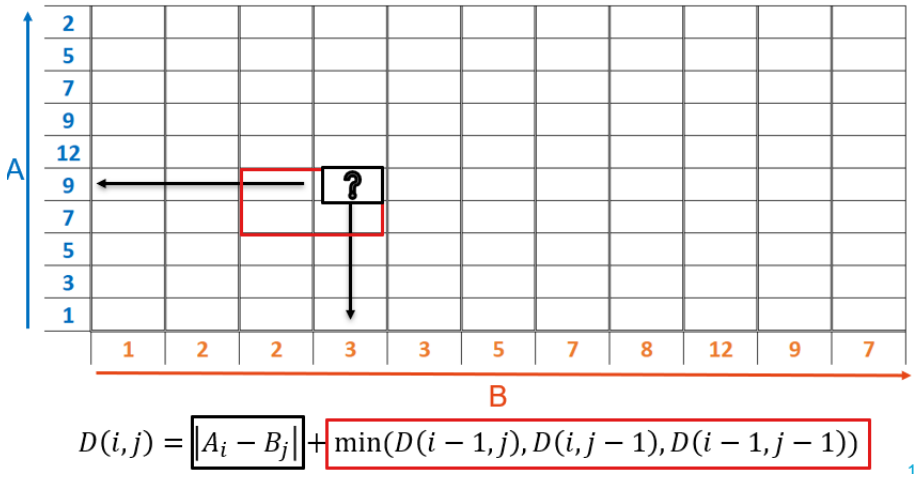


Figure 3.2: Graphical representation of the DTW equation

The output of DTW is a 'similarity' score. This score is only meaningful when compared with the similarity scores of other sequences. Therefore, one must first collect all the similarity scores, and scale them to a self-defined range before they can be used further. Moreover, the similarity score is converted to a distance score using the equation 3.1. This score measures the distance between two sequences but is not a 'distance metric' since it does not fulfill the triangular inequality. In our case, since we do not compare distances calculated using other methods, we do not care if a distance measure is metric. An example of DTW in action is shown in Figure 3.3. The blue and orange series represent two sequences, and the dotted line between the series represents point-pairs that are mapped to each other for distance computation. The only negative aspect of using DTW is that it takes a long time to compute even on the fastest implementation of Python – fastdtw [42], which is used in this project. However, the benefits of using DTW far outweigh the speed issue, as it is incredibly noise tolerant and can calculate an interpretable distance between sequences even in the presence of delays. Therefore, DTW was selected to compute distances between numeric sequences.

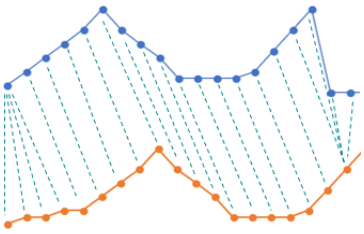


Figure 3.3: Dynamic Time Warping distance measure

3.2.3. SEQUENCE ALIGNMENT

Sequence alignment is a way of arranging sequences of DNA, RNA or protein to identify regions of similarity that may be a consequence of the functional or evolutionary relationship between sequences [1]. Sequence alignment can be considered as the categorical alternative of DTW, but instead of using Euclidean distance, each match gets a reward, while each mismatch and an introduced gap gets a penalty. When two data points have the same category, it is considered as a *match*; otherwise, it is considered as a *mismatch*. A *gap* is an empty character inserted in place of a mismatch to avoid getting a high penalty. There are two types of alignment methods: 1) Local alignment: when one wants to find a sub-string in a longer string, and 2) Global alignment: when one wants to match two full sequences with each other. The output of the sequence alignment algorithm is an alignment score, similar to DTW. The main drawback of using Sequence alignment is that it requires the appropriate substitution matrix, which gives scores for matches, mismatches, and gaps. In the case of Bio-informatics, many of these substitution matrices have been introduced over the years. Unfortunately, for malware analysis, no such matrix exists. Moreover, in the presence of default parameter values, the algorithm performs worse than any other technique we evaluated. Therefore, it was decided not to use this technique further.

3.2.4. LONGEST COMMON SUBSEQUENCE

As the name suggests, the Longest Common Subsequence (LCS) algorithm [2] finds the longest consecutive sequence of items that are common to both categorical sequences. This technique does not give a distance score, but rather the LCS of the two sequences. Hence, one must figure out how to map it to a number. In the thesis presented by Wong Hon Chan [23], the length of LCS is used as the measure of similarity.

Consider the following example: The LCS between sequence `A = [1,1,2,3,4,2,3]` and sequence B = `[2,2,1,2,3,3]` will be `[1,2,3]` and the similarity score would be 3.

However, in the case of this research, the results produced by LCS were uninterpretable. There were cases where the two sequences looked nothing alike, but because they both had, what we defined noise, the LCS turned out to be the noise, rather than interesting behavior. Hence, the distance measure was overshadowed by the presence of delays and noise in the sequences. Therefore, it was decided not to use LCS anymore either.

[1] https://en.wikipedia.org/wiki/Sequence_alignment
[2] https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

3.2.5. NGRAM ANALYSIS AND COSINE DISTANCE

As mentioned previously, sequences are broken into multiple smaller sequences, which we call Ngrams, using a sliding window of length `n`.

Consider the following running example: Sequence `A = [1,1,2,3,4,2,3]` will be converted into the following bigrams: `[(1,1),(1,2),(2,3),(3,4),(4,2),(2,3)]`; and Sequence B = `[2, 2,1,2,3,3]` will be converted into the following bigrams: `[(2,2),(2,1),(1,2),(2,3),(3,3)]`. We define a set of bigrams as a sequence's *Bigram Profile*.

Once all sequences are transformed into their bigram profiles, a set of common bigrams (let's call it `C`) is selected. Then, for each sequence, a numeric list equal to the length of `C` is generated. For each bigram in `C`, its frequency in the sequence's bigram profile is appended to the list. If a bigram does not exist in the bigram profile, 0 is appended instead. This step transforms the bigram profiles into their vector representation.

In the above example, the common set C would be: `[(1,1),(1,2),(2,3),(3,4),(4,2), (2,2),(2,1),(3,3)]`. Hence, the resulting vectors for the two sequences will be: `sequence A = [1,1,2,1,1,0,0,0]` and `sequence B = [0,1,1,0,0,1,1,1]`

Cosine curves can be used to measure the distance between two vectors, which is essentially the cosine angle between the two. 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). In the above example, the cosine similarity comes out to be 0.526, so the distance between sequence A and B is 0.474 (from Eq. 3.1).

This technique is, by far, the most commonly used one in literature to measure similarity in text [43, 44], in DNA and RNA sequences [45], and even in software code matching [16]. In addition, this technique is the fastest and provides the most interpretable results among all the techniques we have tried. Hence, Ngram analysis has been used to compute distances between categorical sequences.

3.3. CLUSTERING ALGORITHMS

Once the distance between two sequences can be computed, a mechanism is required that can group the sequences closest to each other. There exists a family of clustering algorithms in Machine Learning that aim to group items in bins based on their similarity. Each algorithm has its pros and cons. We narrowed down the algorithms we wanted to use based on the following criteria:

- Must be able to operate on pre-calculated distance matrix and not on the original dataset since our dataset is a collection of sequences that many algorithms cannot handle.
- Clusters should be free to be of any shape and size for robustness.
- The algorithm should require fewer parameters to be tweaked.
- The algorithm's output is intuitive and easy to analyze.
- The algorithm is known to give good results.

3.3.1. AGGLOMERATIVE CLUSTERING

Agglomerative clustering is a type of 'hierarchical clustering', where the goal is to build a hierarchy of clusters. Agglomerative follows a bottom-up approach, where each point starts in its own cluster. Then, iteratively, the clusters that are closest to each other are merged to form a larger cluster. This process is repeated until only one cluster is left. For example, consider the clustering of three points `(a, b, c)` as shown in Figure 3.4. Each point forms its own cluster. In the next step, `a` and `b` are merged, followed by merging with `c`. The distance between the newly formed cluster and the rest of the clusters is the minimum distance between its contained points and the rest of the clusters. This is what the 'Single-Linkage' clustering does. There are other methods to measure the distance as well. For example, the 'Complete-Linkage' calculates the maximum distance between a cluster's points and the rest of the clusters, and 'Average-Linkage' computes the average distance between a cluster's points and the rest of the clusters. However, Single-linkage is the most commonly used method.

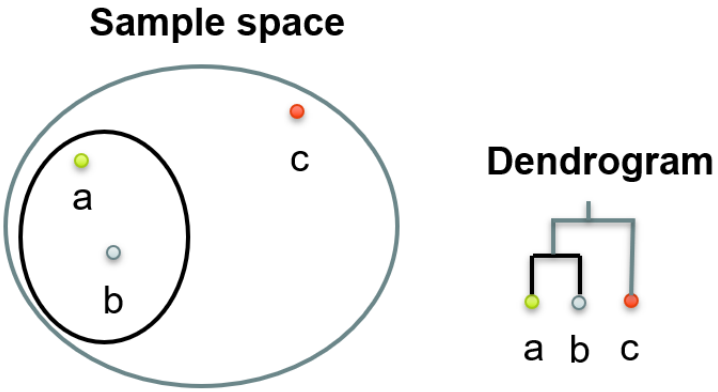


Figure 3.4: Example of Agglomerative clustering

The output of this algorithm is a dendrogram, where leaves are individual data points, and the increasing levels of the tree show the cluster mergers (right side of Figure 3.4). It is used to represent phylogeny in different fields, e.g., species of animals or relationships between malware families. The downsides to using this algorithm are that the Single-linkage is very sensitive to noise and that the user has to provide a cut-off point manually, which is an unintuitive parameter and will result in clusters of equal sizes.

3.3.2. HDBSCAN

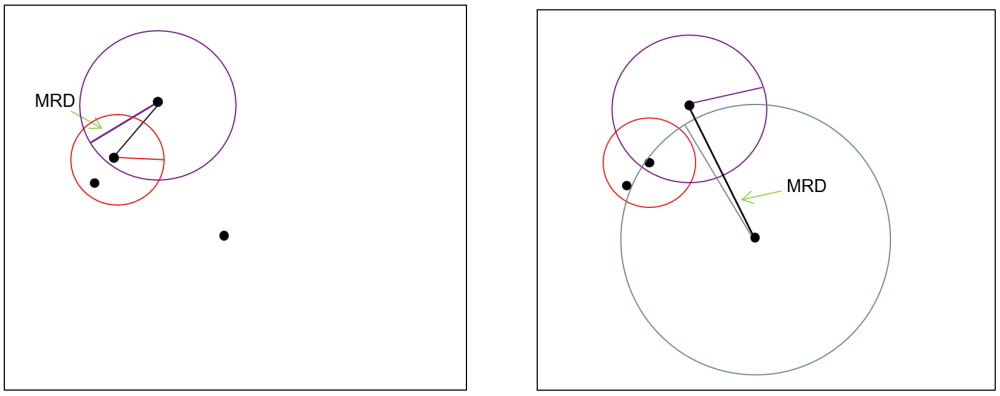
HDBScan [46] stands for Hierarchical DBScan algorithm – an extension of DBScan, which used to require more parameters and was unable to build clusters of varying densities. HDBScan uses robust Single-linkage hierarchical clustering and then extracts flat clusters based on cluster stability. It is a density-based algorithm, meaning clusters are groups that are located in a densely populated area. HDBScan has two input parameters, i.e., minimum cluster size and k-nearest neighbors.

The underlying algorithm is Single-linkage, which is very sensitive to noise. Real data has noise and outliers, so a data point in a wrong place can serve as a bridge between two clusters. To mitigate this problem, the idea is to conceptually push away the noisy points from 'good' points and

to reduce the distance among 'good' points that are placed closer together. *Noise* can be intuitively defined as points whose probability of membership to any cluster is too low, as a result of either being too far away from all other points or by forming a bridge between two or more clusters. In order to do this, an estimate of density is required. The concept of Mutual Reachability Distance (MRD) is used to conceptually bring points in high density closer together and increase the distance of points lying in low-density areas. The MRD between points a and b is the maximum of two distance metrics defined below:

1. The kth nearest neighbor provides a local estimate of density. The 'distance' is equal to the radius of a circle that one needs to draw around a point to encapsulate k nearest neighbors. Naturally, if a point is in a high-density area, the radius will be small. Conversely, the radius will be much bigger if the point lies in a low-density area. This distance is also called the *core distance* of a point.

2. The original distance measure between two points. This value can be read off from the pre-computed distance matrix.



(a) When points are close-by

(b) When points are farther away

Figure 3.5: Working example of Mutual Reachability Distance

Consider the points in Figure 3.5. Let k=2, so the circles would be drawn of a radius such that 2 points can be encapsulated inside the cluster. In case (a), the points are closer together, so the MRD is the core distance of the bigger (purple) circle, which is also the same as the original distance measure between the two points (shown as a black line). In case (b), the points are far away from each other, so they get pushed even farther as the MRD is the distance between the two points (shown as a black line). The higher the value of k is, the more noise will be integrated inside the clusters.

Next, a weighted graph is generated where the data points are vertices, and the edges represent the MRD between points. Then, given an iteratively decreasing value of a threshold, vertices whose MRD is higher than the threshold are dropped. Essentially, we start with a fully connected graph and move towards a fully disconnected graph. The subsections of the graph that are connected to each other are called *connected components*. The different values of the threshold give a hierarchy of connected components. In order to make this algorithm scalable and more efficient, the concepts of spatial indexing and minimum spanning tree are used in the implementation of connected components.

Next, we want to extract flat clusters from the resulting hierarchical tree. This is where the 'minimum cluster size' parameter is used. At each level, when the tree splits into a branch that has less items than the minimum cluster size, it considers them as falling out of the cluster and labels them as *noise* instead of putting them in a new cluster. The longer a cluster remains intact – points fall out of the cluster, rather than getting split into multiple clusters, the more persistent or stable that cluster is considered. The stability of a cluster is formally defined as the distance from the birth of the cluster until it gets split into new clusters. In order to extract the final clusters, we traverse the tree bottom up, choosing clusters that have higher stability than that of its descendants. This step is there to avoid having larger clusters when the smaller clusters have been more persistent and have sizes larger than the minimum cluster size parameter. Finally, we end up with clusters of different shapes and sizes, where the points are more likely to be in the correct clusters since we only select the stable clusters and remove noisy points in the process.

3.4. CLUSTER VALIDATION TECHNIQUES

Clustering algorithms provide a dataset divided into groups based on similarity. In case the labels are present, one can use a test set to validate the quality of clustering by calculating the precision and recall. On the other hand, if the ground truth labels are absent, there is no intrinsic way of calculating the quality of the clusters. Many indexes exist that try to quantify the cohesion and separation of the clusters.

- *Cluster cohesion* is defined as how similar and close by the items are within a cluster.
- *Cluster separation* is defined as how diverse are items belonging to different clusters.

A clustering is considered generally good if it exhibits high cluster cohesion and high cluster separation.

3.4.1. SILHOUETTE INDEX

Silhouette analysis presented in 1987 by Rousseeuw [47] can be used to quantify the separation distance between the resulting clusters. It is a graphical representation technique, where the silhouette plot shows how well within the cluster a point is. The silhouette value represents how similar a data point is to the members of its own cluster and how different it is to the members of other clusters. This measure has a range of [-1, 1]. Let i be the centroid of a cluster; $s(i)$ be the Silhouette value of that centroid; $a(i)$ be the average distance between i and all other data points within the same cluster (a way to measure cluster cohesion); and $b(i)$ be the lowest average distance of i to all data points in any other cluster, of which i is not a member (a way to measure cluster separation). Then the Silhouette index is formally written as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3.3}$$

which can also be written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ a(i)/b(i) - 1, & \text{if } a(i) > b(i) \end{cases} \tag{3.4}$$

3.4.2. DAVIES-BOULDIN INDEX

DB Index is an older measure, introduced in 1979 by Davies and Bouldin [48]. It is an internal evaluation scheme, where the validation of cluster quality is made using features inherent to the dataset. Hence, a good value reported by this method is non-comparable to other indexes. Let $M_{i,j}$ be the separation between the i^{th} and the j^{th} cluster; S_i be the within-cluster scatter for cluster i; and S_j be the within-cluster scatter for cluster j. DB Index aims to maximize the inter-cluster distance ($M_{i,j}$) and minimize the intra-cluster distance (S_i and S_j). The equation for DB index is formally defined as:

$$D_i = \max_{j \neq i} \frac{S_i + S_j}{M_{i,j}} \tag{3.5}$$

If N is the number of clusters:

$$DB = \frac{1}{N} \sum_{i=1}^{N} D_i \tag{3.6}$$

3.4.3. VISUAL ANALYSIS

This technique is manual as one has to manually explore each data point in a cluster and inspect whether its current assignment to a cluster makes sense. Utilizing visualizations, such as heatmaps and scatter plots help ease the analysis, such that one can visualize the dataset itself and can make their own 'visual clusters'. Then, one can compare the result of the 'visual clusters' to the clusters provided by the algorithm. This technique is highly subjective but works well because humans are typically good at finding patterns and similarities using visual analysis [49].

This technique is particularly useful when one wants to perform an exploratory analysis of whether the data inside a cluster makes sense or not, as opposed to when validating the clustering algorithm itself. In addition, this technique also works in cases where all other cluster validation indexes fail, since the 'goodness' of a cluster is also self-defined and context-dependent notion [50].

3.5. SUMMARY

In this chapter, we explained the necessary background required to fully understand the rest of the report. We first describe sequential data and why it is a good choice for behavioral modeling. Then, we describe several distance measures that can be used to measure the distance between sequences – the techniques for numeric sequences are Point-to-point mapping and Dynamic Time Warp distance; and those for categorical sequences are Sequence alignment, Longest Common Subsequence, and Ngram analysis. We also describe two clustering algorithms (Agglomerative and HDBScan) and three cluster validation techniques (Silhouette index, DB index, and visual analysis) that have been evaluated in this research.

4

Dataset Exploration

This project aims to develop a clustering technique that is applicable to the real world. This would eliminate the bias towards synthetic malware that models scenarios that might never occur in real life. Therefore, it was decided to work with a security company, which specializes in malware analysis and threat intelligence.

4.1. Dataset collection

ABC has provided the dataset used in this project. The dataset is a collection of malware samples detected by their system in 2017. The dataset is composed of only banking malware because ABC's main clientèle are financial institutions.

The data has been collected as follows: ABC has an infrastructure with which they detect malware samples. If the sample turns out to be a unique one, which they have not analyzed before, they execute it in a sandboxed environment and collect behavioral information. They have a series of different sandboxes so that if a malware is able to detect that it is being executed in a sandbox and refuses to generate behavioral data, they execute it in a different one with changed configurations. The goal is to execute the malware to collect information, such as file system activities, API calls, and the generated network traffic. The sample is executed for a variable amount of time, which is unknown to us. Since only network behavior concerns this project, we only focus on it. Hence, among other kinds of files, the generated network activity is saved in a Pcap file. Each Pcap file refers to one malware sample. The Pcap file is zipped and is renamed to its SHA1 hash, and then is stored in an archive. These hashes are stored in a database corresponding to the entry of its original binary's sample. This database also contains other metadata, such as the time and date when the malware was detected, the malware family label assigned to it, VirusTotal's output, and the reference to the binary executable associated to the Pcap file. ABC graciously provided us with approximately 1.2 Million Pcap files and some of their associated metadata entries.

4.2. DATASET LABELING

ABC has a set of YARA [1] rules to classify malware samples into families. YARA is a tool for malware researchers that helps to identify and classify malware samples. With YARA, one can create a set of rules, which are essentially descriptions of malware families based on their textual or binary patterns. The rules used by ABC are entirely based on the static analysis of binaries of the different malware samples they have analyzed over the years. Malware analysts update rules of a malware family whenever a sample of that family has changed significantly. The labels assigned to malware samples are stored in the database, referenced by the hash of the malware's binary.

4.3. DATASET FILTERING

There are approximately 1.2 Million Pcap files in the dataset, all in a zipped format. Each Pcap file refers to a unique malware sample. In addition, the metadata file contains 47,271 entries associated with malware samples. A script was written to unzip those Pcap files that have a corresponding entry in the metadata file. This step resulted in a little over 47K Pcap files. Each Pcap file had a corresponding malware family label. Each malware family had a variable size. It was decided to focus on a few prevalent and well-known malware families. Crowe [1] reports a list of malware families that pose the highest threat in 2016-2017 [2]. Hence, a total of 15 banking malware families were chosen from those lists for the purpose of this project. 1196 Pcap files are associated with the chosen malware families. The family names and their contribution to the final dataset are given in Table 4.1. The filtered sample set is merely 2.5% of the original dataset, which is not a representative sample set. However, the feature set we have chosen eliminates the potential for bias because of its genericity. Moreover, some chosen families are significantly bigger than the others, which also has the potential of causing skews. However, a pairwise distance matrix is calculated for all samples, which gives each sample in every malware family an equal opportunity. Hence, a malware family's size is not correlated with the clustering results.

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

[2] https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-financial-threats-review-2017-en.pdf

Family name	Samples	Percentage Contribution (%)
Blackmoon	887	74.10
Gozi ISFB	122	10.19
Citadel	70	5.85
Zeus VM AES	29	2.42
Ramnit	22	1.83
Dridex Loader	15	1.25
Zeus v1	10	0.83
Zeus Panda	10	0.83
Gozi EQ	7	0.58
Dridex RAT Fake Pin	7	0.58
Dridex	6	0.50
Zeus P2P	4	0.33
Zeus	3	0.25
Zeus OpenSSL	2	0.17
Zeus Action	2	0.16
Total (N)	**1196**	

Table 4.1: Composition of malware families and their contribution in the experimental dataset

4.4. LEVEL OF GRANULARITY OF SEQUENCES

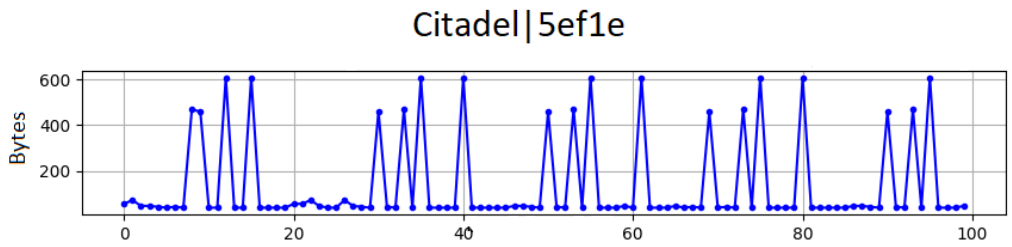
There is much work on network traffic analysis to solve a number of problems, e.g., intrusion detection, anomaly detection, building behavioral models of malware, etc. However, to the best of our knowledge, there does not exist any work that answers how low-level the features need to be in order to capture the behavior of the entity that they are analyzing. We already know from literature survey (see Chapter 2) that there is an emphasis on application layer headers and even on features extracted from the payload. We want to know whether that is the only way forward and whether we can modify the way we build our features in a way that does not go too deep in a packet's content. A discussion with an experienced network analyst from ABC led us to select the potential level of granularities that might capture the malware behavior. Hence, in this project we experiment with sequences generated at two different levels of granularity: 1) Pcap-level, and 2) Connection-level.

In order to understand the granularities of the sequences, consider the following analogy: There is a room filled with people. The host is having a conversation with each of the guests. Your task is to detect whether that room is filled with potentially malicious conversations, e.g., regarding murder or drug dealing. It would be even better if you can identify which of the guests seem suspicious based on the kind of conversation they are having with the host. For example, a drug-dealing expert informed you that specific gestures are often used when negotiating a drug deal. Hence, you should be on a lookout for familiar gestures and flagged keywords.

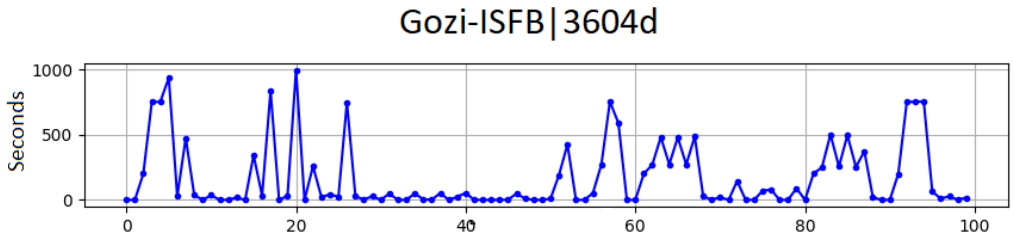
The room in this example is a packet capture (Pcap) file. The host of the party is the localhost, which is represented by the IP address of the virtual machine the malware is executed in. The guests of the party are the IP addresses the localhost connects to. The goal is to detect and group similar malicious conversations/behaviors, without reading the actual payload of the involved packets.

4.4.1. PCAP-LEVEL SEQUENCES

Back to the analogy: One way to monitor the room and its residents is to listen to every word every person says starting from the beginning of the party till the end and flag any suspicious words that you hear. This is equivalent of creating a sequence from every single packet in the Pcap file. This is called the 'Pcap-level granularity'. For example, a sequence of packet sizes would be generated at the Pcap-level by concatenating the packet size of each packet in the sequence. Similarly, a sequence of time-elapsed between packets would be generated at the Pcap-level by concatenating the time-difference between receiving the previous and current packet, starting from the first packet till the last packet of the Pcap. The visual examples of these sequences are shown in Figures 4.1. These figures show the first 100 packets of the Pcap. The figure's title shows the assigned malware family label followed by the first five characters of the SHA1 hash of the Pcap file. In the given dataset of 1196 Pcaps, the average length of Pcap-level sequences is approximately 190 packets.



(a) Sequence of packet sizes measured in bytes)



(b) Sequence of interval between packets measured in seconds

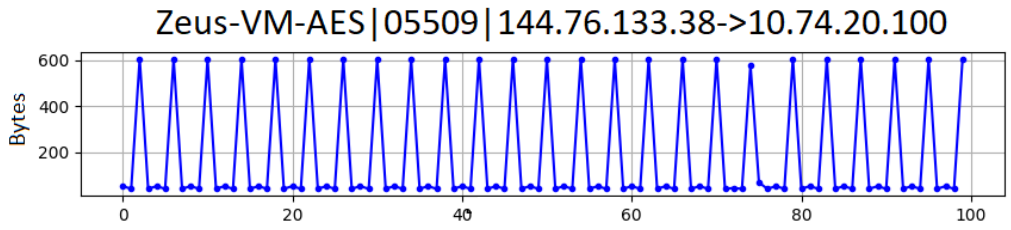
Figure 4.1: Examples of Pcap-level sequences

There are some caveats to using Pcap-level sequences. The resulting sequences are, on average, too short for behavioral models, such as state machines, and take too long to compute distances, such as Dynamic Time Warp distance. Secondly, the way the packets are concatenated to generate sequences jumbles up packets being sent to and received from different hosts. The interference from other connections can hide the subtle patterns present in the way hosts communicate with each other. However, on the positive side, Pcap-level sequences show a bird's eye view of what goes on right after the malware sample is executed.

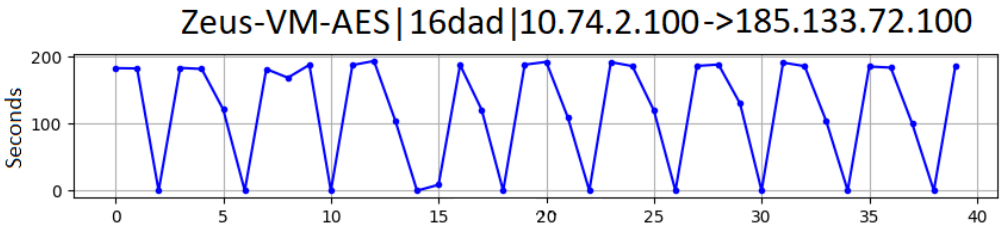
4.4.2. CONNECTION-LEVEL SEQUENCES

Back to the analogy: Another way to spot malicious conversations is to zoom in on each individual conversation, such that the words spoken by the host are considered a separate sequence than the response received from one of the guests. This is equivalent of constructing a sequence *only* from the packets transferred between (source IP → destination IP). This is called the 'Connection-level

granularity'. An *Outgoing connection* is defined as the packets transferred between (localhost → guest IP), where guest IP is the IP address of one of the hosts the localhost connects to. On the contrary, an *Incoming connection* is defined as the packet transfers between (guest IP → localhost), where guest IP is the IP address of one of the hosts that is replying to the localhost. Figures 4.2 show examples of incoming and outgoing connections, respectively. Figure 4.2(a) shows the sizes of the first 100 packets of an incoming connection from 144.76.133.38. Figure 4.2(b) shows the time difference between the first 40 packets of an outgoing connection to 185.133.72.100. In the given dataset of 1196 Pcaps (or 8997 connections), the average length of Connection-level sequences is approximately 20 packets, which is significantly smaller than the average of Pcap-level sequences.



(a) Incoming connection of packet sizes



(b) Outgoing connection of interval between packets

Figure 4.2: Examples of Connection-level sequences

Figures 4.1 and 4.2 show behaviors at different levels. However, at the Connection-level, much more insight can be obtained about the behavior, since we also have access to the host's IP address. Each Pcap file consists of three connections on average, which on the Pcap level are all jumbled up inside a single sequence. Connection-level sequences are smaller than Pcap-level sequences. However, because Connection-level sequences are at a deeper level than Pcap-level sequences, they are able to capture clearer behavioral patterns of individual hosts. It is also much easier to distinguish hosts exhibiting similar behavioral patterns versus distinguishing behavioral patterns in whole Pcap files.

4.5. DATASET SPECIFICATION

For the clustering, we only consider sequences that are longer than a fixed threshold in order to avoid introducing artifacts in the results due to huge variance in the length of the considered sequences. After a conversation with an experienced malware analyst, we hypothesize that if two malware samples have some relationship, e.g., code sharing or being controlled by the same attacker, the first few packets, also known as the *handshake*, transferred to an IP address may be similar. We expect that the similarity in the handshakes will be reflected in higher-level features. However, it is unclear how many packets constitute the handshake. The threshold should be such that it is big

enough to allow the initial handshake to be captured and short enough that it does not take too long to compute distances between sequences. Since there is no straightforward way of setting this value, the average length of the sequences was chosen as the threshold. This value is the average number of packets in a Pcap (or connection) of the whole dataset. Since the dataset is very skewed, most of the sequences are shorter than the selected threshold. In case of Pcap-level granularity, only 129 Pcaps out of the 1196 files are longer than the pre-defined threshold of 190 packets. In addition, there are a total of 8997 connections in the 1196 Pcap files. In case of Connection-level granularity, only 733 connections are longer than the threshold of 20 packets. Hence, the dataset considered for clustering forms 10% of the total Pcaps and 8% of the total connections. A summary of the dataset size is given in Table 4.2.

	Pcap-level sequences	**Connection-level sequences**
No. Pcap files	1196	1196
No. Connections	-	8997
Average length in bytes (*avg*)	190	20
Revised No. Pcap files (>=avg)	129	216
Revised No. Connections (>=avg)	-	733

Table 4.2: Summary of number of sequences at Pcap- and Connection- level granularity

4.6. SUMMARY

In this chapter, we introduced ABC as the data provider for this project. We also described how the dataset was collected and labeled. The provided dataset was initially composed of 1.2M Pcap files, which were filtered down to 1196 files coming from 15 pre-selected banking malware families. In addition, two levels of granularity were selected to generate the sequential features – Pcap-level (generating sequences from each packet in a Pcap), and Connection-level (generating sequences from packets sent from host `A` to host `B`).

5

FEATURE-SET EXPLORATION

In this chapter, the feature-set that we believe characterizes a malware's network behavior has been identified. There are two sets of features, one for each level of granularity. We also provide justifications for why the features were chosen and the differences that exist between feature sets of Pcap-level sequences and Connection-level sequences. In addition, the feature representation for the baseline analysis has also been described. Finally, we explain which distance measures were used to calculate distances between the features.

There are a few considerations for the choice of the feature set. In light of available literature, we want to develop a technique that is generalizable to more than one type of protocol-based malware. Hence, the feature set should allow clustering different type of malware. Secondly, although netflows are more privacy-preserving than Pcaps, they are often sampled so they cannot be used for building sequential behavioral models as some behavior would be missing due to sampling. Moreover, behavioral models such as state machines require long uninterrupted sequences to provide a reasonable statistical distribution of data. Since the majority of the samples in the provided dataset have short lengths, reliable state machines cannot be built. Therefore, the feature set should not rely on excessively long sequences and must capture the uninterrupted behavior of malware. Lastly, since the project's emphasis is on banking malware, following are some of the features commonly present in a banking malware, and that can be seen in its network traffic:

1. Presence of Command and Control server to exfiltrate the data to, and to receive the commands from. This will be indicated by contacting the same IP address multiple times. To verify this, threat intelligence databases, such as VirusTotal [1], threat Crowd [2] and Threat Miner [3] can be checked for reports related to that IP.

2. Data theft/exfiltration – in particular, theft of credentials or confidential files. This will be indicated by repetitive big packets being sent from the localhost to a suspicious IP address.

3. Receiving attack commands or configuration updates from the Command and Control server. This will be indicated by receiving big packets from a suspicious IP address.

[1] https://www.virustotal.com
[2] https://www.threatcrowd.org/
[3] https://www.threatminer.org/

4. Heartbeat packet being sent to the C&C communicating that the infected host is still alive and part of the botnet. This will be indicated by periodic small packets being sent to a suspicious IP address.

5. Fraud can be potentially visible in the anomalous network activity being generated from an infected host. Banking malware is now moving towards employing hosts to mine cryptocurrency. Most of the behavior (read: actual mining) is done offline, but they report back to the C&C server to consolidate intermediate results. This should potentially be visible in the payload or the request query parameters.

Other behaviors such as spying, integrity violation, and persistence are capabilities that are done offline so they cannot be seen in the network traffic. However, a feature set is required that is capable of capturing these behaviors, without having to look inside the payload.

5.1. SELECTED FEATURES

The inspiration for the feature set came from the IO-graph utility in Wireshark [4] – a tool to capture and analyze network traffic. An example of an IO-graph is shown in Figure 5.1. The x-axis shows the time, and the y-axis shows the number of packets/bytes sent/received. The IO-graph shows a graphical representation of the high-level behavior exhibited by a Pcap file. If a botnet has a periodic behavior, it will be shown in the IO-graph as peaks in the graph at regular intervals. We can further drill down in the IO-graph by showing only the traffic transferred between two hosts or traffic using a specific protocol.

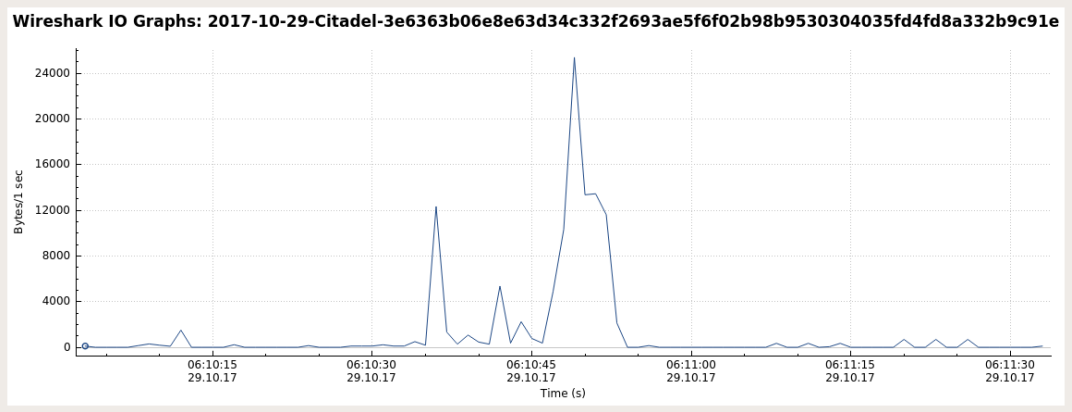


Figure 5.1: An example IO-graph showing behavior of a Citadel malware sample

Hence, the initial feature set was the packet sizes and the interval between packets. However, we quickly realized that these features were too high level to show any interesting malicious behavior. In addition, the IO-graph shows sequential behavior aggregated on time basis (e.g., per second or millisecond). This causes the sequences to lose detailed information. For example, it will show that 500 bytes (or 3 packets) per second were sent, but will not show that the first packet contained all the 500 bytes. Considering this to be crucial behavioral information, the interval in the chosen feature set was calculated on a per-packet basis instead. Next, the protocol number utilized by the packets was also added in the feature set. The intuition was that if the malware samples are related, then they would switch protocols in a similar order. In addition, as part of the literature review, a number

[4] https://www.wireshark.org/

of features in the Network-layer and the Transport-layer header were identified that were promising enough to characterize a malware's network behavior. A few of them were port numbers, Time to Live (TTL), and packet fragmentation flags.

One important note is that these features may not distinguish malicious from benign behaviors. The abstractness provided by the high-level feature set cannot guarantee the difference between benign and malicious behaviors. Therefore, the presence of a human analyst is still required to sift through the resulting clusters. The goal of these clusters is to show similar behaviors. Those behaviors can all be benign, malicious or can also be a malware behaving like a benign entity. No matter what the ground truth is, the resulting clusters will provide actionable intelligence on which malware samples behave similarly and what that similar behavior is.

5.1.1. PACKET SIZES

The first chosen feature is the size of the *IP datagram* of each packet. The intuition is that the usage of certain packet sizes and the periodicity in the usage of similar packet sizes might indicate similar underlying infrastructure. Of course, there is a cap on how big a datagram can be. However, if a packet size is used frequently, which does not appear in other traffic, for example, or if similar packet sizes are being sent/received at regular intervals, they might indicate suspicious behavior. In addition, if big packets are being sent frequently, it might indicate exfiltration, and if big packets are received frequently, it might indicate the dropper file is downloading the malware payload.

As a motivating example, upon visualizing the content of Pcap files in the provided dataset, we find the following case: A malware sample visits a host and downloads a .dmg file (an image file for MacOS), while another sample visits a different host and downloads an .iso file (an image file for Windows). Both of these malware samples have been labeled as Gozi-ISFB, but they do not necessarily have to be labeled the same to exhibit similar behavior – any malware sample can be programmed to contact a host and download an image file. The crux is that the timings of the requests these samples send, and the responses they receive are similar. Although the exact hosts contacted, the exact file downloaded and the content of the packet itself is different, the semantics seem similar. On top of that, this similarity is also reflected in the high-level features of packet sizes. An example of a sequence of packet sizes is shown in Figure 5.2. The y-axis shows the packet size in Bytes, while the x-axis shows the packet number. The figure shows the use of a 300-byte packet after every six packets.

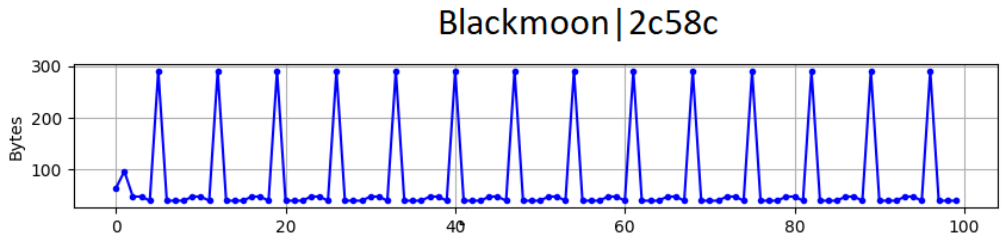


Figure 5.2: An example sequence showing packet sizes in bytes

5.1.2. INTERVAL BETWEEN PACKETS SENT/RECEIVED

This feature captures the *time interval between the previous and the current packet.* Most of the packets are sent/received with less than a second's interval. There are a few packets that are sent after 10-15 seconds but that behavior is rarely seen. If the interval is measured with a smaller unit,

such as in microseconds, the interval may get polluted with noise introduced due to network delays. Therefore, after experimentation, we measured the interval in *milliseconds* rounded off to three decimal places. The intuition for this feature is similar to that of packet sizes – if two malware samples report to the same C&C or the same attacker, the way the malware is programmed would be similar, and that should be visible in the timing of when the malware becomes active and when it goes to sleep. In addition, heartbeat packets [5] are also sent periodically to inform the C&C server that the infected host is still part of the botnet. An example of a sequence generated with time interval is shown in Figure 5.3. It shows that the malware is much slower in communicating with its C&C at the start of the capture. It also shows a periodic switch between faster packets and slower packets.

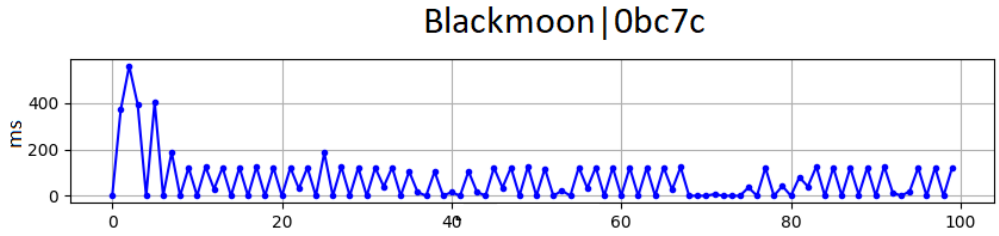


Figure 5.3: An example sequence showing intervals in milliseconds

There are a few caveats to using this feature: Time intervals of Pcaps are comparable only if the two Pcaps were collected on the same network, in order to avoid introducing network-related artifacts. In addition, malware can only exfiltrate information when that information becomes available. Hence, the malware may remain asleep for a variable amount of time before becoming active. However, as soon as malware is installed on a victim's machine, it needs to establish contact with its C&C server to notify about the newly-infected host. This initial contact, which we define as the *handshake*, is expected to remain similar for different infected hosts. Hence, only the interval information available during the handshake is usable for clustering similar behaviors.

5.1.3. PROTOCOL

The protocol number field in the IP header shows the type of protocol used by the packet. IANA [6] has standardized a protocol number for each protocol ranging from protocol number 0 (HOPOPT) to 254 (used for experimentation). Protocol number 255 is reserved by IANA. Some protocol numbers for well-known protocols are: 1 (ICMP), 4 (IPv4), 6 (TCP), and 17 (UDP). The intuition for choosing the protocol number is that if attackers use unconventional protocols to hide their actions, it will be detected. In addition, if two malware samples are controlled by the same attacker, they would switch between protocols in a similar way. This feature also makes the clustering technique protocol-agnostic and will be able to cluster different types of malware. An example of protocol switching behavior is shown in Figure 5.4, which shows that the packets periodically switch between TCP and UDP.

[5] https://www.ixiacom.com/company/blog/mirai-botnet-things
[6] https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

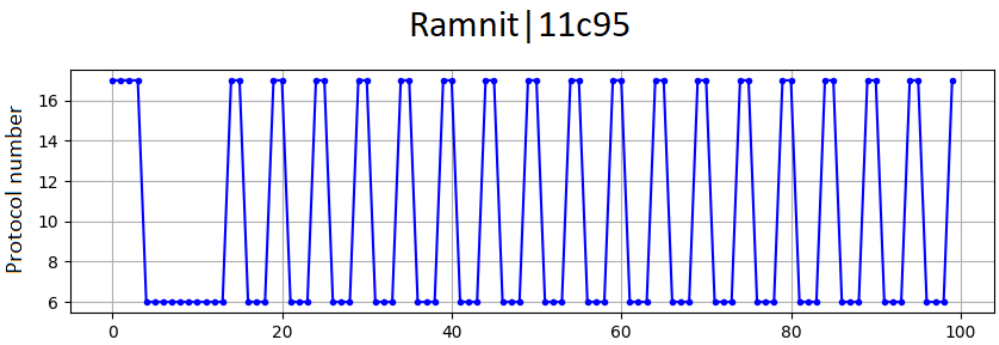


Figure 5.4: An example sequence showing protocol switching behavior

5.1.4. PORT NUMBERS

If IP addresses are identifiers of a host, then the port numbers can be considered as the 'doors' that hosts use to communicate with the outside world. Port numbers are part of the Transport layer header. Some port numbers of well-known services are: 22 (SSH), 80|8080|8088 (HTTP), 443 (SSL), 53 (DNS). Port number 0 is reserved for requesting dynamic ports in socket programming [7]. The intuition behind using this feature is similar to the ones above. When exploring this feature, a number of interesting behaviors were identified. For example, we could identify the protocol the malware was most likely operating on, e.g., Figure 5.5(a) shows an HTTP-based malware and Figure 5.5(b) shows a DNS-based malware. We could also identify potentially malicious behavior based on the usage of certain port numbers. For example, ports 0 to 1023 are well-known ports that are used by system processes. Ports 1024 to 49151 are registered port numbers for specific services, and ports 49152 to 65535 are private port numbers that can be used for customized services. Services running behind some port numbers are vulnerable[8], which makes the use of certain port numbers suspicious.

For the feature set, two sequences were created – one for source ports and the other for destination ports.

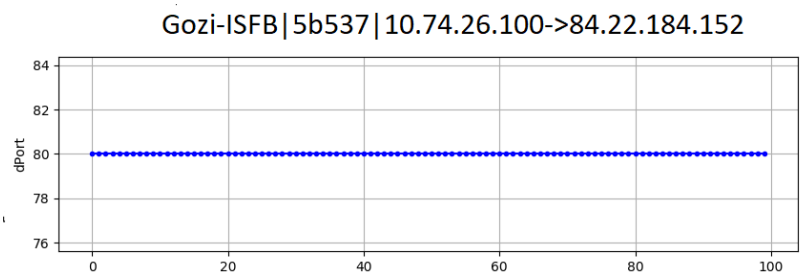
5.1.5. FEATURES THAT WERE DISCARDED

A number of different features were evaluated, which we believed could characterize a malware's network behavior. Following are some features that were not useful in describing a malware's network activity.

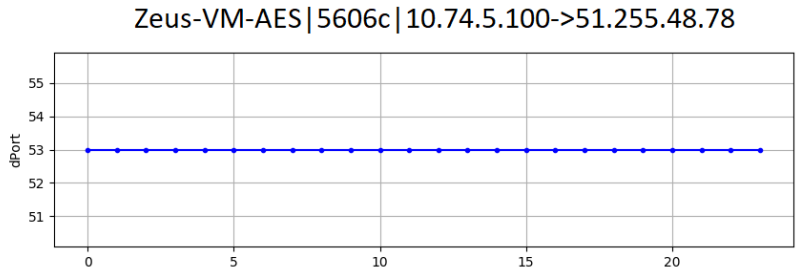
IP ADDRESSES

We chose port numbers instead of IP addresses since IP address can be considered Personally Identifiable Information, especially in countries like the Netherlands where most of the hosts have a fixed IP address. In addition, if we use IP address as a feature, multiple C&C servers controlled by the same attacker, each identified by its own IP address, will not be grouped together.

[7] https://www.lifewire.com/port-0-in-tcp-and-udp-818145
[8] https://www.dummies.com/programming/networking/commonly-hacked-ports/



(a) An example sequence showing HTTP-based malware



(b) An example sequence showing DNS-based malware

Figure 5.5: Visualizing HTTP- and DNS- based malware using port numbers

TIME-TO-LIVE (TTL) VALUE

During literature review, we came across the study by Yamada and Goto [51] in which they claim that TTL value is an excellent indicator of malicious activity. However, when we experimented with the TTL value, we found that it is always outside the 'malicious value range' presented in the paper. It can either indicate that the malware samples we are analyzing are too sophisticated to use TTL value to exfiltrate information or that this feature used as a sequence does not capture dynamic behavior. In either case, this feature was dropped after seeing their distribution. An example is shown in Figure 5.6, where the TTL value oscillates between 51 and 52 hops, both of which fall in the safe range.

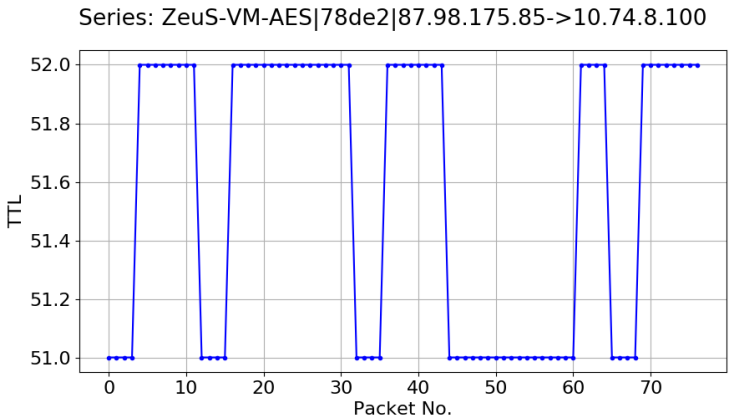


Figure 5.6: An example sequence showing TTL values. This feature was discarded because it always stays in the safe range

DNS QUERY

After reading the study by Pomorova *et al.* [52], we could use DNS query types as features for clustering DNS-based malware. However, because it limited our capability to handle other types of malware and because our dataset consisted of only a limited number of DNS-based malware samples, we decided to drop this feature as well.

5.2. FEATURES FOR PCAP-LEVEL SEQUENCES

The features chosen in the previous section were used to generate sequences on the Pcap-level granularity. However, there were a few features that did more harm than good for the characterization of the malware's network behavior at this granularity.

We chose three sequences of `(packet size, interval, protocol)` to represent the network-level behavior of malware at the Pcap-level granularity. A summary of the feature set used at the Pcap-level is presented in Table 5.1.

The port numbers had an excessive amount of noise, which did not show any interesting patterns that would have helped us in characterizing the malware's network activity. We believe that the noise is introduced as a result of values from different connections getting jumbled up together. An example of the sequence of source port numbers at the Pcap-level is shown in Figure 5.7. The figure does show some periodic behavior, but almost all the Pcap files generate the same kind of behavior.

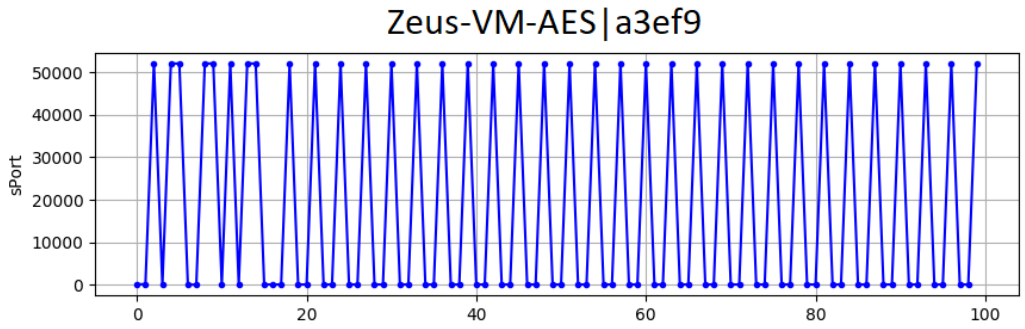


Figure 5.7: An example sequence showing source port numbers. It was discarded because of excessive noise

5.3. FEATURES FOR CONNECTION-LEVEL SEQUENCES

When a connection is established, it usually uses only a single protocol for communication. This is why at the connection-level, the sequence generated by protocol numbers remains constant. Moreover, it does not provide any additional useful information that the port numbers do not already provide. For example, if a host is communicating over HTTP, there would be a sequence of 80's as the source port feature, while the protocol would be a constant 6 for TCP, which does not provide any additional information because we already know that HTTP uses TCP on the Transport layer. An example of a sequence of protocols at the connection-level is shown in Figure 5.8. Hence, we dropped protocol number as a feature and instead used a quadruple of `(packet size, interval, source port, destination port)` to characterize malware's network activity at the Connection-level granularity. A summary of the feature set used at this level is presented in Table 5.1.

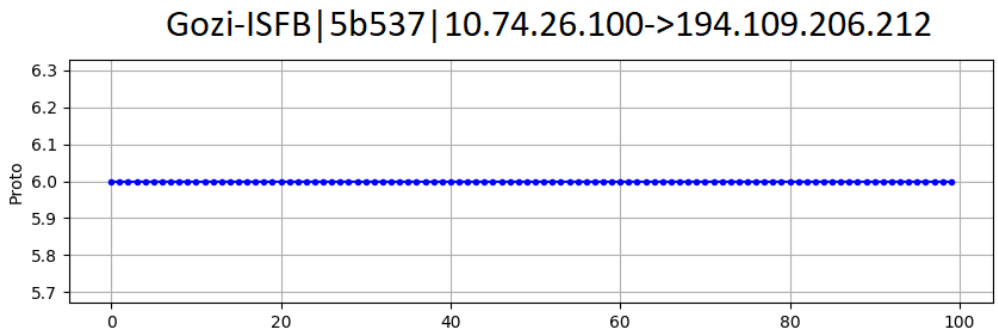


Figure 5.8: An example sequence showing protocol numbers at Connection-level. It was discarded because of its constant behavior

5.4. DISTANCE MEASURES

Once we have decided the feature set that will be used to characterize the malware's network behavior, a distance measure is needed that can, appropriate to the requirements, measure the distance between two sequences. In this section, we describe which of the distance measures from Section 3.2 have been used to measure the distance between sequences.

For numeric sequences – packet size and interval, we have used Dynamic Time Warping (Section 3.2.2) to calculate the distance between sequences. As mentioned in Section 3.2.1, the point-to-point mapping method is too stringent to accommodate noise or delays, which are common in network traffic.

For categorical sequences – protocol numbers, source and destination ports, we have used Ngram analysis (Section 3.2.5) to transform the sequences into their vector representation and then we have used cosine distance to measure the distance between two vectors. The value of n is set to 3 as an experimental value. We also experimented with Sequence alignment (Section 3.2.3) and Longest Common Subsequence (Section 3.2.4), which not only gave uninterpretable results but were also very slow in computing the distances. In addition, we also experimented with discretizing numeric sequences (Chapter 6). In that case, we also use Ngram analysis to measure the distance between the discretized packet sizes and interval sequences. A summary of the distance measures used at both the Pcap-level and Connection-level is presented in Table 5.1.

At both the Pcap-level and the Connection-level, multiple sequences represent a malware sample. The distance measure from each feature needs to be consolidated into a single number that represents the distance for the whole Pcap/connection. Hence, each distance value is scaled to the range [0-1]. Then, a simple unweighted average is used to combine all the distances. Although unweighted average has its limitations, it works well for the proposes of this research. Hence, for Pcap-level, the final distance measure is calculated as:

$$d_{pcap}(a,b) = \frac{d_{packetSize}(a,b) + d_{interval}(a,b) + d_{proto}(a,b)}{3} \tag{5.1}$$

where a and b are two Pcap files; $d_{pcap}(a,b)$ is the final calculated distance between a and b; $d_{packetSize}(a,b)$ is the distance between the sequences of packet sizes of a and b; $d_{interval}(a,b)$ is the distance between the sequences of intervals between a and b; $d_{proto}(a,b)$ is the distance between the sequences of protocols between a and b.

And the final distance measure for Connection-level is calculated as:

$$d_{conn}(a,b) = \frac{d_{packetSize}(a,b) + d_{interval}(a,b) + d_{sourcePort}(a,b) + d_{destPort}(a,b)}{4} \tag{5.2}$$

where a and b are two connections; $d_{conn}(a,b)$ is the final calculated distance between a and b; $d_{packetSize}(a,b)$ is the distance between the sequences of packet sizes of a and b; $d_{interval}(a,b)$ is the distance between the sequences of intervals between a and b; $d_{sourcePort}(a,b)$ and $d_{destPort}(a,b)$ are the distances between the sequences of source ports and destination ports between a and b, respectively.

	Features	Feature set	Data type	Distance measure
Pcap-level	3	Packet sizes	Numeric | Categorical (Trigrams) Sequences	DTW | Cosine
		Interval	Numeric | Categorical (Trigrams) Sequences	DTW | Cosine
		Protocol	Categorical (Trigrams) Sequences	Cosine
Connection-level	4	Packet sizes	Numeric Sequences	DTW
		Interval	Numeric Sequences	DTW
		Source port	Categorical (Trigrams) Sequences	Cosine
		Destination port	Categorical (Trigrams) Sequences	Cosine
Pcap-level Baseline	3	Packet sizes	Float	Absolute distance
		Interval	Float	Absolute distance
		Protocol	Unigrams	Cosine
Connection-level Baseline	4	Packet sizes	Float	Absolute distance
		Interval	Float	Absolute distance
		Source port	Unigrams	Cosine
		Destination port	Unigrams	Cosine

Table 5.1: Summary of the feature set, their representation and distance measures for Sequence and Baseline clustering

5.4.1. BASELINE FEATURES AND DISTANCE MEASURE

There does not exist a direct baseline, with which the results of this research can be compared. Hence, a synthetic baseline was created to evaluate the effectiveness of using sequences-as-features. The feature set used for both granularities was kept the same. However, the way they were represented and the distance calculation was modified to remove any order-related information. The baseline aggregates the sequences into singular values, which are used as features instead.

For numeric features – packet sizes and the interval between packets, the average of the sequence is taken. For example, the sequence of my coffee-intake throughout the week [1,2,3,4,1,1,1] is represented as an average of 1.8 coffees. The distance between the two features is then the absolute distance between the averages.

For categorical features – protocol, source and destination ports, trigrams were used to capture order-related information. Hence, for the baseline, unigrams (n=1) are used instead. The rest of the

process is kept the same – converting the unigram profiles into vectors and measuring the distance between two features using cosine distance.

The distances at both granularities were calculated using Equations 5.1 and 5.2. A summary of the feature set representation and the distance measure for the baseline are shown in Table 5.1.

5.5. SUMMARY

In this chapter, we described the feature set that, according to us, best characterizes malware's network activity at the two levels of granularity under consideration. Four sequences-as-features are selected in general: 1) Packet sizes measure the size of IP datagram in bytes, 2) Interval between packets measure the time difference between the previous and current packet in milliseconds; 3) Protocol number, and 4) Source and Destination port numbers. IP addresses, TTL values, and DNS queries were discarded as features.

At the Pcap-level, a triple sequence of (packet sizes, interval between packets, protocol number) is used to characterize malware families. At the Connection-level, a quadruple sequence of (packet sizes, interval between packets, source ports, destination ports) is used to characterize malware families. Dynamic Time Warping distance is used to measure the distance between numeric sequences. Ngram analysis and cosine distance are used to measure the distance between categorical sequences. For the baseline version, all sequence-related information is removed from the features. So, the numeric features are represented with the average of their sequence and distance is measured using the absolute distance between two averages. Categorical features are represented using unigrams, and the cosine distance is used to measure the distance between two unigram profiles.

6

SEQUENCE DISCRETIZATION

This chapter introduces three discretization techniques that can be used to convert numeric sequences into categorical ones. Since this project deals with sequences-as-features, there is a need to define a notion of 'similarity' among them. There exist different distance measures, as described in Section 3.2, where some are applicable on numeric sequences and some on categorical ones. In order to evaluate different types of distance measures, it is essential to convert the input sequence into the applicable data type.

In Machine learning, many clustering algorithms only operate on numeric features, since calculating distances between numeric features is simpler and more efficient. Hence, categorical features may be converted into numeric ones by using One Hot Encoding. *One Hot Encoding* is a technique to convert categorical features into binary vectors by creating an individual feature corresponding to each category and assigning 1 to data points that have that category and 0 to data points that do not [1]. Since the features we are dealing with are sequences themselves, applying One Hot encoding to them would result in multiple binary sequences corresponding to each category. The higher the number of sequences per malware sample, the more time it would take to compute the distance between them.

On the other hand, some distance measures are only applicable to categorical sequences, and if numeric sequences are used as-they-are, the results become uninterpretable. For example, the distance between the numeric sequences [1,2,3,60,70,80,90,100] and [4,5,6,61,72,82, 91,102] should not be very large, since the values are very close together, as shown in Figure 6.1.

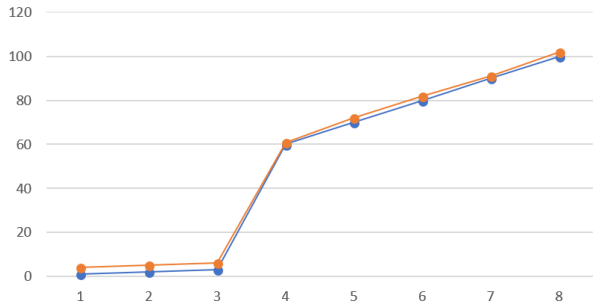


Figure 6.1: An example of a numeric sequence being treated as a categorical one

[1] https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

However, if a technique such as Ngram analysis is used, which is meant explicitly for categorical sequences, it considers a different symbol as a different category. Hence, there is a very high distance between the two sequences, since each number (e.g., 1, 2, 60, 61, etc.) will be considered a separate category. Therefore, numeric sequences need to be converted into categorical ones before such techniques can be used. The process is relatively easier – you define a number of categories and assign each numeric value in the sequence to a category based on certain predefined thresholds. The real challenge lies in deciding what those thresholds should be and in computing them very quickly and efficiently. One straightforward way to determine the minimum and maximum values of a sequence and calculating thresholds based on those. For example, for a sequence of packet sizes, the Maximum Transmission Unit (MTU) of an Ethernet packet is 1500 bytes, and the smallest IP datagram size is 20 bytes for its smallest possible header (the maximum header size can go up to 60 bytes). So, one can simply divide this range using Equal-Width Interval Discretization (Section 2.4). However, this method would only be applicable for sequences that are uniformly/normally distributed in the whole range. For skewed datasets, one category will contain significantly more elements than the others. Since the thresholds chosen are based on the theoretical values of the packets rather than what is actually used in the dataset, the results may not be as intuitive. Therefore, as simple as this method is, it cannot be applied to the cases where thresholds need to be determined from the underlying dataset. In the next section, we describe three light-weight techniques that can help in deciding the threshold values based on the dataset itself.

Assumptions

There are two numeric features under consideration – packet size (measured in bytes) and the interval between packets (measured in milliseconds).

We assume three categories for each of the two features:

1) Packet sizes can be categorized into Small, Medium and Large packets. Smaller values of this feature would represent small packets, while larger values would represent large packets.

2) Intervals can be categorized into Slow, Medium and Fast packets. Smaller values of this feature would represent fast packets, while larger values would represent slow packets.

We choose these three categories as a proof-of-concept for the applicability of the proposed techniques. In practice, if the goal is to discretize sequences in a more fine-grained way, more categories can be introduced, e.g., five categories can be defined, such as (Very small, Small, Medium, Large, Very large).

A note on Mean versus Median

In order to select the thresholds that take into consideration the underlying data distribution, it needs to be applicable to the kind of distribution. For example, if the dataset is highly skewed, mean should not be used because mean is susceptible to skews – a very high value within a sequence of low values can unnaturally inflate the mean value. The dataset used in this project is very skewed as some values are extremely small and some form huge spikes. So mean is not the right measure. Median, on the other hand, only looks at the center of observations, not what an observation's value is like. Median is not an optimal measure because it can also get skewed in the presence of some values that occur too frequently (a common characteristic of a sequence of time intervals, for example). To sum up, one needs to be aware of the limitations of these commonly used measures before using them.

6.1. DISCRETIZATION TECHNIQUES

We propose three discretization techniques:

1. Local percentile method
2. Peak analysis method
3. Global set-percentile method

Below we describe each of the method, their working example, and their pros and cons. To help the reader reason about the proposed techniques, consider the following t-shirt size-labeling problem as a relevant analogy:

You are given a pile of t-shirts and are asked to label the shirts in one of the following categories [Small (S), Medium (M), Large (L)]. Do you label them based on how prevalent a range of values is, or purely based on their absolute value? Would it make sense to assign both small and medium to one close range of values just because it was the most prevalent? Would you assign the categories as soon as you see the shirt or would you collect all the possible values and then choose the categories such that when you put the small shirts next to each other, they are comparable? There are multiple correct answers to this problem depending on what the goal is. Each of the proposed technique handles this issue differently.

6.1.1. LOCAL PERCENTILE METHOD

The local percentile method has been borrowed from the work of Pellegrino *et al.* [41]. For a given sequence, the 33^{rd} and 66^{th} percentiles of the sequence are calculated as the thresholds for the categories.

WORKING EXAMPLES

Consider the following sequences, where assume that they have been reordered in ascending order to aid visualization:

1) [1,1,2,2,3,3,4,4,5,5,6,6]. The 33^{rd} percentile is 2 and the 66^{th} percentile is 4. Hence, for a value v in the sequence, if v <= 2, then it is Small; if v > 2 but v <= 4, then it is Medium; and if v > 4, it is Large. So, the sequence is transformed into: [S,S,S,S,M,M,M,M,L,L,L,L].

2) [1,1,1,1,1,4,4,5,7,7,700]. The 33^{rd} percentile is 1 and the 66^{th} percentile is 5. Hence, for a value v in the sequence, if v <= 1, then it is Small; if v > 1 but v <= 5, then it is Medium; and if v > 5, it is Large. So, the sequence is transformed into: [S,S,S,S,S,M,M,M,L,L,L].

3) [1,98,150,151,152,400]. The 33^{rd} percentile is 98 and the 66^{th} percentile is 151. Hence, for a value v in the sequence, if v <= 98, then it is Small; if v > 98 but v <= 151, then it is Medium; and if v > 151, it is Large. So, the sequence is transformed into: [S,S,M,M,L,L].

4) [1,1,1,1,1,1,1,1,1,1,1,1,10,11,12,13,1000,1000,1000]. Both the 33^{rd} and the 66^{th} percentile are 1. Hence, for a value v in the sequence, if v <= 1, then it is Small; if v > 1 but v <= 1, then it is Medium; and if v > 1, it is Large. So, everything over 1 goes directly into the Large category and the sequence is transformed into: [S,S,S,S,S,S,S,S,S,S,S,S,L,L,L,L,L,L].

5) [5,5,5,5,5,5,5,5,5,5,20,20,20,20]. Both the 33^{rd} and the 66^{th} percentile are 5. Hence, for a value v in the sequence, if v <= 5, then it is Small; if v > 5 but v <= 5, then it is Medium; and if v > 5, it is Large. So, everything over 5 goes directly into the Large category and the sequence is transformed into: [S,S,S,S,S,S,S,S,S,L,L,L,L].

DISCUSSION

When the data is normally distributed (i.e., in example 1), the local percentile method works well. The values are well-distributed among the categories. However, in anomalous traffic, such as that of malware, there are a lot of traffic fluctuations. This creates a skewed dataset of values either larger than normal, i.e., in example 2, or values too small, i.e., in example 3. Secondly, if the sequence is such that a few values repeat too often, and the other values with large differences do not occur as much, this method groups all the rare ones in a single category, i.e., in example 4. Lastly, if the sequence is too fragmented, such that the thresholds end up being the same, one category will always be skipped. Hence, the resulting thresholds are prone to noise and may overshoot/undershoot categories based on the percentile (median-based) values.

Local percentile method misses the global perspective. Packets that are categorized as small that come from different sequences might not be in the same range if their sizes are compared. One sequence might only have packets in range, e.g., 1000-9999, so a packet of size 1000 is labeled small, and the other sequences might only have packets in range, e.g., 5-100, so a packet of size 1000 overshoots even the large category. Since the categories were assigned on a local perspective, we are forced to consider a packet of size 1000 and 5 as equally small.

On the positive side, this method may help retain the local structure of a sequence. For example, one sequence with packets in range 0-100 bytes and another with range 0-1000, both having the same structure, such as a similar interval of sending packets, will end up having similar discretized sequences. Consequently, this will help cluster sequences having similar local substructures together.

6.1.2. PEAK ANALYSIS METHOD

The ideal use-case involves similar malware performing similar actions ending up getting grouped. In certain cases, when the dataset is extremely fragmented, the local percentile method does not make sense anymore because it works on medians. We define *fragmented datasets* as having a few values being repeated multiple times and being far apart from each other. Therefore, it would make sense to look at the frequency of each value in the sequence. This method is called the *peak analysis method*. In this method, a frequency table of the values occurring in a sequence is constructed, and the top three most frequent values are collected – these form the highest peaks, and because the dataset is fragmented, these peaks are far apart from each other. Next, the three frequent values are ordered in ascending order, and the average between subsequent peak values is calculated. If $val1$, $val2$, $val3$ are the top three frequent values, then:

$$thresh1 = \frac{val1 + val2}{2} \tag{6.1}$$

$$thresh2 = \frac{val2 + val3}{2} \tag{6.2}$$

In case three peaks are not available in the data, this method simply falls back to the local percentile method.

WORKING EXAMPLES

Consider the following sequences, where assume that they have been reordered in ascending order to aid visualization:

1) [2,2,2,5,5,5,5,10,10,10,10,100,400,800]. The top three frequent values are: $val1 = 2$; $val2 = 5$; $val3 = 10$. The thresholds will be: $thresh1 = 3.5$; $thresh2 = 7.5$. Hence, the sequence will be transformed into: [S,S,S,M,M,M,M,L,L,L,L,L,L,L].

2) [10,11,12,13,13,43,43,50,80,700,700,900,1000]. The top three frequent values are: $val1 = 13$; $val2 = 43$; $val3 = 700$. The thresholds will be: $thresh1 = 28$; $thresh2 = 371.5$. Hence, the sequence will be transformed into: [S,S,S,S,S,M,M,M,M,L,L,L,L].

3) [0,0,0,5,5,5,5,5,7,7,9999]. The top three frequent values are: $val1 = 0$; $val2 = 5$; $val3 = 7$. The thresholds will be: $thresh1 = 2.5$; $thresh2 = 6$. Hence, the sequence will be transformed into: [S,S,S,M,M,M,M,M,L,L,L]

4) [5,5,5,5,5,5,5,5,5,5,20,20,20,20]. Because there are only two frequent values – 5 and 20, the method falls back to the local percentile method and the result is the same as the discretized sequence shown in example 5 of Section 6.1.1.

DISCUSSION

This method allows for the categories to be assigned based on the prevalence of values rather than statistics. This technique is different from the other two techniques because it is purely frequency-based. In the t-shirt analogy, this represents the scenario where one assigns the categories based on how frequent the sizes are – if shirts of sizes 2, 3, and 10 are most frequent, then the categories S and M end up being closer to each other than category L. The intuition behind this technique is that because a value is so frequent, it must mean something interesting so, that value must get its own category. On the other hand, the rare but distant events will all get crammed into another category. It can be useful if the goal is to assign categories based on the prevalence of a close-by range.

This method works well when there exist nearly-fragmented values in the dataset (frequently occurring values that are far apart from each other). Other rare but similar values would get absorbed in the nearest category of frequent items. On the other hand, this technique does not work on, for example, a dataset where values exist closely or where all values are frequent, or none is frequent. For example, this method would not work on an almost normally distributed dataset.

In general, at least one category will be full of similar values, which may be considered uninteresting depending on the context. This technique is value-agnostic, meaning if the dataset looks like example 3, then 0 gets a whole category, 5 gets the second category while the 7's and 9999 get crammed into one. Whereas, one might have wanted to see 9999 get another category given the assumption that rare events are not noise but rather malicious, and hence interesting.

The local percentile and peak analysis methods both operate locally on the sequences. This means that the discretized sequences may not be comparable to other sequences based on their actual values, e.g., a small packet of one sequence may not always be in the same range as a small packet of another sequence.

6.1.3. GLOBAL SET-PERCENTILE METHOD

A global standpoint is when all possible values occurring in the whole dataset are collected in an initial pass, and the thresholds are calculated on that list. This gives a fixed threshold for the whole dataset, meaning that if a value is considered small in one sequence, the other values considered small in other sequences would surely lie in the same range. This helps standardize the discretized sequences. In addition, a potential scenario is when some values occur so frequently that they become uninteresting. They also affect the median by skewing it unnaturally towards the frequent values. This is the *frequency problem* – the most frequent items overshadow rare but interesting items. To mitigate this problem, all possible values in the entire dataset are read, and only the unique ones are collected. Next, the 33^{rd} and 66^{th} percentile values of that *global set* are calculated as the thresholds. Hence, this method becomes frequency-agnostic.

WORKING EXAMPLES

Consider the following examples with a global set of [10,13,20,22,30,33,40,44,50,1,2,3,4,5,6,7,9,11,15, 17,19,21,23,38,45]; the 33^{rd} percentile is 9.33 and the 66^{th} percentile is 22.6:

1) [10,10,20,20,30,30,40,40] would transform into [M,M,M,M,L,L,L,L]

2) [10,13,20,22,30,32,40,44] would transform into [M,M,M,M,L,L,L,L]

Also, consider the following examples with a global set of [0,1,2,3,4,10,20,30,40]; the 33^{rd} percentile is 2.33 and the 66^{th} percentile is 16.6:

3) [0,0,1,1,2,2,3,3,4,4] would transform into [S,S,S,S,S,S,M,M,M,M].

4) [10,10,20,20,30,30,40,40] would transform into: [M,M,L,L,L,L,L,L].

DISCUSSION

As time passes by and additional sequences are observed, the global set gets enriched with additional values, and it starts becoming more normally distributed, which helps exploit the full use of the percentiles. The sequences become globally comparable in a more objective way. In addition, the global thresholds help dampen the noise from the sequences and make them cleaner. For example, in the case of examples 1 and 2, the global thresholds have provided tolerance and have smoothed out the small bumps in values. If local percentiles were being considered here, the structure of examples 1 and 2 would not have been the same.

On the contrary, the positive point of local percentile method becomes negative in the global set-percentile method. Consider an example of similar local substructures, as shown in examples 3 and 4. The similar local structure of the sequences present in these examples is lost because the thresholds are calculated globally.

6.2. DISCRETIZING PCAP-LEVEL SEQUENCES

At the Pcap-level granularity, the sequences are long and are composed of each packet in the entire Pcap file. Longer files allow for a variety of different behaviors (and values) to be contained in a single sequence.

There are strong points, both for and against each of the three discretization techniques that

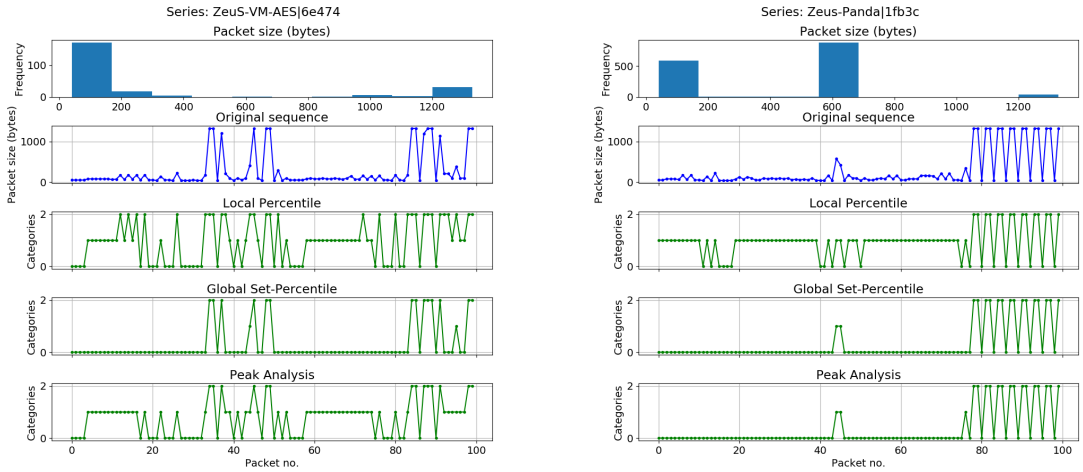
have been explained above. However, based on the data distribution of the two numeric features – packet sizes and the interval between packets, two different techniques would be applicable for them.

Packet sizes are discretized using the Global Set-Percentile method and time intervals using Peak Analysis method. In the following sections, we elaborate further on why they are the optimal choices:

6.2.1. DISCRETIZING PACKET SIZES

The feature of packet sizes represents the size of the IP datagram of each packet. These values can typically range from 20 bytes to 1500 bytes, which is the MTU for Ethernet. Upon visualizing the dataset, it was found that the values are not normally distributed within this range – the values are skewed towards packet sizes smaller than 100 bytes. In addition, there are local groups of values that lie very close to each other. Hence, clear fragmented peaks are not always available.

With packet sizes, the goal is to find significant variations in the sequences because they are interesting and we do not want them to get overshadowed because of noise generated from frequently occurring values – anomalous packet sizes could indicate malicious activity, and similar-occurring transitions in packet sizes could indicate similarity in malicious behaviors. In addition, we do not want small differences in packet sizes to get overemphasized and be considered in a different category. Figures 6.2 show two concrete examples of sequences of packet sizes, where the result of discretization using global set-percentile thresholds performs the best. The figure shows four series and a histogram. The title shows the malware family label assigned to the Pcap file followed by the first five characters of the Pcap's hash. The histogram shows the data distribution for the whole sequence. The first series (blue) shows the first 140 values of the sequence. The other three series (green) represent the three discretization techniques – local percentile, global set-percentile, and peak analysis method. The focus of the reader should be on comparing which of the three green series represents the blue series in the best way, and also note the histogram distribution, which also guides the choice of the best method to select.



(a) Example 1

(b) Example 2

Figure 6.2: Comparing discretization techniques for packet sizes at Pcap-level granularity

For the dataset under consideration, the thresholds are $thresh1$ = 362.62 bytes and $thresh2$ =

827.34 bytes. Figure 6.3 shows the distribution of values in the global set. Each point is one distinct value in the set. The red and green lines show the 33^{rd} and 66^{th} percentile boundaries, respectively.

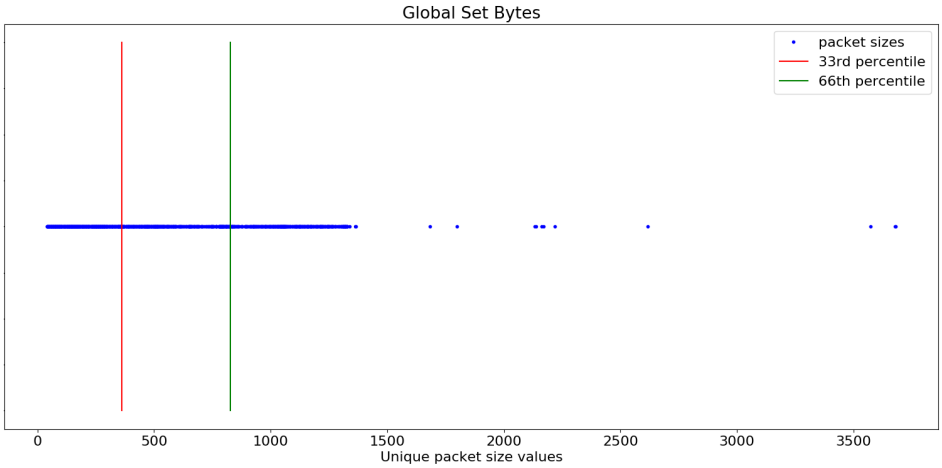


Figure 6.3: A visual representation of the global set used in the Global set-percentile method

Getting back to the Figures 6.2, the global set-percentile method cleans away the noise of the sequences while keeping the original structure intact. In subfigure (a), the data distribution (shown in the histogram) is such that there are local groups of values in the range of 0-400, and then again from 1000-1200. The local percentile method overshoots the categories multiple times. For example, the smaller peaks at the beginning of the sequences are assigned the same categories as the highest peaks in the sequence. Hence, this technique performs the worst. The peak analysis method, however, performs well because of the presence of at least three peaks in the data distribution – it correctly distinguishes between the smaller peaks and the larger peaks. However, it also sometimes overshoots categories, e.g., from 0-20 and 60-80 on the x-axis, it inconsistently assigns multiple data points the Medium category. In subfigure (b), we see a similar trend – the global set-percentile method performing the best by dampening the noise and keeping the overall structure of the sequence intact; and the local percentile method overshooting categories on every chance it gets. However, in the case of the peak analysis method, it performs almost as good as the global set-percentile method because of the presence of its ideal conditions – the presence of three perfectly fragmented peaks.

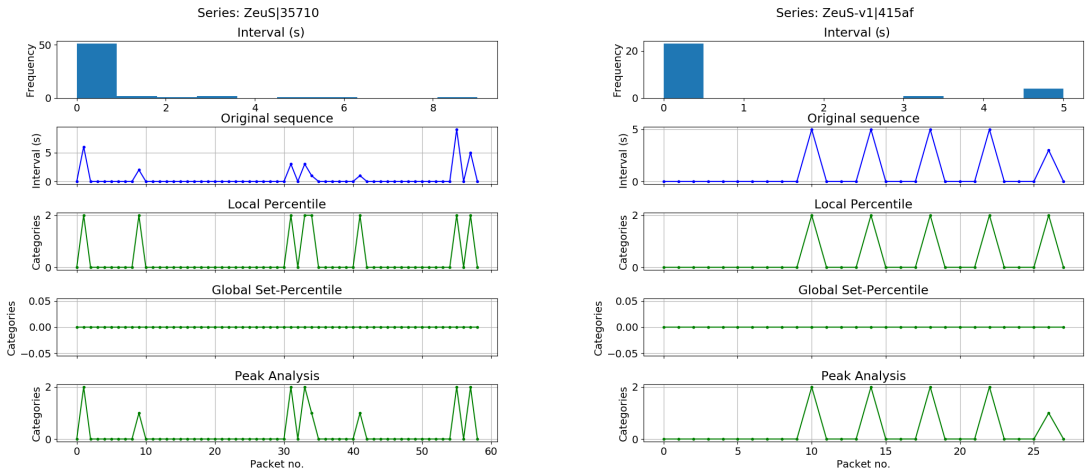
In conclusion, we found the global set-percentile method to be the best discretization technique for packet sizes. It makes the comparison objective among different sequences by assigning similar categories to packet sizes lying in similar ranges. It is also able to keep the sequence's overall shape intact and straightens out any noise that might come its way.

6.2.2. DISCRETIZING INTERVAL BETWEEN PACKETS

The feature of time interval represents the time (in milliseconds) between the current packet and the previous packet. Upon data exploration, it was found that most of the values lie under 5 seconds. However, sometimes delays as high as 16 seconds have been observed. In addition, the dataset is extremely skewed and equally fragmented – the range of possible intervals is very small, and often, the values lie far apart from each other. Hence, this dataset seems ideal for the peak analysis method.

The time interval feature shows how fast or slow a packet is. It can be influenced by a number

of factors, such as network delays, the system sitting idle, or that the malware is programmed to send out packets after regular intervals. We are interested in identifying the latter case – we are looking for frequently occurring values of this feature, which might represent periodicity. Moreover, large values of intervals occur very rarely, which makes it safe to assume that they either occur due to network delays or because there is no active communication happening. For example, an interval of 20 seconds and 60 seconds, both provide the same information that the malware sample's network activity has been idle for a relatively long time. Therefore, such infrequently occurring high values can be thrown in a single category. On the other hand, we should be able to distinguish between the various frequently occurring values. Figures 6.4 show two concrete examples of sequences of the interval between packets. In order to make the comparison easier, the unit of time in the figures is seconds.



(a) Example 1

(b) Example 2

Figure 6.4: Comparing discretization techniques for interval between packets at Pcap-level granularity

In both the figures, two things are very apparent: 1) the histograms show how fragmented the dataset is, and 2) most of the packets are sent with less than a second of delay, which creates sequences where not many peaks exist. As mentioned previously, the dataset is composed of some rarely occurring large interval values. This results in the global thresholds getting skewed by these large values. The global thresholds for this dataset are: $thresh1$ = 12 seconds and $thresh2$ = 26 seconds. This results in almost all the sequences discretized into a single category of Fast packets, which is also apparent in both subfigures (a) and (b). The local percentile method has the same drawback as explained for the packet sizes feature – it tends to overshoot categories quite frequently. In both the subfigures, the local percentile method categorizes all the peaks as slow-paced packets – it does not distinguish between smaller and larger peaks at all. On the other hand, because of the fragmented nature of the dataset, the peak analysis method seems to perform the best. In both of the subfigures, the peak analysis method correctly categorizes smaller peaks as medium-paced packets and larger peaks as slow-paced packets, while maintaining the overall structure of the sequence intact. There is only one scenario when the peak analysis method performs poorly – if there is an insufficient number of frequently occurring values in the sequence, the method will fall back on the local percentile method, which performs the worst in almost all scenarios.

6.2.3. TO DISCRETIZE OR NOT TO DISCRETIZE

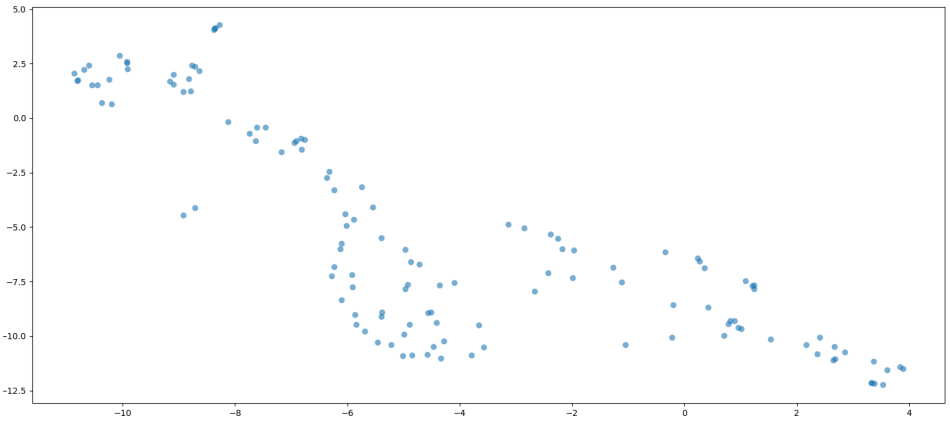
Typically, distance measures for numeric sequences are much slower than those for categorical ones. In the present case, calculating DTW is 16x slower than calculating cosine distance even on a 3.8x shorter sequence. This is shown in Table 6.1. The table shows that it takes 7.07 seconds to discretize 387 sequences (129 sequences per feature), but still the total time it takes to measure the distance between two sequences is 0.19 seconds. However, the fact remains that discretization loses subtle details and precision of the numeric sequences by mapping them in only a few possible categories. Yet, one must ask whether the lost information will affect the clustering performed on the discretized sequences or whether the so-called precision in numeric sequences only adds noise. Hence, in this section, a comparison is made between the data distribution of the Pcaps resulting from numeric versus discretized sequences. The resulting distribution will help visualize the number of potential clusters in the data, and how potentially cohesive and well-separated they will be. For example, if the dataset is too noisy, the clusters will not be well-separated, and the clustering algorithm will have a hard time creating good clusters. Hence, based on the dataset distribution, one can decide whether or not to discretize the sequences.

Figures 6.5 show the dataset distribution of Pcap files represented by numeric (non-discretized) sequences (subfigure a) and discretized sequences (subfigure b). Each blue dot represents a Pcap file in 2-dimensional space, relative to its pairwise distance from every other Pcap in the dataset. Additional explanation of how these plots were generated is given in Section 7.2.1. The scales of the figure do not represent anything concrete. However, the reader should focus on how the data points are scattered on the plot. Remember: the goal of clustering is to extract cohesive clusters since the data points inside them are the most similar to one another. Secondly, the method should allow clusters of different shapes and sizes for added robustness.

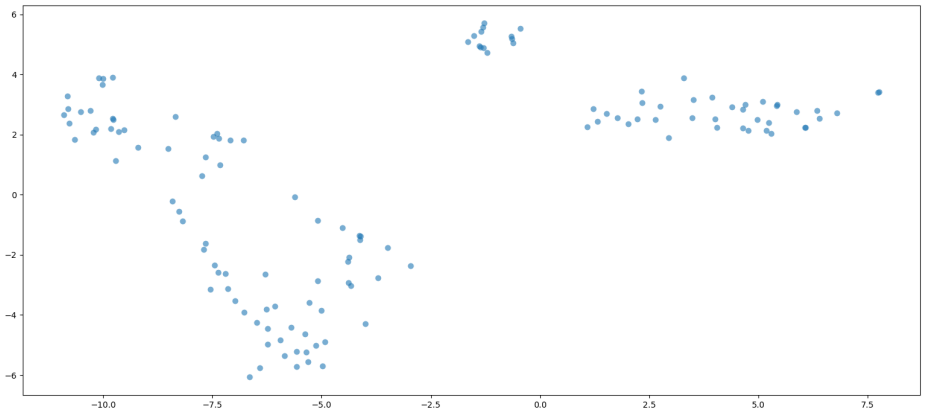
Purely based on the distribution of the points, Figure 6.5(b) seems much better. It shows that there are roughly 3-4 neatly separated clusters. It is also easy to draw the bounds of the clusters. This happens because the extra, and possibly irrelevant values of numeric sequences are compressed into a few categories. Hence, the total possibilities for each data point get reduced significantly, making them go closer to the points they are most likely similar to. Figure 6.5(a) shows the distribution of data points represented by numeric sequences. Because of the high precision of each sequence, the Pcaps end up having a variety of different distances from other Pcaps. This has created 1-2 large clusters whose bounds are very hard to determine, especially compared to Figure 6.5(b). To sum up, discretizing numeric sequences will not only make the clustering 16x faster, but it will also have a positive effect on the final clustering under similar conditions. This analysis has been touched upon further in Chapter 7.

	Length of sequences	**Time to discretize sequences**	**Time to calculate distance matrix**	**No. samples* No. features= No. sequences**	**Time to measure distance per sequence**
Pcap-level, Non-discretized Sequences	First 50 packets	-	1176.87 sec	129*3=387	3.04 sec
Pcap-level, Discretized Sequences	First 190 packets	7.07 sec	70.02 sec	129*3=387	0.19 sec
Connection-level, Non-discretized Sequences	First 20 packets	-	8502.8 sec	733*4=2932	2.9 sec

Table 6.1: Performance evaluation of distance measurement between sequences at Pcap- and Connection- level



(a) Non-discretized sequences



(b) Discretized sequences

Figure 6.5: Comparing data distribution of discretized and non-discretized sequences at Pcap-level

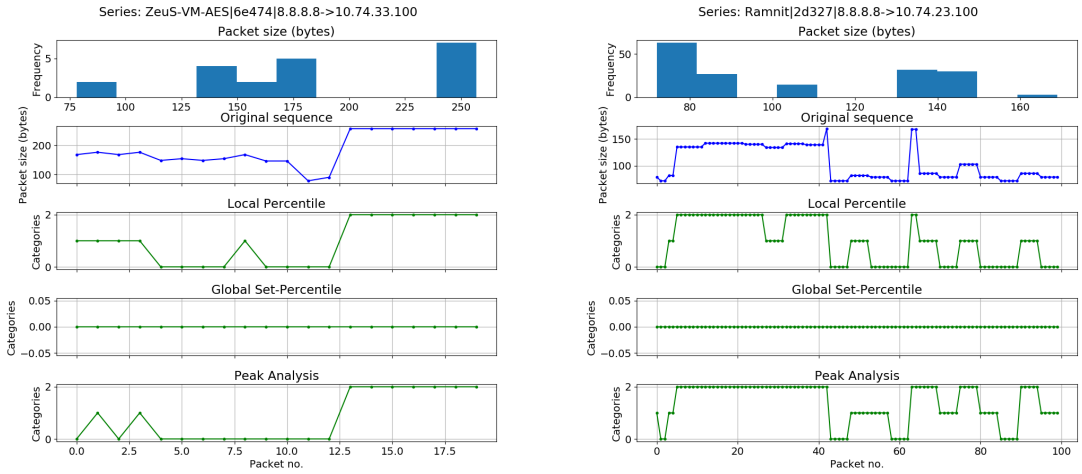
6.3. DISCRETIZING CONNECTION-LEVEL SEQUENCES

At the Connection-level granularity, behavior can be observed with much higher precision than in the case of Pcap-level granularity. At the Pcap-level, the individual conversations are all jumbled up with each other, so additional steps are required to clear away the potential noise in order to see any behavioral patterns. In the case of Connection-level sequences, those behavioral patterns are already visible, and it is much easier to infer about the similarity in behavior by comparing these sequences.

6.3.1. TO DISCRETIZE OR NOT TO DISCRETIZE

As mentioned in Sections 6.2.1 and 6.2.2, Global Set-Percentile method performs the best in discretizing packet sizes, and Peak Analysis method performs the best in discretizing interval between packets at the Pcap-level granularity. For discretizing packet sizes at Connection-level, the threshold values would remain the same as for the Pcap-level, since the connections are essentially made out of Pcap-level sequences. So at $thresh1$ = 362.6 bytes and $thresh2$ = 827.3 bytes, most of the Connection-level sequences fall in only a single category. Two concrete examples of this happening are shown in Figures 6.6. The Global Set-Percentile method maps all the packets to the Small category. In addition, Local Percentile method has already been established to perform the worst

in all scenarios by overshooting categories, and Peak Analysis method to require a certain kind of data distribution, which packet size feature often does not have. For discretizing interval between packets at Connection-level, a similar trend is observed because the distribution of individual sequences has changed significantly. In any case, the average length of Connection-level sequences is approximately 20 packets, which is significantly smaller than that of Pcap-level sequences (i.e., 190 packets). This means that the distance measures for numeric sequences (e.g., DTW) will take relatively less time in computing distances between sequences than in the case of Pcap-level ones. Table 6.1 shows that it takes roughly 2.9 seconds to measure the distance between two numeric sequences. This value seems too high compared to 0.19 seconds, but it is not worth losing precision over. Hence, it was decided not to discretize numeric sequences at the Connection-level granularity.



(a) Example 1

(b) Example 2

Figure 6.6: Comparing discretization techniques for packet sizes at Connection-level granularity

6.4. SUMMARY

In this chapter, we explained three straightforward discretization techniques to convert numeric sequences into categorical ones. The techniques are 1) Local Percentile method, 2) Peak Analysis method, and 3) Global Set-Percentile method. At the Pcap-level, discretizing sequences makes distance measurement between sequences 16x faster, and helps in extracting cleaner cluster boundaries. The Global Set-Percentile method is used to discretize packet sizes, while the Peak analysis method is used to discretize the interval between packets. However, at the Connection-level, the sequences are better represented without discretizing them.

7

CLUSTERING METHODOLOGY

Once the feature set for each granularity is selected, and the distance measure is defined, a distance matrix of size `nxn` is created, where `n` represents the number of Pcaps/connections in the dataset. The distance matrix defines the pairwise distance of every sequence against every other sequence in the dataset. This distance matrix is provided as a parameter to the clustering algorithm, which uses it to create the final clusters.

In this chapter, the selected clustering algorithm and its parameter settings have been explained. In addition, the procedure for result compilation has been described, including static versus dynamic cluster analysis.

A NOTE ON ATTACKER EVASION

Like any cat and mouse game, the results of this research will inevitably give attackers an edge over defenders as they will be able to use evasion techniques more effectively. However, the goal of this research is not to protect this method from attacker evasion. The goal is to show that even with their current technology and tactics, patterns are still available in simple high-level features for defenders to exploit, and that Deep Packet Inspection is not the only solution to identify malicious activity.

7.1. CLUSTERING ALGORITHM SELECTION

Both Single-linkage Hierarchical clustering and HDBScan clustering algorithms (explained in Sections 3.3.1 and 3.3.2) were evaluated. The single-linkage algorithm did a great job of building the hierarchical tree or the dendrogram that showed how the different Pcaps/connections were related to each other. However, the algorithm required the user to provide a parameter defining a cut-off of the tree in order to extract the flat clusters. This parameter is fundamentally unintuitive to provide. In addition, it is highly rare to have just one cut-off point that satisfies the extraction of all the optimal clusters – in many cases, only a few good clusters can be extracted along with some bad ones. One possible solution to find the best cut-off point is to use a statistical cluster quality measure, such as DBIndex, and iteratively try many different values of the cut-off point in order to find the optimal one (which minimizes the DBIndex). This solution was implemented, but the extracted clusters were not optimal, and no interesting patterns were found in the resulting clusters.

Next, the HDBScan clustering algorithm was evaluated. At the time of writing this report, this al-

gorithm performs the best among all the other available clustering algorithms, especially for Python implementations [1]. The algorithm requires only two parameters – minimum cluster size and k-nearest neighbors. Depending on the data distribution, these parameters are not too difficult to determine. It also supports the option of having either 'leaf clusters' or 'root clusters'. A *Root cluster* is defined as a cluster obtained from a sub-root of the hierarchical tree. The *leaf clusters* are formed by the sub-trees of the root cluster, where the size of each leaf cluster is still larger than the minimum cluster size. This helps in extracting smaller and more specialized clusters. Even on the default values of its parameters, this algorithm outperforms Hierarchical clustering algorithm by many folds.

Hence, because of the robustness of HDBScan, this was the algorithm of choice for this project. An important quality of HDBScan is that it does not forcefully assign each point to a cluster. All points whose membership to any cluster cannot be determined, either because of high distance with all other points or because they form bridges between two or more clusters, are considered as *Noisy points*. Hence, a dataset with clear cluster boundaries will have less noise. In the current context, Pcaps/connections that are either too different from all the others or that are somewhat in the middle of two types of behaviors are considered noise. In this project, all noisy points have been dropped from further analysis. An overview of the number of points discarded as noise at each granularity is given in Table 8.1.

7.2. CLUSTER ANALYSIS

A difficult challenge in clustering is to determine the quality of the resulting clusters. The 'goodness' of a cluster is a self-defined and context-dependent notion. Ideally, a metric is defined that can capture the quality of the clusters. Regarding the current problem at hand, the following are the properties that must be captured in the quality index:

- Difference in behavior should fall in different clusters. Each cluster should capture one behavior only. Therefore,well-separated clusters are desired.
- Cluster homogeneity should be high, which means that the items inside a cluster should exhibit very similar behavior.
- Clusters should be cohesive since items farther away from the core could add increasing amounts of noise.

Two cluster quality metrics are specifically meant to measure cluster separation and cohesion. They are Silhouette index and DB Index, which have been explained in Sections 3.4.1 and 3.4.2. However, the values of these indexes do not capture the expected density-based 'goodness', which we define for our clusters. The clustering that was optimal according to our definition resulted in poor values of Silhouette and DBIndex and vice versa. Hence, we needed to define our own quality metric, which we have left as future work. Instead, we have laid the groundwork by performing visual analysis as explained in Section 3.4.3. The visual analysis involves three steps: 1) Dataset Visualization: Visualizing the placement of Pcaps/connections with respect to other Pcaps/connections in the dataset, 2) Cluster Content Visualization: Visualizing the content of each cluster by visualizing each feature individually, and 3) Cluster Label Analysis: Analyzing the distributions of Pcaps/connections with respect to their assigned malware family labels.

[1] http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html

7.2.1. DATASET VISUALIZATION

In order to visualize the distribution of the dataset, its dimensionality needs to be reduced to 2D because of the limitations of current technology. Each Pcap/connection in the dataset is represented as a number of sequences, one for each feature. Once the distance matrix is computed for each level of granularity, each Pcap/connection is represented as a set of distances relative to all the other Pcaps/connections in the dataset. Hence, one Pcap/connection is an n-dimensional array, where `n` is the size of the dataset. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) [2] and t-Distributed Stochastic Neighbor Embedding (t-SNE) [53] can be used to reduce the dimensions of a dataset to visualize it in 2D. t-SNE was chosen because it is specially optimized for the visualization of very large datasets. t-SNE takes an n-dimensional dataset and plots it on a 2D scatter plot. An example of such a scatter plot is shown in Figure 7.1. Each Pcap/connection is shown as one data point. The plot's x- and y-axes represent the relative distances in reduced dimensions. For analysis purposes, they do not convey any important information. However, the plot itself shows the positioning of points relative to other points in the dataset, which is useful in determining the ideal number of clusters.

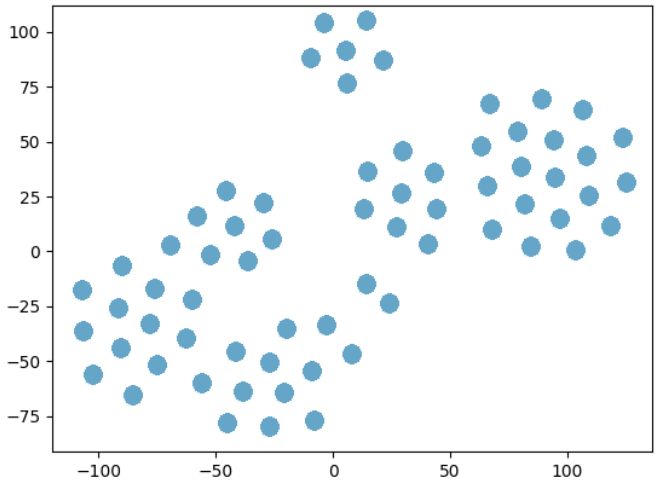


Figure 7.1: t-SNE example – dataset visualization pre-clustering

t-SNE can also be applied once the clustering algorithm has extracted flat clusters. Figure 7.2 shows the extracted clusters from the dataset shown in Figure 7.1. The parameter setting was (minimum cluster size = 6 and k-nearest neighbors was left unspecified). Each color represents a distinct cluster, and the annotation on the data point shows the cluster number the point belongs to. The figure shows that the dataset was divided into six clusters. Three gray points with the label '-1' are considered noise since the algorithm could not determine their membership to any cluster. According to the visual analysis, these clusters seem to be very cohesive, and none of the clusters seem to overlap other clusters.

A number of different parameters of the clustering algorithm were evaluated until the clusters closest to the expected ones were formed. The configuration dataset on which the evaluation was performed was roughly 5% of the original dataset.

[2] https://en.wikipedia.org/wiki/Principal_component_analysis

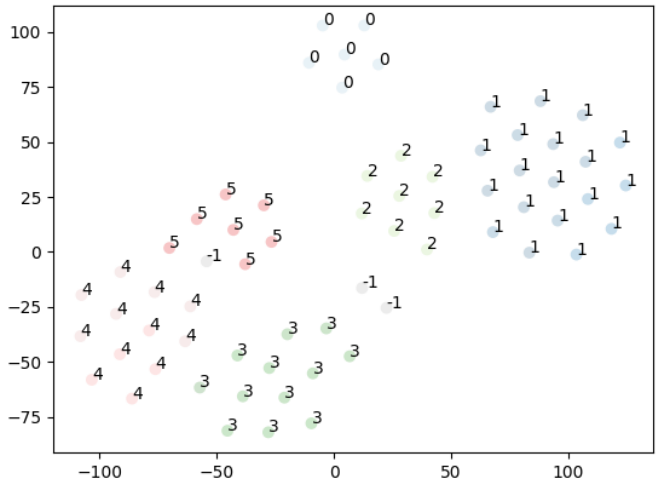


Figure 7.2: t-SNE example – dataset visualization post-clustering

7.2.2. CLUSTER CONTENT VISUALIZATION

Each of the Pcap/connection in a cluster has at least four sequences associated to it, one corresponding to each feature. We utilize heatmaps to visualize sequences in a cluster, one feature at a time. Heatmaps represent each value of a feature as a color. An example of such visualization is shown in Figure 7.3. The figure shows one of the clusters extracted by HDBScan. The title shows the cluster number (which is meant to enumerate the kind of behaviors extracted), followed by the name of the feature whose values are visualized in the graph. The x-axis represents the packet number. Each row corresponds to a sequence of one Pcap or connection depending on the granularity and shows the feature value of the first 20 packets in a Pcap/connection. The row labels are encoded to reduce the space they take. There are at most three parts of the row labels, showing: 1) the assigned family name of the Pcap/connection encoded as the first three letters of the family name (e.g., Zeus-Panda becomes ZPA), 2) the first five characters of the SHA1 hash of the Pcap encoded as a unique 3-digit number (e.g., 1e1e4 becomes 128), and 3) the source and destination IP address encoded as 3-digit numbers for each unique host (e.g., 211.105.106.1 becomes 046) (only applicable for Connection-level granularity). The mapping of the original values to the encoded ones are given in Appendix A.

This figure shows that there are eight connections, exhibiting two major kinds of behaviors, encapsulated in this cluster. The first three connections send packets of similar sizes with similar intervals. A different kind of behavior is common among the rest of the five points. The reader should observe the slight lag in the 4^{th} connection, which was elegantly handled by DTW. Because the overall distance measure is made up of distances from at most four features, the heatmaps for the other features will also be visualized in the same way. This analysis provides a deeper understanding of what the content of each Pcap/connection looks like with respect to that of other Pcap/connections in the same cluster.

A quick False Positive analysis was also performed on the basis of cluster content visualization. A *False positive* is defined as a Pcap/connection placed in cluster `x`, but whose values are almost identical to Pcaps/connections in cluster `y`. In addition, a Pcap/connection placed in a cluster all of whose features are different from other members of that cluster is also labeled as a False positive.

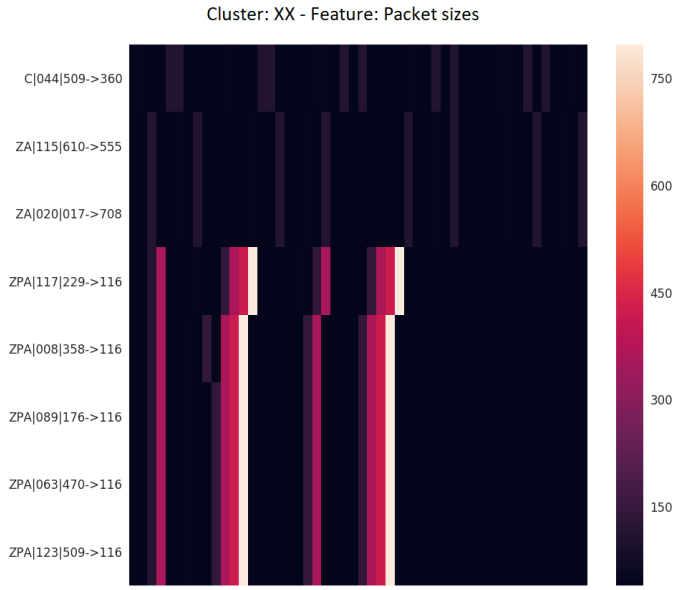


Figure 7.3: Heatmap example – Visualizing cluster content

7.2.3. CLUSTER LABEL ANALYSIS

Each cluster represents a unique behavior exhibited by a group of Pcaps/Connections. Each Pcap was assigned a family label based on its static analysis, which may or may not correspond to its network-level behavior. An important aspect of this research is to find out whether malware samples' family labels also map to their network behavior. Hence, each malware family is analyzed separately by finding the percentage of samples split across various clusters. We identify the family split based on two situations:

1) **Uniformity/Diversity of behavior**: *For a malware family, uniform behavior means only one kind of behavior, while Diverse behavior means multiple kinds of behaviors.*
Uniform malware samples all lie in one cluster since only one aspect of their behavior is visible. On the other hand, malware shows diverse behavior when Pcaps or connections of the same family are divided across several clusters, showing the different (potential) attacking capabilities possessed by the malware family.

2) **Common/Rare behavior**: *For multiple malware families, the behavior common among them is called Common behavior, while the behavior only specific to one malware family that is never seen in another family is called Rare behavior.*
The common behavior can either be benign-running operating system services or common attacking behaviors showing a relationship between different malware families. Rare behaviors, on the other hand, are represented by clusters that only contain samples from one malware family, and no other family is observed behaving in that way. These behaviors distinguish one malware family from the others. Hence, rare clusters can be used to extract behavioral signatures of samples behaving in unique ways.

7.3. PCAP-LEVEL CLUSTERING

At the Pcap-level, there are three features: packet sizes, the interval between packets, and the protocol used (details in Section 5.2). The sequences of packet sizes and the interval between packets have been discretized because the dataset becomes cleaner compared to its non-discretized alternative and high-quality clusters can be extracted (explained in Section 6.2.3). Hence, three categorical features represent each Pcap. The distance between sequences is calculated using trigram analysis (explained in Section 3.2.5). The formula used to consolidate all the distances in a single number is given in equation 5.1. The average length of a sequence at the Pcap-level is 190 packets, and there are a total of 129 Pcaps out of 1196 fulfilling the criteria to be considered. Hence the distance matrix is of dimensions 129x129.

The parameters provided to HDBScan were:

- Minimum cluster size = 7
- K-nearest neighbors = 1

Since the data distribution is sparse, a lower k-neighbor value will reduce the number of Pcaps discarded as noise. In addition, the smallest well-separated group of points contained seven Pcaps. The clustering algorithm produces five clusters. There are, on average, 21 Pcaps in each cluster. 25 Pcaps are discarded as noise since their membership to any cluster could not be determined. The final five clusters are made up of 104 Pcap files. The Pcap distribution after clustering is shown in Figure 7.4. The gray points without any label are noise and have been discarded from further analysis. Clusters 2 and 3 almost overlap each other, but the rest of the clusters are well-separated. Further analysis of these clusters is presented in Chapter 8.

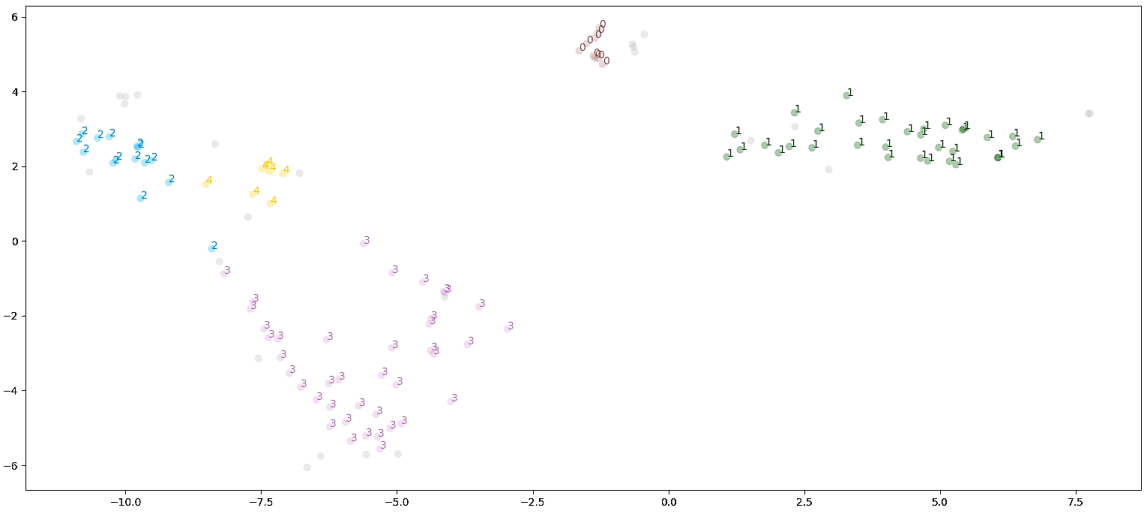


Figure 7.4: Data distribution of clusters obtained at Pcap-level granularity

As a cursory comparison of discretizing sequences (Figure 7.4) versus non-discretized sequences, under similar parameters, the following results are observed:

64 Pcaps are discarded as noise when clustering with numeric sequences, as opposed to 25 points in the case of discretized sequences. Even though a detailed cluster analysis shows that the

same data points populate clusters in both cases, there are a few cases of points shifting clusters. In addition, the clusters made from discretized sequences are well-separated, which results is easier bounds definition for clusters. Lastly, distance matrix calculation on discretized sequences is 16x faster on 3.8x longer sequences than non-discretized, numeric sequences. Hence, discretizing sequences at the Pcap-level is the optimal choice.

7.4. CONNECTION-LEVEL CLUSTERING

At the Connection-level, there are four features: packet sizes, the interval between packets, source and destination ports (details in Section 5.3). There are two numeric sequences: packet sizes, and the interval between packets. The distance between them is calculated using Dynamic Time Warping (explained in Section 3.2.2). In addition, there are two categorical sequences: source and destination port numbers. The distance between them is calculated using trigram analysis (Section 3.2.5). The formula used to consolidate all the distances in a single number is given in equation 5.2. The average length of a sequence at the Connection-level is 20 packets. There are a total of 8997 connections coming out of 1196 Pcap files. Out of 8997 connections, only 733 connections fulfill the criteria to be considered. These 733 connections form a total of 216 Pcap-files. Hence, the distance matrix is of dimensions: 733x733.

The parameters provided to HDBScan were:

- Minimum cluster size = 7
- K-nearest neighbors = 7

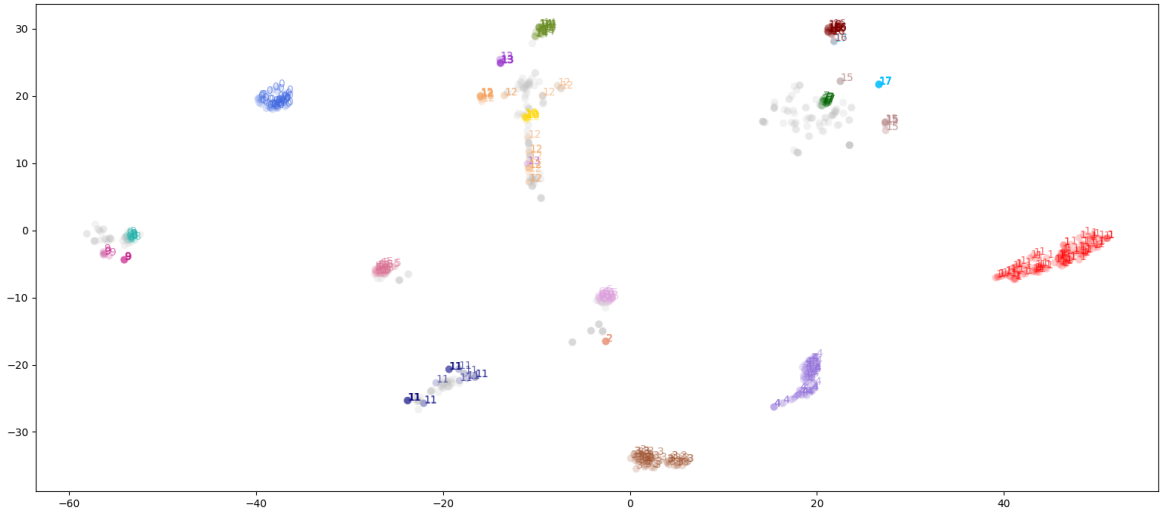


Figure 7.5: Data distribution of clusters obtained at Connection-level granularity

The data distribution for Connection-level is dense, so the k-neighbor value has been increased to match the minimum cluster size. The smallest well-separated cluster size is the same as that of the Pcap-level. The clustering algorithm produces 18 clusters. There are, on average, 25 connections in each cluster. 284 connections are discarded as noise. The final 18 clusters are made up of 449 connections coming from 172 Pcap-files. The data distribution after clustering is shown in Figure 7.5. The gray points without any label are noise and have not been considered in further analysis. It

can be seen that most of the clusters are well-separated. Clusters (12,13,14) and clusters (15,16,17) are overlapping among each other. Further analysis of these clusters is presented in Chapter 8.

7.5. SUMMARY

In this chapter, we explained the motivation behind using the HDBScan clustering algorithm and describe the parameters used at the Pcap- and Connection-level granularity. We also describe three cluster analysis techniques used to understand the resulting clusters in more detail – 1) Dataset visualization helps estimate the number and quality of resulting clusters; 2) Cluster content visualization provides an in-depth view of what behaviors are grouped inside a cluster and also helps identify false positives; and 3) Cluster label analysis provides a comparison between clusters generated from static (family labels) and dynamic (network traces) analysis.

8

Results and Discussion

In this chapter, we discuss the insights that we have obtained from clustering malware samples' network activity. We divide this section into five parts. First, we present the findings of Pcap-level and Connection-level analysis. Next, we summarize the most interesting attacking capabilities the proposed method is able to detect. Then, a comparison of clusters obtained from static versus dynamic analysis is presented, followed by a brief comparison with the baseline version.

8.1. Pcap-level Cluster analysis

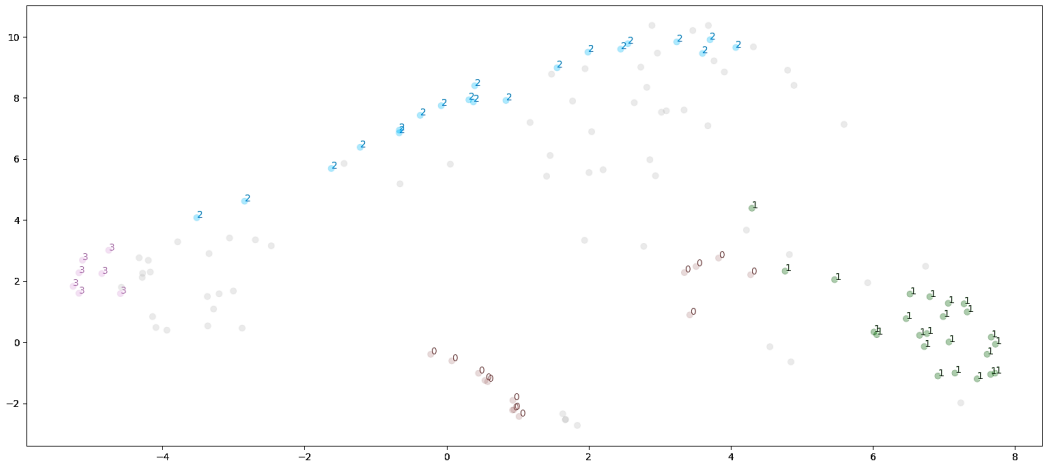
At the Pcap level, 104 Pcap files produce 5 clusters. A detailed analysis of the cluster content and detected behaviors is explained below.

8.1.1. Discretized versus Non-discretized Sequence Clustering

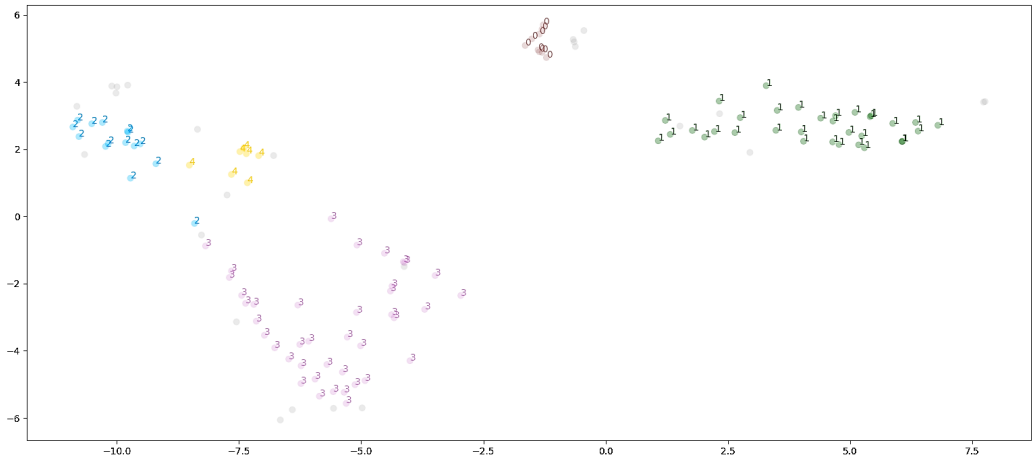
64 Pcaps are discarded as noise in non-discretized sequences, as opposed to only 25 for discretized sequences. This happens because there are many points that form bridges between core clusters, as shown in the t-SNE plot in Figure 8.1(a).

Clusters 0 and 1: The Pcaps in cluster 0 of discretized sequences (Figure 8.1(b)) also lie in cluster 0 of non-discretized sequences (Figure 8.1(a)). The same case is observed for cluster 1. However, 5 data points in cluster 0 of non-discretized sequences lie very close to cluster 1, which indeed are labeled as cluster 1 in discretized sequences. We consider these 5 points as false positives for non-discretized sequences. Therefore, it can be concluded that the clusters 0 and 1 of discretized sequences are much more cohesive and well-separated than those of non-discretized sequences.

Clusters 2, 3, and 4: The Pcaps labelled as noise in discretized sequences are also marked as noise in non-discretized sequences. However, 39 additional Pcaps in non-discretized sequences are also labeled as noise. These additional noise points come out of cluster 2 and 3 (the long island of points in Figure 8.1(a)). In addition, the cluster 3 of discretized sequences is represented by a combination of cluster 2 and noise points in non-discretized sequences. Moreover, the cluster 4 of discretized sequences is entirely discarded as noise in non-discretized sequences.



(a) Non-discretized sequence clusters



(b) Discretized sequence clusters

Figure 8.1: Comparing non-discretized and discretized sequence clusters

To sum up, non-discretized sequences discard a lot of Pcaps as noise, compared to discretized sequences. Hence, in the rest of the chapter, we only explore clusters obtained from discretized sequences.

8.1.2. CLUSTER ANALYSIS

In general, a lot of periodicity can be seen for each cluster in terms of packet sizes and the interval between packets. However, the protocol feature does not show any interesting patterns. An example of a cluster's protocol sequence is shown in Figure 8.2. The protocol feature only switches between TCP, UDP, ICMP, and IGMP protocols, which does not show any interesting patterns.

In general, each cluster shows a distinct type of behavior. In the following subsections, we analyze the behavior captured by two such clusters:

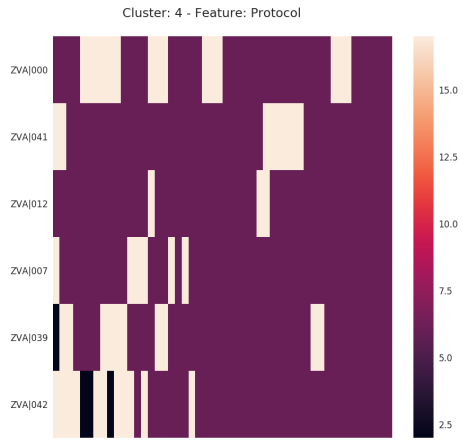
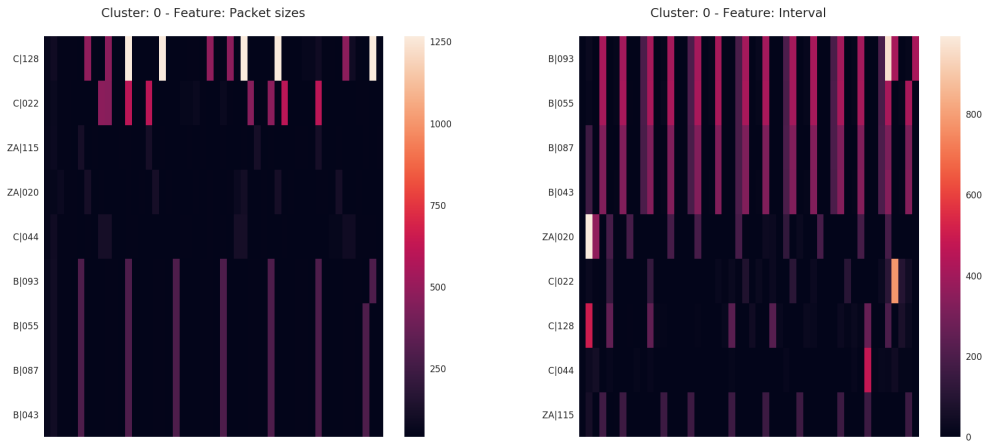


Figure 8.2: An example cluster of sequence of protocols at Pcap-level

A CASE OF BLACKMOON

Cluster 0 contains four unique Pcap files labeled as Blackmoon. They all have extremely similar behavior. The four rows of periodic lines show that similarity in Figures 8.3(a) and 8.3(b). Upon traffic analysis of two randomly selected samples, it was found that they both contact the same C&C at 103.7.30.86, which has been reported as malicious on many threat intelligence databases. The traffic shows a series of TCP 3-way handshake, followed by an HTTP-GET request of exactly 305 bytes, followed by a request to close the TCP connection. Hence, it makes a pattern of six TCP packets followed by one big HTTP packet. In addition, the port number 80 of C&C server is contacted each time. However, the source ports are gradually incremented by one (starting from 1044 until 1059) at each new TCP session. A snapshot of the Pcap file is shown in Figure 8.4.



(a) Sequence of packet sizes

(b) Sequence of intervals

Figure 8.3: Feature set visualization of Cluster 0 at Pcap-level

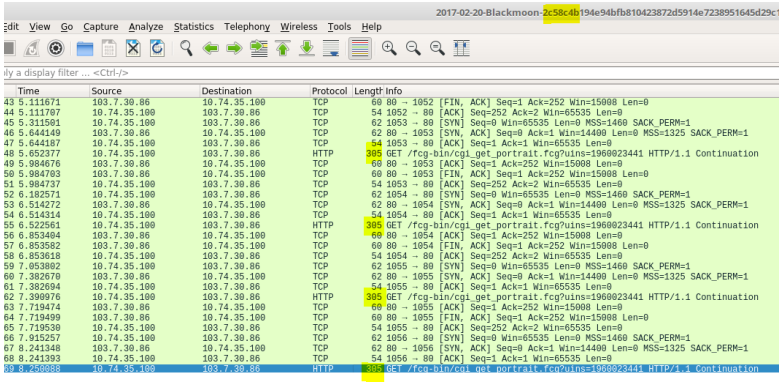
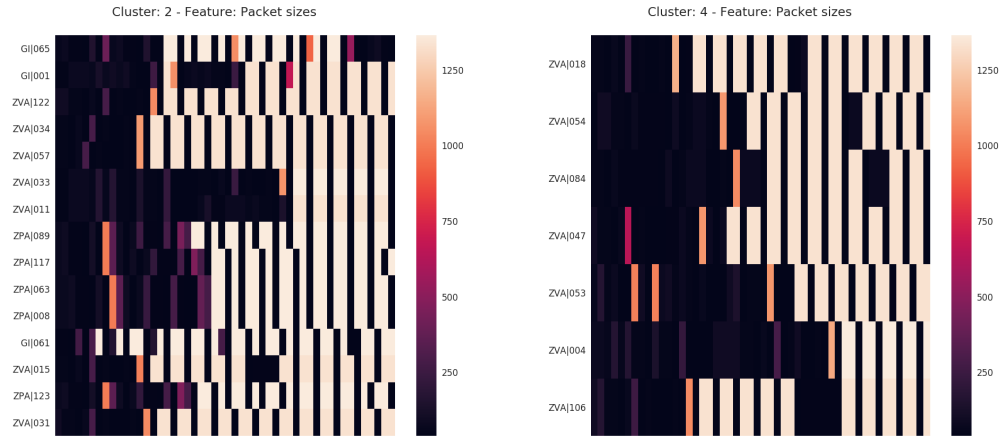


Figure 8.4: Blackmoon's traffic content exploration in Wireshark

A CASE OF GOZI AND ZEUS

Clusters 2 and 4 lie very close to each other on the scatter plot (Figure 8.1(b)). Their Pcaps' sequences of packet sizes are also very similar, as shown in Figures 8.5. We analyze two samples from cluster 2 – one labeled as Gozi-ISFB and the other as Zeus-VM-AES. They both lie in subfigure (a) and have alternating behavior between huge and small packets. Upon traffic analysis, we found that Gozi-ISFB visits a website and downloads tor/t32.dll file. This .dll file is a TOR client that is used along with a Rig Exploit Kit. In addition, the IP address it visits to download the .dll file is also reported as malicious in several threat intelligence databases. Upon traffic analysis of the Zeus-VM-AES sample, we found that it sends a POST request to the domain letit2.bit/, which is the landing page for Chthonic banking malware. The periodic behavior in the figure is a series of 2 large TCP packets delivering the exploit kit followed by one acknowledgment packet as a response.



(a) Sequence of packet sizes

(b) Sequence of packet sizes

Figure 8.5: Feature set visualization of Clusters 2 and 4 at Pcap-level

8.1.3. SUMMARY

In light of the observations shown above, the clusters do in fact group together similar behaviors, and they are malicious in most of the observed cases. However, because the dataset was collected in a controlled environment, we can be sure that the behaviors grouped are malicious. However, if

several benign requests were made to a server to download a legitimate software, we expect to see similar patterns. Moreover, at the Pcap-level granularity, multiple IP addresses are combined in a single sequence, so it is difficult to distinguish benign connections from potentially malicious ones.

It has also been observed that malware performs periodic actions, e.g., sending a heartbeat message to the C&C server to let it know that it is still part of the botnet. However, our observations show that the periodicity is not always regarding 'time', but is more frequently regarding packet sizes. One explanation behind this observation is that the time interval can be affected by network delays, but the packet sizes are immune to it. Hence, even if the periodicity of the time interval is lost due to delays, the periodicity of packet sizes can be exploited to group similar behaviors together.

8.2. CONNECTION-LEVEL CLUSTER ANALYSIS

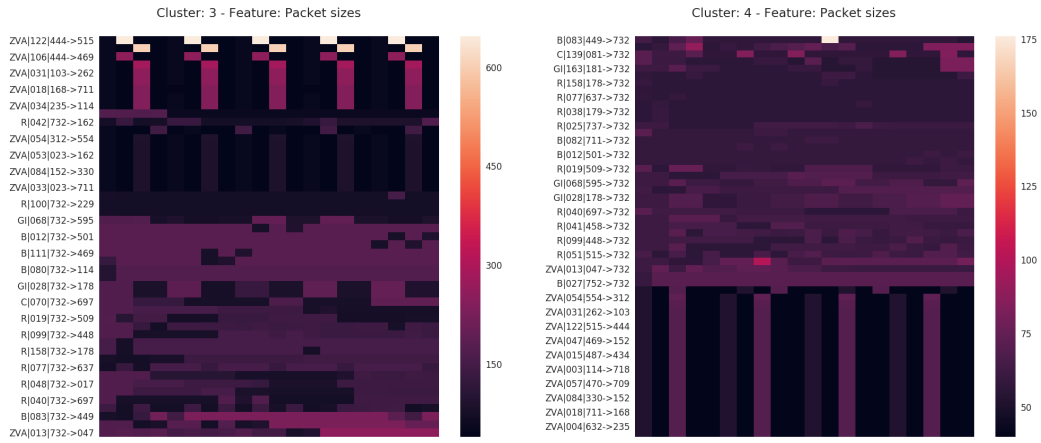
At the Connection-level, 449 connections produce 18 clusters. A detailed analysis of the clusters and the behaviors they exhibit is explained below.

8.2.1. CLUSTER ANALYSIS

Clusters 0, 1, and 2: Cluster 0 represents all connections broadcasting to 239.255.255.250, which is used by the SSDP protocol to find Plug and Play devices. Cluster 2 represents all connections broadcasting to 224.0.0.252, which is used by Link-Local Multicast Name Resolution (LLMNR) protocol to find local network computers. Cluster 1 represents all requests to the broadcast IP address of the localhost, potentially in search of other devices connected to the local network. These three clusters seem to exhibit device searching behavior. However, it cannot be concluded with certainty whether they are used for malicious purposes.

Clusters 3 and 4: Cluster 4 contains outgoing traffic to multiple IP addresses. The responses of those hosts are received as incoming connections in cluster 3. These clusters show malware samples primarily labeled as Zeus-VM-AES, Ramnit, and Blackmoon. Cluster 4 shows two distinct kinds of requests being sent, while the responses can be grouped into five types (from visual analysis of heatmaps in Figure 8.6). All the IPs contacted by the samples of this cluster operate on DNS (port number 53), while the contacted port numbers of localhost are over 60000. Ramnit contacts 8.8.8.8, which is Google's public DNS. However, there are reports that Ramnit uses DNS to query for DGA domains [1]. Three samples from Zeus-VM-AES contact 144.76.133.38, which has been reported to drop Cerber ransomware. Three other samples of the same family contact 62.113.203.55 and 62.113.203.99, which are both associated with ZLoader ransomware [54]. Another two unique samples of the same family contact 89.18.27.34, which were detected two days apart. This IP address is associated with a Window's malware, called CoinMiner. In addition, we also found five Pcap files, each of which has two associated IP addresses, visible in 2 connections each. All 10 of those IP addresses have been reported as malicious. This can indicate the presence of multiple C&C servers for each of these malware samples.

[1] https://malwarebreakdown.com/2017/07/24/the-seamless-campaign-drops-ramnit-follow-up-malware-azorult-stealer-smoke-loader-etc/



(a) Incoming connections (b) Outgoing connections

Figure 8.6: Feature set visualization of Clusters 3 and 4 at Connection-level

Cluster 7: Cluster 7 contains outgoing connections, all of which belong to Zeus-VM-AES (Figure 8.7). Five connections from this cluster contact 47.91.124.165, which has been reported as the C&C server of Despicable.ME malware dropped via Rig Exploit Kit. These five samples were all detected on the same day and were executed on different Virtual machines. Hence, we can be sure that the IP address is the C&C belonging to Zeus malware. We also found two samples contacting 185.195.24.6 and then one sample was detected 15 days later, which contacts 185.195.24.139. Hence, it seems that the malware changed its C&C server to a host within the same subnet in the 15-day period. All these samples are HTTP-based malware since they operate on port number 80. The port numbers of the localhost are all very high (in the range from 52500 to 62500).

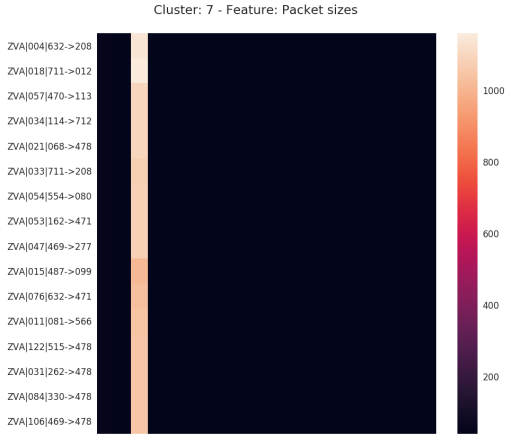


Figure 8.7: Outgoing connections from Cluster 7 at Connection-level

Cluster 10: Cluster 10 contains incoming connections belonging to Zeus-Panda and Blackmoon malware families, as shown in Figure 8.8. Two samples from Zeus-Panda and one sample from Blackmoon contact 13.107.4.50, which is an official Microsoft IP address, but has been reported as a C&C server. In addition, two samples from Zeus-Panda contact 88.221.14.11, while one sample from Blackmoon contacts 88.221.14.16, which belongs to the same subset and hence, may be related to each other. The former IP address belongs to an Internet explorer banking malware targeted towards

the bank of Queensland in Australia, and the former one has also been reported as malicious.

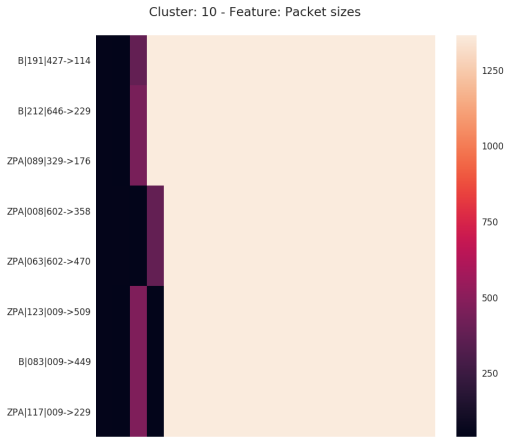
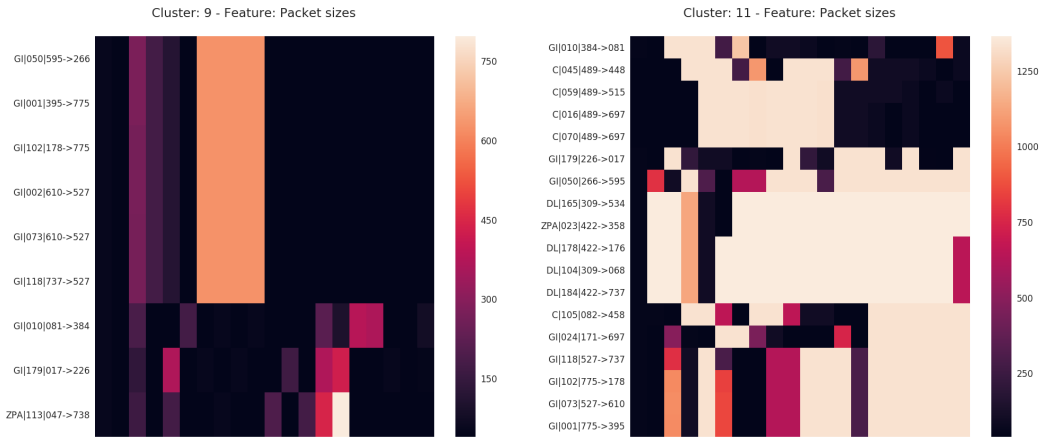


Figure 8.8: Incoming connections from 10 at Connection-level

Clusters 9 and 11: Cluster 9 contains connections that send requests to several IP addresses and Cluster 11 contains the resulting incoming connections, as shown in Figures 8.9. Cluster 9 is heavily composed of Gozi-ISFB's samples, while cluster 11 also contains incoming connections from other malware families, e.g., Dridex-Loader and Citadel. In cluster 9, there are three connections that contact 194.109.206.212, which has been reported as the C&C of ransomAQE malware originated from the Netherlands; and two connections contact 86.59.21.38, which has been reported as Troldesh ransomware. In cluster 11, there are three samples labeled as Zeus-Panda and two samples labeled as Dridex-Loader, all of which contact the same IP address – 40.113.17.180. However, we were unable to find any reports regarding the maliciousness of this IP address. In any case, this IP address uncovers some collaboration between the two malware families. All the samples in these clusters represent HTTPs-based connections as they communicate over port number 443 (SSL).

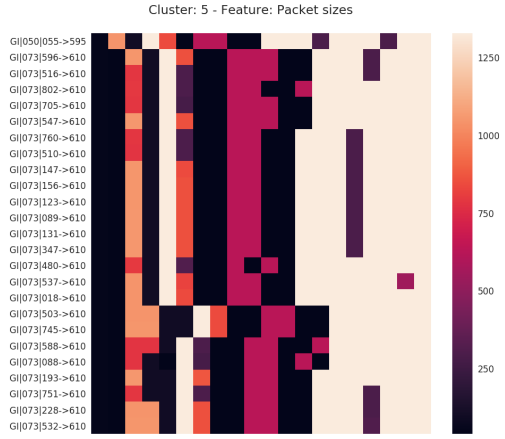


(a) Outgoing connections (b) Incoming connections

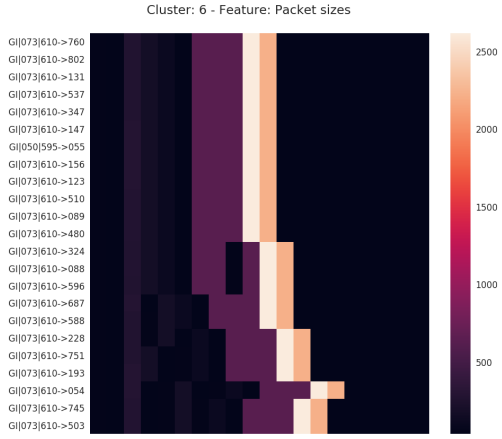
Figure 8.9: Feature set visualization of Clusters 9 and 11 at Connection-level

Clusters 5, 6, and 8: The connections inside clusters 5, 6, and 8 show an interesting behavior – they all contain connections from one Pcap file, which is labeled as Gozi-ISFB. Interestingly, this

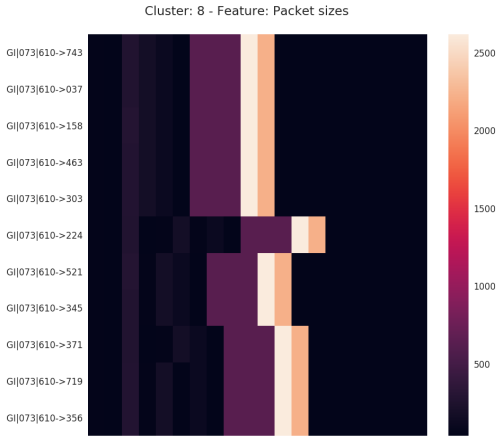
Pcap file has a total of 111 connections, 62 of which are split between clusters 5, 6 and 8 (Figures 8.10). The rest of the connections were discarded as noise. Each of the 62 connections contacts unique IP addresses. Cluster 6 and 8 (subfigures (b) and (c)) contain outgoing traffic, and cluster 5 (subfigure (a)) contains incoming traffic from these IP addresses. The reason why clusters 6 and 8 were split is that the port number of the contacted IP addresses is different – port 443 (SSL) is contacted in case of cluster 8, and port 9000 (EverQuest World Server) is contacted in case of cluster 6.



(a) Incoming connections



(b) Outgoing connections



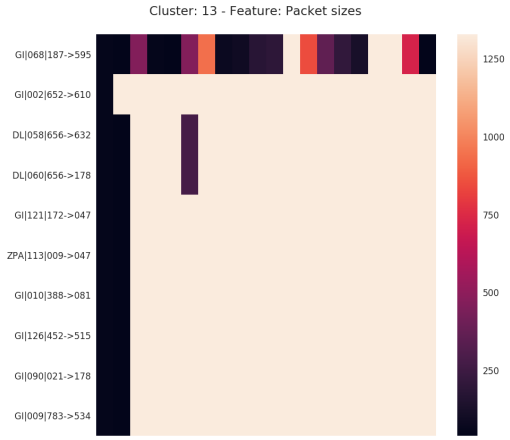
(c) Outgoing connections

Figure 8.10: Feature set visualization of Clusters 5, 6, 8 at Connection-level

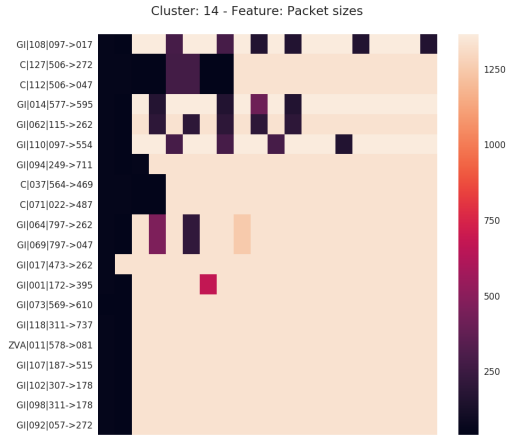
Clusters 12, 13, 14, 15, 16, and 17: Clusters 12, 13, 14 and 15, 16, 17 are not well-separated. Clusters 12, 13 and 14 contain of incoming connections (Figure 8.11), while the latter ones contain their resulting outgoing connections (Figure 8.12). Cluster 12 contains four Gozi-ISFB samples contacting 171.25.193.9, which is associated to RansomAQE malware originating from the Netherlands; four Citadel samples contacting 178.255.83.1, which has reports of being associated to WannaCry ransomware; two Zeus-Panda samples contacting 188.120.243.35, which is a Russian IP contacting EvilBets.com. In addition, four Dridex-Loader samples contact 80.83.118.233, and three Citadel samples contact 185.72.178.171, about which we could not find any malicious reports. In cluster 14, two Citadel samples contact 46.235.9.33, which was found in the blacklist of a company's IDS;

and two Gozi-ISFB samples contact 69.156.240.29, which was found on a ransomware tracker website. There were no conclusive reports about the maliciousness of the IP addresses contacted by connections in cluster 13. On the flip slide, cluster 16 is dominated by Gozi-ISFB samples, of which most are malicious. There were no conclusive reports regarding the IP addresses contacted by the Dridex-Loader samples in Cluster 15 and 17.

(a) Incoming connections



(b) Incoming connections



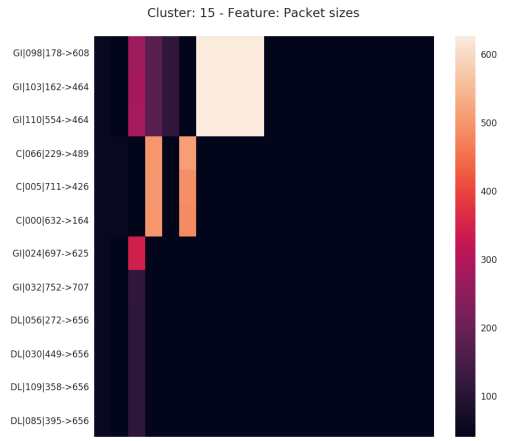
(c) Incoming connections

Figure 8.11: Feature set visualization of Clusters 12, 13, 14 at Connection-level

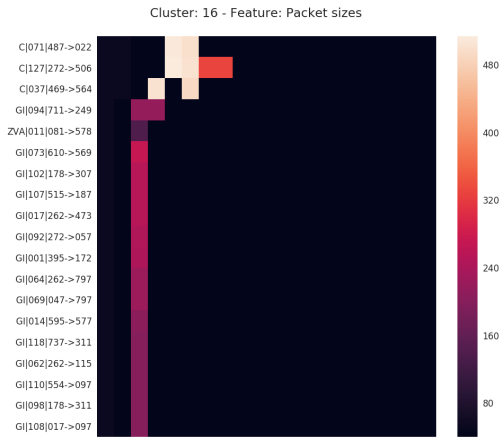
Upon detailed cluster analysis, the clustered space can be divided into four regions, as shown in Figure 8.13. The clusters on the left are all incoming connections, while the clusters on the right are outgoing connections. The top section is for HTTP-based connections (communication over port 80), while the lower section has a mix of SSL-, DNS-, and other protocol-based connections.

8.2.2. FALSE POSITIVE ANALYSIS

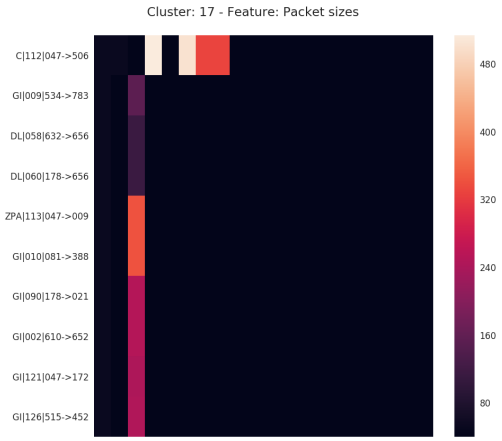
As mentioned in Section 5.4, in order to find the distance between two connections, four sequences of one connection are compared against four sequences of the other connection. This means that for every data point in a cluster, there are four associated heatmaps, one for each feature. As introduced in Section 7.2.2, a connection is flagged as a False Positive if none of the four features match



(a) Outgoing connections



(b) Outgoing connections



(c) Outgoing connections

Figure 8.12: Feature set visualization of Clusters 15, 16, 17 at Connection-level

the other members of its cluster. On the other hand, if multiple connections contact the same IP address and have the same values of the four features, but still end up getting split into different clusters, then they are also flagged as false positives.

Out of 449 connections, only 2.6% are flagged as False positives. They originate from 33% of clusters – cluster numbers 13, 14, 15, 16 and 17. It was expected since clusters 13, 14 and 15 overlap each other, and clusters 15, 16 and 17 overlap each other, as shown in the scatter plot 7.5.

- Two distinct connections of Gozi-ISFB get split into clusters 13 and 14. They both contact 185.64.219.6 and have negligible difference in their features.
- Similarly, two distinct connections from Citadel get split into clusters 16 and 17. They both contact 46.235.9.33 and have negligible differences in their features.
- Six distinct connections of Dridex-Loader contact 80.83.118.233. Four connections end up in cluster 15 and two in cluster 17. All six have minor differences in their interval between packets.

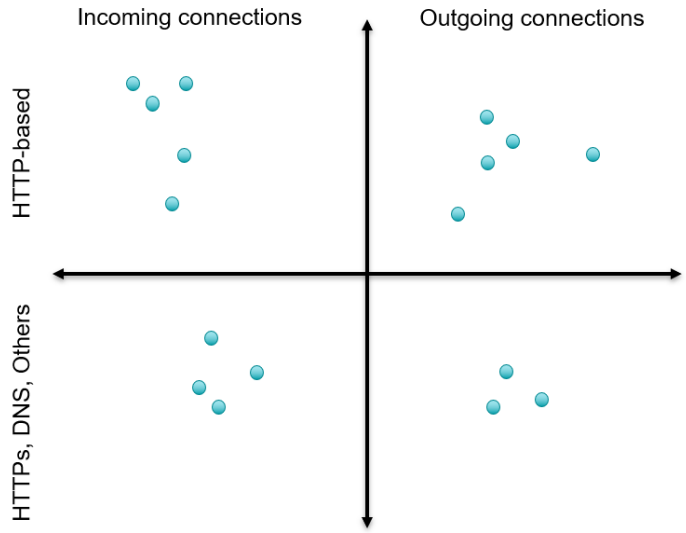


Figure 8.13: Division of Connection-level cluster space

- Finally, two distinct connections from Gozi-ISFB get split into cluster 16 and 17. They both contact 185.64.219.6 and the value of their source port is off by just one value (49166 versus 49165).

8.2.3. IP ADDRESS ANALYSIS

We collected 5% (40) unique IP addresses, at random, from all the 18 clusters, which were contacted by at least 2 connections. These IP addresses were manually analyzed if they had any existing reports in threat intelligence databases, such as VirusTotal, ThreatCrowd, ThreatMiner, and Cymon.io. Out of the 40 IP addresses, 80% of them had reports of being associated with various kinds of malware, e.g., ransomware, banking Trojans, and cryptocurrency miners. For the other 20%, no conclusive remarks could be made since the malware samples were executed in a controlled environment. We suspect that these IP addresses are either malicious but were never reported, or are contacted as a result of evasion techniques used by the malware sample.

8.2.4. SUMMARY

In light of the observations shown above, it is evident that the proposed technique groups together different attacking capabilities of malware samples. Connections that behave similarly are grouped, with a False positive rate of 2.6%, irrespective of their family labels. The results show a different way of clustering malware samples. Upon reflection, we believe that organizations care more about the kind of attack that is being performed rather than which malware family is attacking them. Hence, the proposed technique clusters similar attacking behaviors rather than extracting signatures for specific malware families, which has been the aim of most standard anti-virus software [16].

It is also apparent that at the Connection-level, the behavioral patterns are clearer and a lot more information is available to infer about the kind of behaviors captured in the clusters. Although clustering at the Connection-level requires more resources compared to Pcap-level, the results are very promising. Optimizing Connection-level clustering for performance is left as future work.

8.3. DETECTED ATTACKING CAPABILITIES

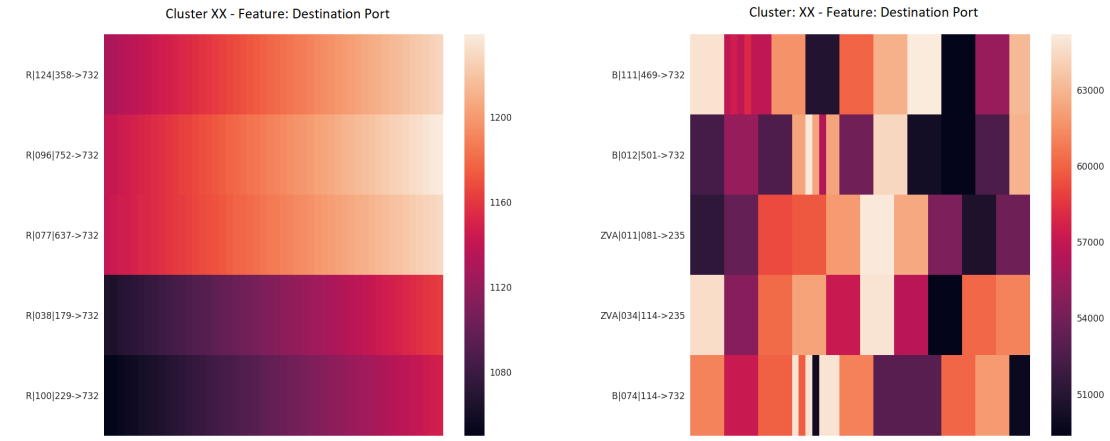
This section lists the interesting behaviors that were captured in the clusters obtained at the Connection-level granularity:

8.3.1. INCOMING/OUTGOING CONNECTIONS

The clustering algorithm is successfully able to distinguish between incoming and outgoing connections even though the IP address was not used in the feature set. One might assume that the port numbers are responsible for this reason – the direction where the port number stays fixed shows the outgoing traffic. However, we observed multiple instances where both the source and destination ports were not fixed to a single value.

8.3.2. PORT SCANS

Two types of port scans have been detected as a result of using sequences of port numbers as features. A couple of examples are shown in Figures 8.14. Subfigure (a) shows a systematic port scan where ports from 1080 till 1200 are contacted incrementally. This causes the series to be shown as a gradient. Subfigure (b) shows a randomized port scan, which scans ports from 51000 to 63000 randomly, which causes the series to have a checkered effect.



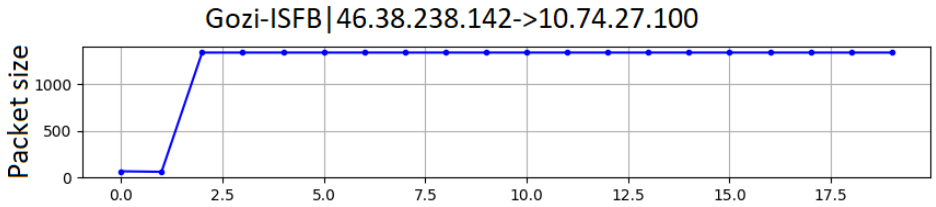
(a) Systematic port scan

(b) Randomized port scan

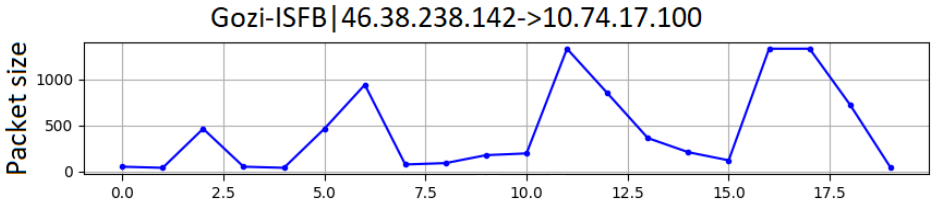
Figure 8.14: Example of port scans detected at Connection-level

8.3.3. SPLIT-PERSONALITY C&C SERVERS

Several instances were found where multiple connections contacted the same IP address, but the behavior exhibited was so different that they ended up being split into different clusters. For example, Figures 8.15 show two distinct samples of Gozi-ISFB contacting a reportedly malicious server located in Germany (46.38.238.142). They were split in clusters 13 and 14, which both contain incoming connections. The first sample receives a small packet followed by a series of 1200-byte packets. The second sample shows a periodic behavior where small packets are followed by large packets of range 600 to 1800 bytes. However, the cause of this split-personality behavior is unknown.



(a) Connection 1 with server

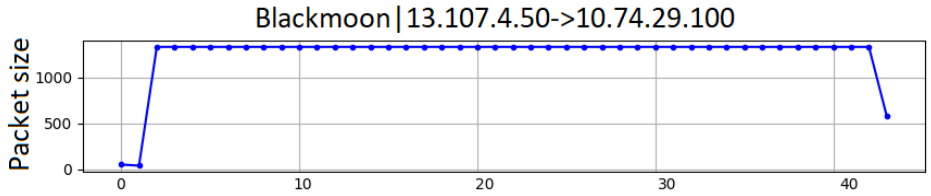


(b) Connection 2 with server

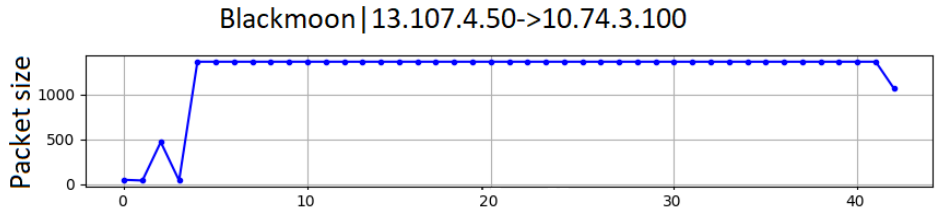
Figure 8.15: Example of a server showing split personalities at Connection-level

8.3.4. SAME C&C FOR MULTIPLE FAMILIES

A few instances were noted where connections coming from different malware families contacted the same C&C server, and they all ended up in a single cluster. For example, in cluster 10, three Zeus-Panda samples and two Blackmoon samples contact 13.107.4.50, which has been reported as a malicious IP by multiple threat intelligence websites. The behavior of the server is also very similar as shown in Figures 8.16. This suggests that either the YARA rules mislabeled one of the samples or that the authors of these samples are working in collaborations, such that they share the same C&C server.

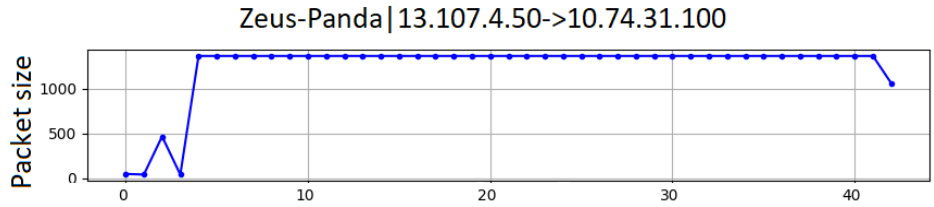


(a) Blackmoon sample 1

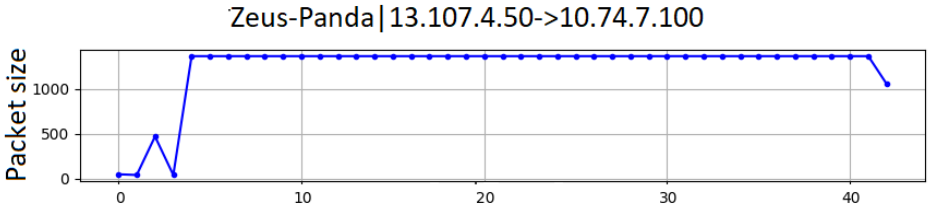


(b) Blackmoon sample 2

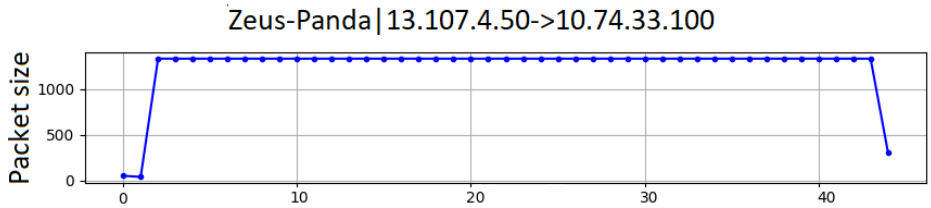
Figure 8.16: Zeus-Panda and Blackmoon sharing a C&C server



(c) Zeus-Panda sample 1



(d) Zeus-Panda sample 2

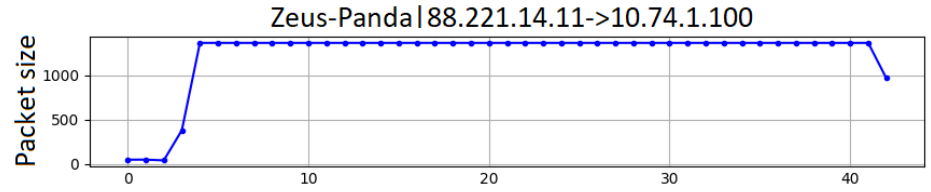


(e) Zeus-Panda sample 3

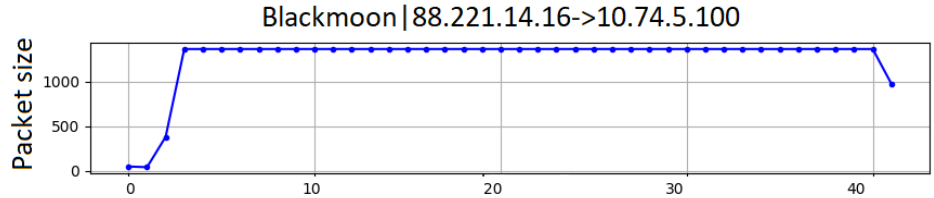
Figure 8.16: Zeus-Panda and Blackmoon sharing a C&C server (Cont.)

8.3.5. MALICIOUS SUBNETS

Several instances were noted where connections within the same cluster contacted IP addresses that fell in the same subnet. For example, in cluster 10, two Zeus-Panda samples contact 88.221.14.11 (Figure 8.17(a)), one Blackmoon sample contacts 88.221.14.16 (Figure 8.17(b)), and one Gozi-ISFB sample contacts 88.221.14.121 (Figure 8.17(c)). These examples suggest some collaboration among the authors of those malware samples.

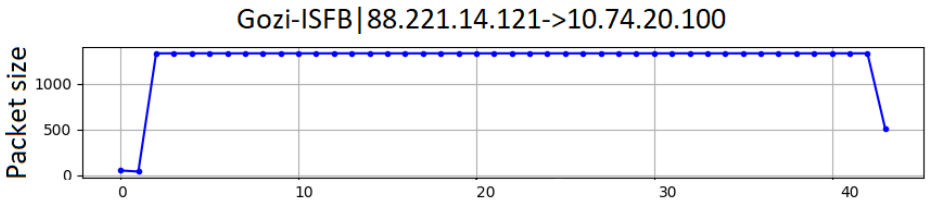


(a) Sample 1



(b) Sample 2

Figure 8.17: Potentially malicious subnet 88.221.14.XXX



(c) Sample 3

Figure 8.17: Potentially malicious subnet 88.221.14.XXX (Cont.)

8.4. ABC LABEL ANALYSIS

The family labels assigned by our data partner were done so on the basis of static analysis of the corresponding binary executables. Hence, each malware family can be considered as a cluster on the static analysis level. On the other hand, the goal of this research was to cluster malware based on its network behavior. Since the malware family labels are in such widespread use, it is only fair to have a comparison of this project's results with the family labels. Therefore, this section describes in detail how the malware families are split across the different resulting clusters.

The available families and their contribution to the dataset are given in Table 4.1. As explained in Section 7.2.3, the comparison is performed on two aspects: 1) How many types of behaviors are exhibited by a malware family (Uniformity/Diversity), and 2) How many malware families share the same kind of behaviors (Common/Rare).

8.4.1. PCAP-LEVEL CLUSTER SPLIT

The graphical representation of the Pcaps' family names and their contribution to the dataset is given in Figure 8.18. In addition, the sizes of the resulting 5 clusters are given in Figure 8.19. From the figure, it can be seen that cluster 3 is the largest with 40 Pcaps, while cluster 4 is the smallest with merely 7 Pcap files.

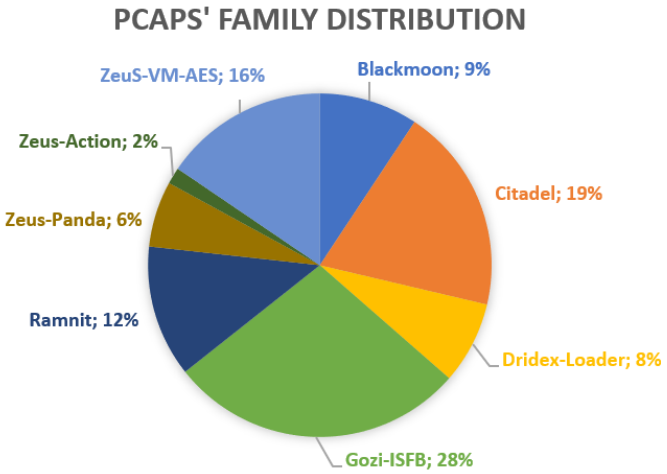


Figure 8.18: Malware families contributing to clusters at Pcap-level

A total of 8 families contribute to the clusters at Pcap-level, as shown in Figure 8.18. Pcaps belonging to the rest of the families were either too short to be considered or were discarded as noise. The split of each of the malware families into different clusters is shown in Figures 8.20.

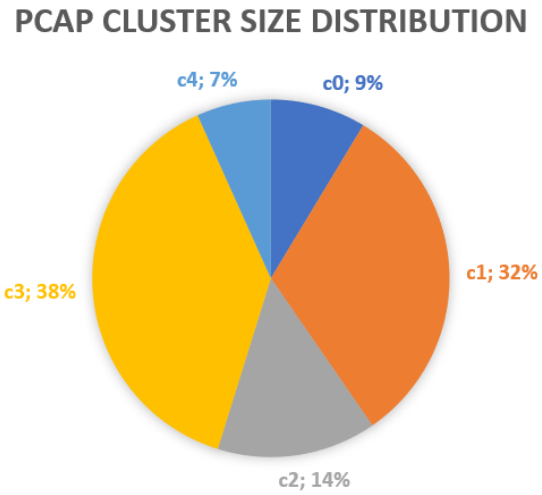
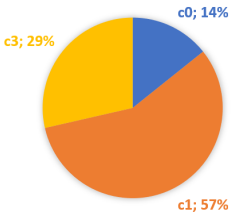


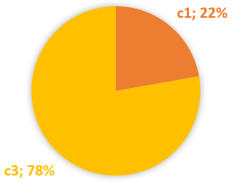
Figure 8.19: Cluster size distribution at Pcap-level

The pie chart of a malware family shows the percentage of Pcaps belonging to that family, assigned to different clusters. These percentages can also be considered as the various kinds of network behaviors exhibited by samples of that malware family.

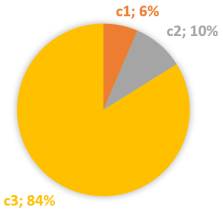
(a) Blackmoon



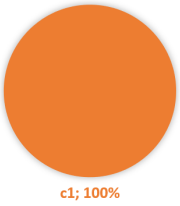
(b) Citadel



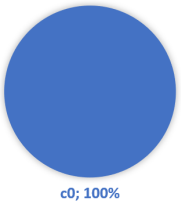
(c) Dridex-Loader



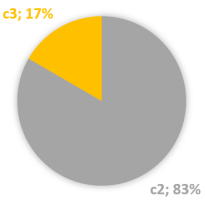
(d) Gozi-ISFB



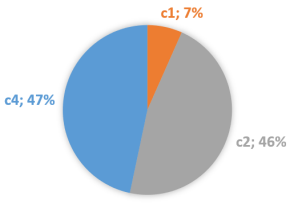
(e) Ramnit



(f) Zeus-Action



(g) Zeus-Panda



(h) Zeus-VM-AES

Figure 8.20: Malware family split across Pcap-level clusters

UNIFORMITY/DIVERSITY

Zeus-Action is the most concentrated family, as the two Pcaps that form this family fall in cluster 0. Similarly, 69% of Pcaps coming from Ramnit fall in cluster 1, while the rest are discarded as noise. Hence, only one kind of behavior is exhibited by the aforementioned malware families. Citadel, Gozi-ISFB, and Zeus-VM-AES, on the other hand, show the most diverse behaviors, as they get divided across 3 clusters (and some Pcaps being discarded as noise). Hence, it can be concluded that the families mentioned above show 3 out of 5 possible behaviors.

COMMON/RARE

Behavior captured by cluster 0 is the most common among all the 8 families – Pcaps from 6 out of 8 families fall in this cluster. On the other hand, cluster 4 captures behavior only exhibited by 35% Pcaps of Zeus-VM-AES – none of the other families show this behavior.

MISCELLANEOUS

Cluster 2 captures behavior common in Zeus-Panda, Zeus-VM-AES, and Gozi-ISFB. Moreover, cluster 3 is the largest among all the clusters. It is dominated by behavior common among Gozi-ISFB and Dridex-Loader, followed by a few Pcaps of Citadel and Zeus-Panda.

8.4.2. CONNECTION-LEVEL CLUSTER SPLIT

Similar to the Pcap-level case, a visual representation of connections belonging to different malware families is given in Figure 8.21. The percentage of each family's samples, at both Pcap-level (see Figure 8.18) and Connection-level (see Figure 8.21) is roughly the same. Some exceptions include Dridex and Zeus samples, which are present at the Connection-level but not at the Pcap-level because their combined connection lengths are not long enough for the Pcap-level. On the other hand, samples of Zeus-Action are present at the Pcap-level but not at the Connection-level because there are many smaller connections present in these samples that are not long enough for the Connection-level. Secondly, the family distribution at both granularities shows the dominance of Gozi-ISFB's samples (28% and 38% for both granularities, respectively), even though Blackmoon had the highest number of samples in the initial 1196-file dataset. This happened because most of the sequences generated from Blackmoon's samples were not long enough to be considered at either level of granularity.

The sizes of the resulting 18 clusters are given in Figure 8.22. The figure shows that cluster 1 is the largest with 90 connections, while cluster 10 is the smallest with merely 8 connections.

A total of 12 families contribute to the clusters formed at the Connection-level granularity. This is dramatically higher than Pcap-level since the minimum length of connections required to be considered in clustering is significantly shorter. The split of each malware families into different clusters is shown in Figures 8.23. The pie chart of a family shows the percentage of connections that are assigned to different clusters. These percentages can also be considered as the various kinds of network behaviors exhibited by connections of a malware family.

UNIFORMITY/DIVERSITY

Four families show concentrated behavior, which falls in only a single cluster. Dridex (made up of two connections) and Gozi-EQ (made up of one connection) fall in cluster 0. Similarly, Zeus-v1

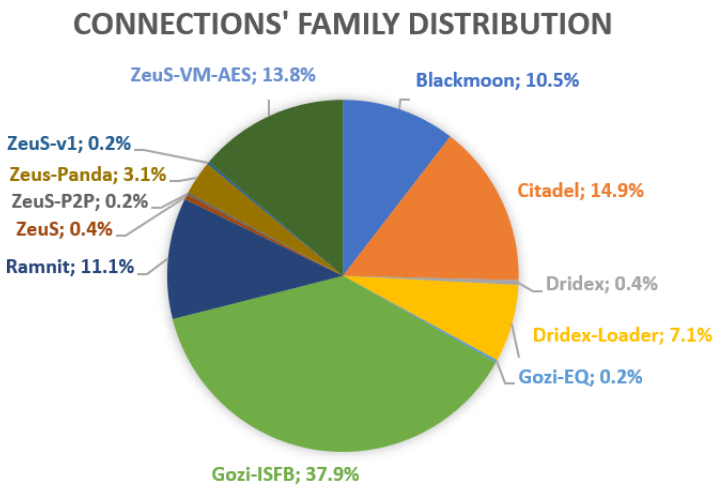


Figure 8.21: Malware families contributing to clusters at Connection-level

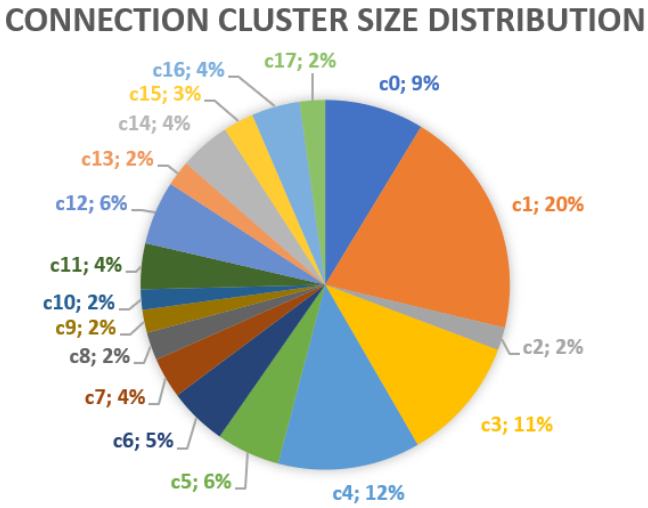
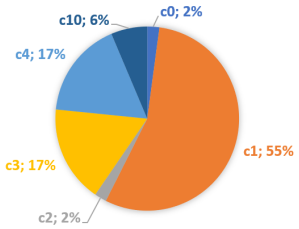


Figure 8.22: Cluster size distribution at Connection-level

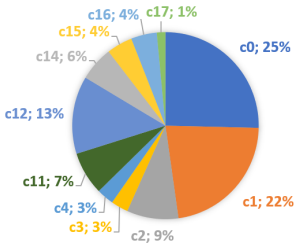
and Zeus-P2P, each made up of one connection, fall in cluster 1. In addition, Zeus is made up of two connections, where one falls in cluster 0 and the other in cluster 12. Clusters 0 and 1 capture behavior similar to standard Windows services, so it is possible that the malware in these samples is not yet activated. Hence, the only connections observed from these families seem benign. On the other hand, Gozi-ISFB shows the most diverse kinds of behaviors, with its connections falling in 16 out of 18 clusters – 16 distinct behaviors were observed from samples of Gozi-ISFB.

COMMON/RARE

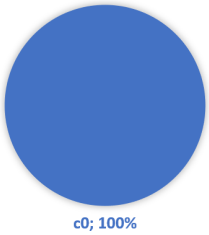
The most common clusters are 0 and 1, which capture behavior shown by Windows services. Since the malware families that are being analyzed are all Windows-based, it explains why 9 out of 12 families have connections in this cluster. Clusters 5, 6 and 8 capture behavior only exhibited by Gozi-ISFB. On top of that, one sample of Gozi-ISFB opens 111 connections (while the average number of connections per Pcap is 3), most of which fall in these clusters. Referring to Section 8.2.1, cluster 5 represents incoming connections while 6 and 8 represent outgoing traffic. Lastly, cluster 7 captures behavior only exhibited by samples of Zeus-VM-AES. Hence, these clusters can be used to



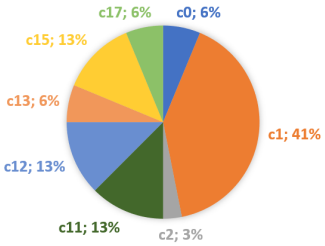
(a) Blackmoon



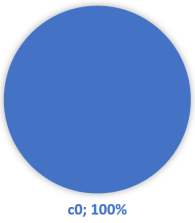
(b) Citadel



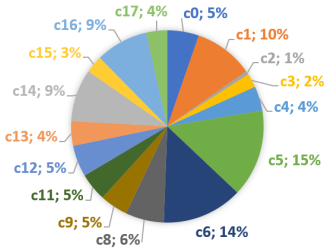
(c) Dridex



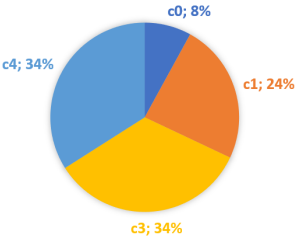
(d) Dridex-Loader



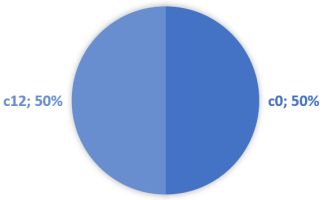
(e) Gozi-EQ



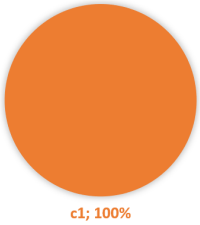
(f) Gozi-ISFB



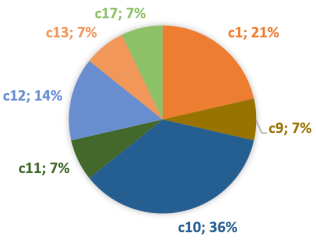
(g) Ramnit



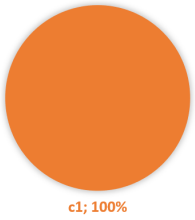
(h) Zeus



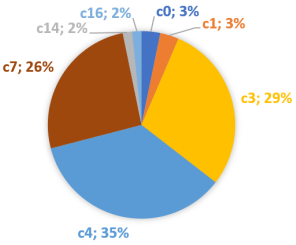
(i) Zeus-P2P



(j) Zeus-Panda



(k) Zeus-v1



(l) Zeus-VM-AES

Figure 8.23: Malware family split across Connection-level clusters

extract behavioral signatures that will help detect these unique malware samples.

MISCELLANEOUS

Cluster 2 captures behavior exhibited by four large malware families – Blackmoon, Citadel, Dridex-Loader, and Gozi-ISFB. Cluster 10 captures behavior exhibited only by Zeus-Panda and Blackmoon. Similarly, cluster 9 captures behavior exhibited only by Zeus-Panda and Gozi-ISFB. This indicates that some aspect of Zeus-Panda, Blackmoon, and Gozi-ISFB is similar, which would not have been

evident if we did not cluster their network behavior.

8.4.3. DISCUSSION

The benefit of clustering malware samples using their network behavior is twofold. First, it splits samples from one malware family across different clusters, showing the variety of behaviors exhibited by that single family. Second, it groups common network behaviors across different malware families, showing relationships between them. On the contrary, a malware family label assigned based on static analysis of the binary does not convey such detailed insights. Hence, the true benefit of this research is that malware can be grouped based on its capabilities, rather than its assigned family name.

8.5. BASELINE COMPARISON

Section 5.4.1 describes the feature set and the distance measure used for comparison with the baseline. The analysis is performed individually for each level of granularity. The parameters of the clustering algorithm are kept the same as mentioned in Sections 7.3 and 7.4 to allow objective comparison. The following subsections analyze how each of the sequence-as-features clusters is represented in the baseline version. In addition, cluster content analysis is performed on different clusters to understand what behaviors they represent. A high-level comparison between sequence and baseline clustering is presented in Table 8.1.

	Granularity	No. items	No. clusters	Average cluster size	Noisy items
Pcap-level clustering	**Discretized Sequential features**	219 Pcaps	5	20.8 Pcaps	25 Pcaps
	Non-discretized Sequential features		4	16 Pcaps	64 Pcaps
	Baseline		8	12.1 Pcaps	32 Pcaps
Connection-level clustering	**Sequential features**	733 Connections	18	24.9 connections	284 connections
	Baseline		20	19.9 connections	335 connections

Table 8.1: Comparison between Baseline and Sequence clustering

8.5.1. PCAP-LEVEL CLUSTERING

For minimum cluster size of 7 and 1 nearest neighbor, 8 clusters are formed. 32 Pcaps are discarded as noise, as opposed to 25 in the case of sequence clustering. The distribution of Pcaps in sequence clustering across different baseline clusters is given in Figure 8.24. The average cluster size is 12.12, as opposed to 20.8 for the sequence clusters. Cluster 0 is the largest, while clusters 5 and 6 are the smallest.

Table 8.2 shows the percentage of samples from sequence clustering split across baseline clusters. The first row states the clusters at the sequence clustering level, while the first column states the baseline clusters. Each cell shows the percentage of a sequence cluster split across different baseline clusters. For example, 44.4% samples from cluster 0 of sequence clustering are discarded

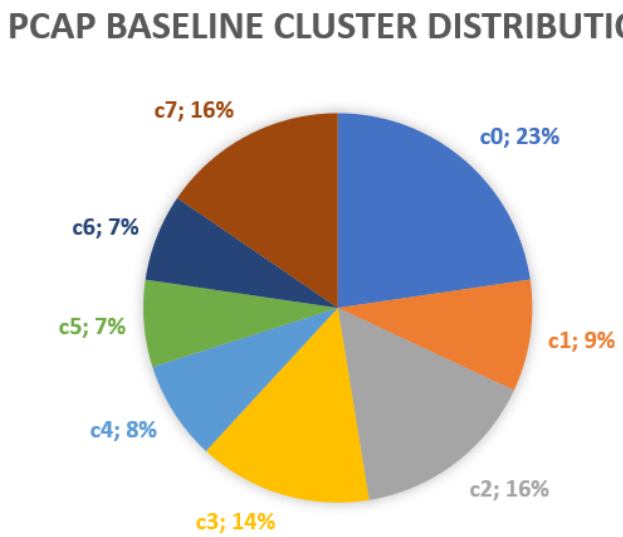
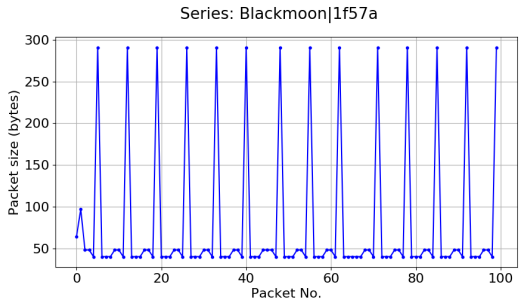


Figure 8.24: Baseline cluster size distribution at Pcap-level

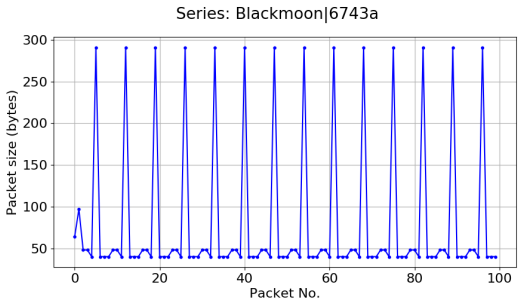
	Sequence clustering (%)					
	c0	**c1**	**c2**	**c3**	**c4**	**Noise**
c0	-	45.4	-	-	-	28
c1	-	24.2	-	-	-	4
c2	55.6	27.4	-	-	-	4
c3	-	-	-	27.5	-	12
c4	-	-	40	2.5	-	4
c5	-	-	26.7	2.5	-	8
c6	-	-	-	17.5	-	-
c7	-	-	26.7	22.5	-	8
Noise	44.4	3	6.6	27.5	100	32
Total	**100**	**100**	**100**	**100**	**100**	**100**

Table 8.2: Cluster split comparison between Baseline and Sequence clustering at Pcap-level

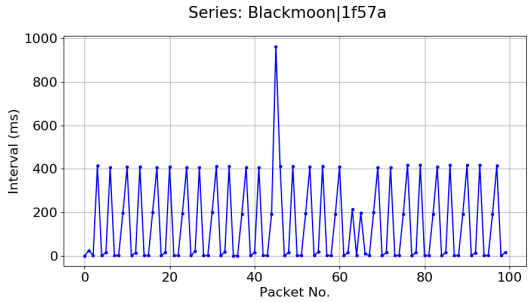
as noise in the baseline clusters. Upon traffic analysis, it is observed that these samples have a high periodicity in both packet sizes and the interval between packets. Two of those samples are shown in Figures 8.25, where (a) and (b) are the sequence of packet sizes, and (c) and (d) are the interval between packets. The rest of the samples in cluster 0 show a little less periodicity, so instead of discarding them as noise, baseline clusters discard the samples with clear periodicity, which is not a desirable result.



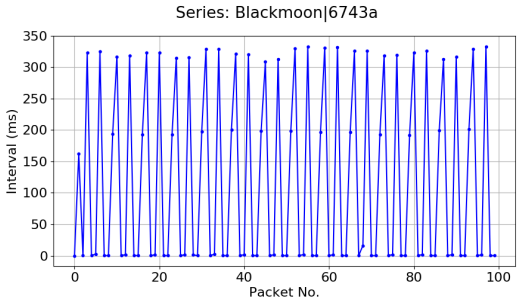
(a) Sample 1 - Packet sizes



(b) Sample 2 - Packet sizes



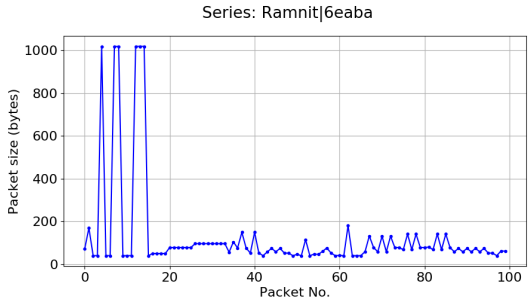
(c) Sample 1 - Interval



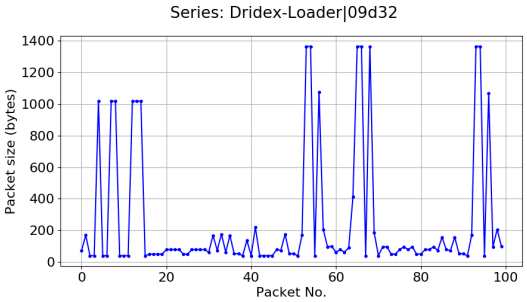
(d) Sample 2 - Interval

Figure 8.25: Similar-looking Pcaps from cluster 0 discarded as noise

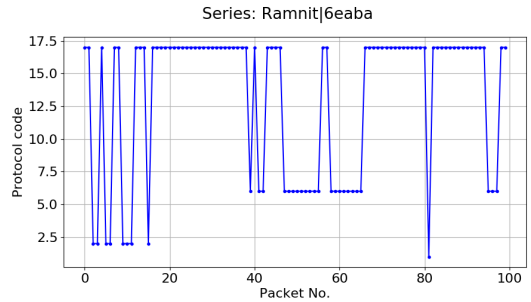
The table also shows that cluster 1 is split into 3 baseline clusters and some noise samples. For example, Figures 8.26 show the sequences of (a,b) packet sizes and (c,d) protocol numbers for two samples from this cluster. All three features of these two samples seem structurally very similar, yet they fall in different baseline clusters.



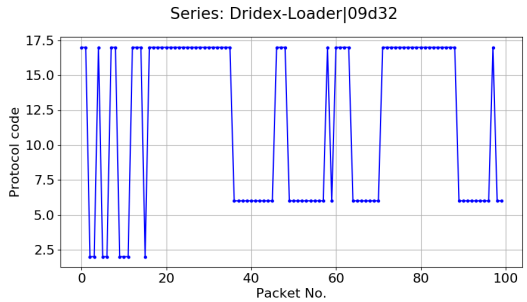
(a) Sample 1 - Packet sizes



(b) Sample 2 - Packet sizes



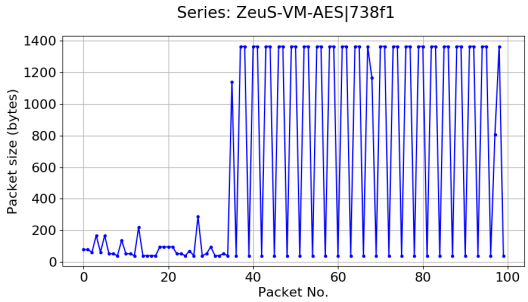
(c) Sample 1 - Protocol



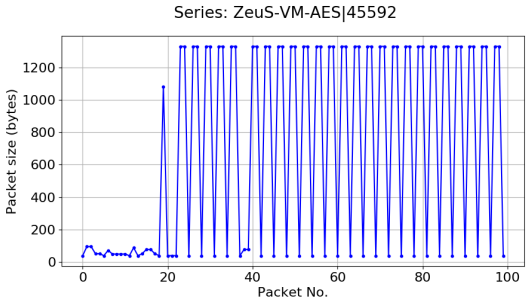
(d) Sample 2 - Protocol

Figure 8.26: Similar-looking Pcaps from cluster 1 split into smaller clusters

A few samples from each sequence cluster are discarded as noise. Moreover, cluster 4 is completely discarded as noise in the baseline clusters. Figures 8.27 show two samples from cluster 4 (sequence of packet sizes), which look so similar, yet they were discarded as noise.



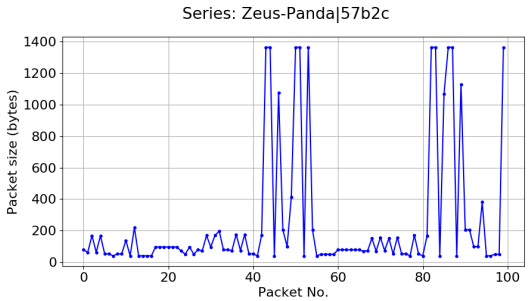
(a) Sample 1 - Packet sizes



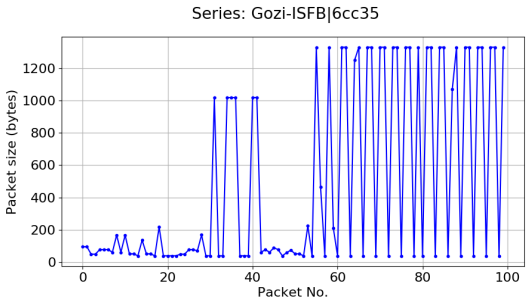
(b) Sample 2 - Packet sizes

Figure 8.27: Pcaps from cluster 4 discarded as noise

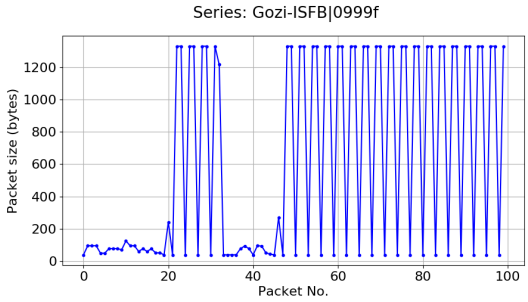
Cluster 3 gets split into as many as 5 clusters and a substantial amount of noise samples. Figures 8.28 show four samples (sequences of packet sizes), each of which ends up in a different baseline cluster without any apparent reason to have split them.



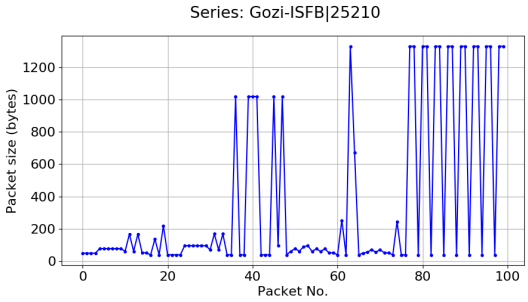
(a) Sample 1 - Packet sizes



(b) Sample 2 - Packet sizes



(c) Sample 3 - Packet sizes



(d) Sample 4 - Packet sizes

Figure 8.28: Similar-looking Pcaps from cluster 3 split into smaller clusters

In summary, clusters are further split into smaller baseline clusters whenever any small difference in features is observed. In many cases, the clusters are split unnecessarily resulting in a large number of smaller and similar clusters.

8.5.2. CONNECTION-LEVEL CLUSTERING

For minimum cluster size of 7 and 7 nearest neighbors, 20 clusters are formed. 335 connections are discarded as noise, as opposed to 284 in the case of sequence clustering. The distribution of connections of sequence clustering across different baseline clusters is given in Figure 8.29. The

average cluster size is 19.9, as opposed to 24.9 for the sequence clustering case. Cluster 3 is the largest, while cluster 1 is the smallest.

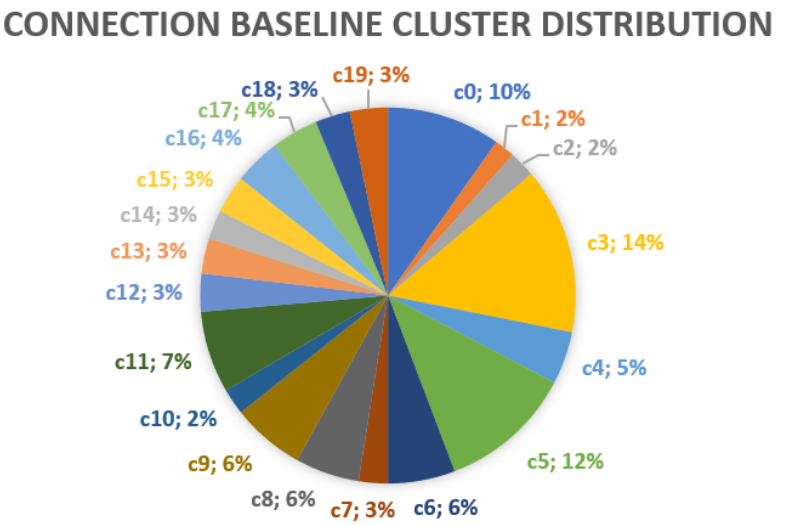


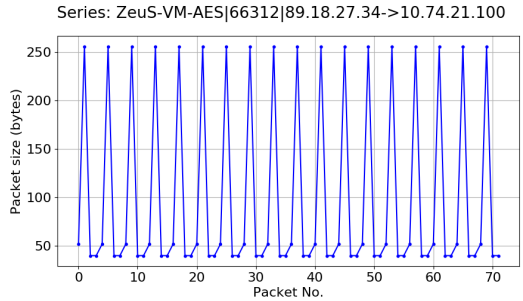
Figure 8.29: Baseline cluster size distribution at Connection-level

	Sequence clustering (%)																		
	c0	**c1**	**c2**	**c3**	**c4**	**c5**	**c6**	**c7**	**c8**	**c9**	**c10**	**c11**	**c12**	**c13**	**c14**	**c15**	**c16**	**c17**	**Noise**
c0	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c1	-	7.8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c2	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c3	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	0.4
c4	-	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c5	-	-	-	93.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c6	-	25.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c7	-	11.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
c8	-	-	-	-	-	-	82.6	-	-	-	-	-	-	-	-	-	-	-	1.1
c9	-	-	-	-	-	92	-	-	-	-	-	-	-	-	-	-	-	-	0.7
c10	-	-	-	-	-	-	-	-	-	-	-	-	24	-	-	-	-	-	1.1
c11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9.9
c12	-	-	-	-	-	-	-	-	-	-	-	22.2	-	-	-	-	-	-	3.2
c13	-	-	-	-	-	-	-	-	-	77.8	-	-	-	-	-	-	-	-	1.8
c14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	83.3	-	-	-
c15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10.5	100	0.4
c16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	84.2	-	-
c17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	80	-	-	-	-
c18	-	-	-	-	-	-	-	-	-	-	-	-	-	90	15	-	-	-	-
c19	-	-	-	-	-	-	-	-	-	-	-	-	48	-	-	-	-	-	0.4
Noise	-	35.6	-	6.1	-	8	17.4	100	100	22.2	100	77.8	28	10	5	16.7	5.3	-	81.3
Total	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**

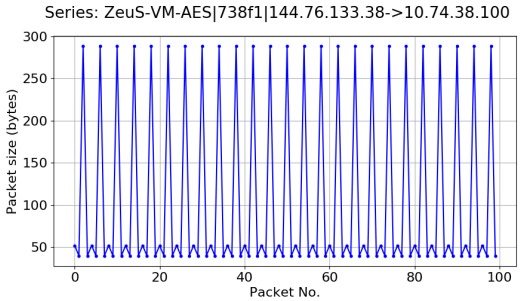
Table 8.3: Cluster split comparison between Baseline and Sequence clustering at Connection-level

Table 8.3 shows the percentage of samples from sequence clustering split across different baseline clusters. The first row states the clusters at the sequence clustering level, while the first column states the baseline clusters. Each cell shows the percentage of a sequence-as-features cluster split across baseline clusters. The table shows that clusters 0, 2, 4, and 17 are kept intact in the baseline clusters. Clusters 3, 5, 6, 9, 11, 13, and 15 are also mapped to a single cluster each, while also shaving away some connections as noise. For example, Figures 8.30 show two connections from cluster 3, one of which (Figure 8.30(a)) ends up as noise in the baseline clusters. The traffic from both these connections is a series of alternating DNS queries, followed by multiple TCP acknowledgement packets (ACK, FIN|ACK, SYN|ACK). Even the figures show the clear similarity in their sequence

structure, yet they are separated in the baseline clusters.



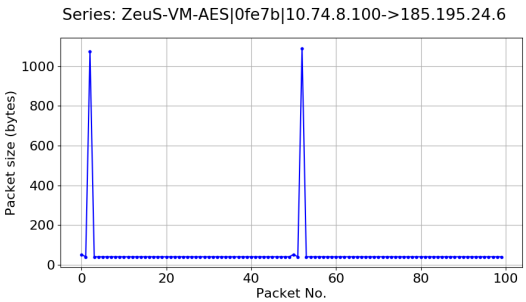
(a) Sample 1 - Packet sizes



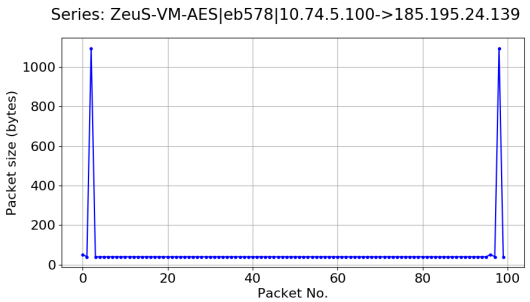
(b) Sample 2 - Packet sizes

Figure 8.30: Similar-looking Connections from cluster 3 discarded as noise

Clusters 7, 8, and 10 are completely discarded as noise, which is a shame since the connections in these clusters are very similar. Figures 8.31 show two samples from cluster 7 (sequence of packet sizes). The network traffic of both these samples starts with a POST request, followed by a number of large TCP packets. The fact that both these samples are connecting to hosts that are present in the same subnet (185.198.24.XX) provides even a more compelling reason of why they should have been grouped, but instead, they are discarded as noise. Similarly, Figures 8.32 show two samples from cluster 10 (sequence of packet sizes). The network traffic shows an HTTP 200 response code, followed by a number of large reassembled TCP packets. Again, these samples are structurally similar, but they were discarded as noise. A similar trend is also observed for connections of cluster 8.

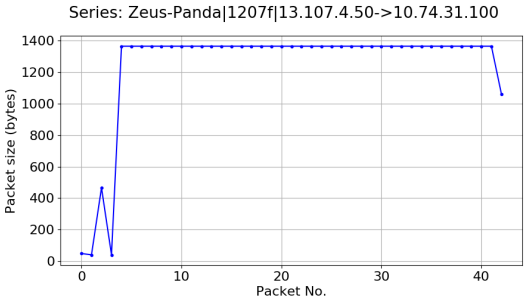


(a) Sample 1 - Packet sizes

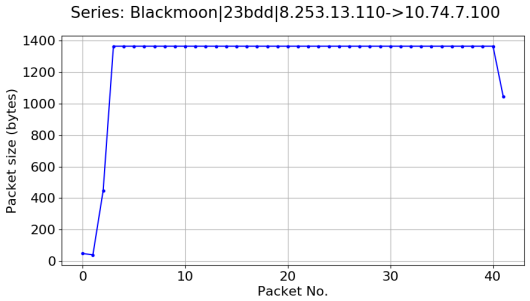


(b) Sample 2 - Packet sizes

Figure 8.31: Similar-looking Connections from cluster 7 discarded as noise



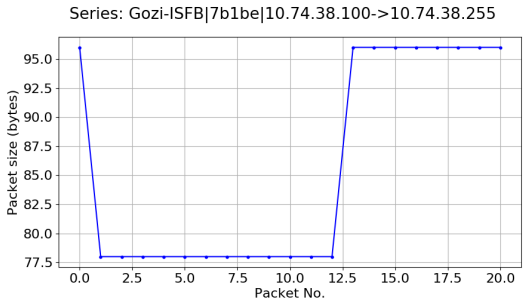
(a) Sample 1 - Packet sizes



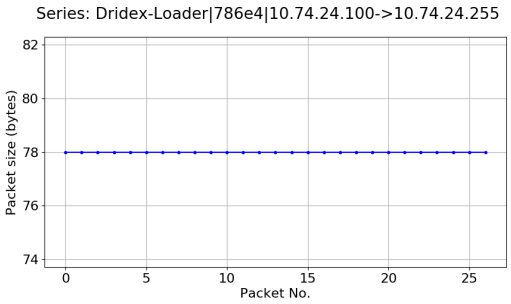
(b) Sample 2 - Packet sizes

Figure 8.32: Similar-looking Connections from cluster 10 discarded as noise

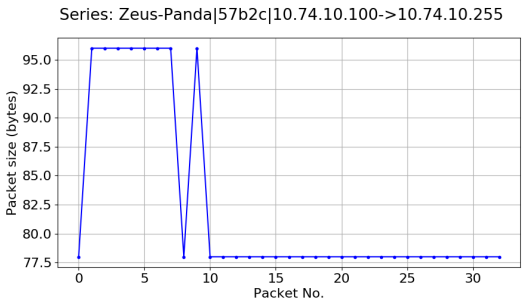
Cluster 1 is split across three clusters, and a significant number of connections are also discarded as noise. For example, Figures 8.33 show three samples from cluster 1 that end up in 3 different baseline clusters. Each sample is using NetBIOS name service (source and destination ports: 137) to retrieve a set of NetBIOS names. The packets are all of type NBNS and are querying for MSBrowser and PrintServer. In addition, the figures show the usage of two packet sizes (78 and 97 bytes), but the order in which they are used is different. Statistically, these three samples should have been similar, especially (a) and (c). Yet, using sequence clustering, these three samples were grouped, while baseline clusters split them into three distinct clusters even though the rest of the features are identical.



(a) Sample 1 - Packet sizes



(b) Sample 2 - Packet sizes



(c) Sample 3 - Packet sizes

Figure 8.33: Similar-looking Connections from cluster 1 split into smaller clusters

In summary, some of the baseline clusters map correctly to the sequence clusters. Moreover, 81.3% of connections discarded as noise in sequence clustering are also labeled as noise in baseline clusters. However, multiple clusters are either discarded as noise or are split across numerous baseline clusters even though the structure of the connections is extremely similar.

8.5.3. DISCUSSION

There is no drastic difference between the assignment of samples in sequence and baseline clusters. In fact, in some cases, the baseline clusters are exactly the same as sequence clusters. The incoming and outgoing connections are also clustered separately in baseline clustering. The main difference arises in the flexibility and leniency of the clusters. The baseline clusters are much more strict as compared to using sequence clustering. Because each sample is represented as a tuple of three to four numbers in the baseline, they are not able to handle the small variances arising from subtle differences in behavior. For example, if even a single feature is different, a different cluster is assigned to that sample altogether. Therefore, a lot more clusters are formed for the baseline version for exactly the same clustering parameters. Moreover, because of the stubborn nature of sample representation, the baseline clustering causes a lot more samples to be discarded as noise than the

sequence clustering. Using sequence clustering also allows leniency in the samples that are grouped – the clusters may seem to contain different behaviors when looking at one feature, but if enough features are 'similar', they get clustered together.

9

LIMITATIONS AND FUTURE WORK

In this chapter, we list down the limitations of our proposed approach and some future directions this project can take. We divide these limitations into three sections: 1) Dataset-related, which lists limitations arising due to the particular dataset used. These limitations can be fixed by choosing a different dataset; 2) Technique-related, which lists limitations in the proposed approach. These limitations can be fixed by tweaking the algorithm and conducting further research on individual components of the algorithm; and 3) Evaluation-related, which lists limitations in the way the research methodology has been evaluated. These can be fixed by coming up with metrics that formalize the validation of the methodology.

9.1. DATASET-RELATED

We did not collect the dataset used for this project. In fact, it was taken from an already collected larger dataset. The issue was that no one from ABC could tell us how the dataset was collected, what configurations were used to run the sandboxed environments, how long the malware samples were executed for, etc. Therefore, it was not clear how to distinguish between noise and malicious events in the data. So, it was assumed that any anomalous data is malicious and not noise due to being executed in a sandboxed environment. In order to make the analysis easier in future studies, the dataset needs to be collected by the researchers themselves, and there is a need for ground truth labels that are applicable for network traffic analysis.

Since the proposed technique is exploratory, we started off by experimenting with a small dataset. Out of 1.2M Pcap files, we analyzed 1196 files making up 15 malware families, out of which only about 220 files make it to the final analysis. Hence, a potential bias could be created because of the skewed dataset. However, because we are clustering the malware based on its network traffic, the family labels assigned to it are not even used. The proposed method is only concerned with the various behaviors exhibited by the Pcaps/connections, not their family labels.

In addition, the proposed technique is also not optimized for performance. Even though we perform trivial optimizations such as caching intermediate results and using relatively small-length sequences to compute distances, the proposed technique is in dire need of optimization before it can be deployed in the real world. Hence, a potential direction is to optimize the proposed technique for performance and to perform this study on a larger dataset. Techniques, such as Locality Sensitive Hashing can then be used to handle larger datasets by coarsely clustering them to reduce

the number of pairwise distance measurements.

9.2. TECHNIQUE-RELATED

Upon deeper analysis of the network traffic, we realized that the network traffic only shows the partial behavior of the malware – it only shows how the malware utilizes the Internet to conduct its objective, rather than a full view of its activities. Therefore, in order to understand the full behavior of the malware, the system-level behavior should also be an integral part of the analysis. We have left combining the network- and system-level analysis as future work because of the lack of expertise to analyze system-level activities at this point. To provide evidence that the system- and network- level activities indeed capture different aspects of malware, we have performed a comparison between network clusters and malware family labels.

Each level of granularity has its pros and cons. For example, Pcap-level captures a bird's eye view of the whole Pcap, but it suffers from noise pollution. On the other hand, Connection-level captures clearer behaviors, but the connections' relationships with each other are lost. Although our findings suggest that Connection-level is better for clustering attacking capabilities, if a malware's behavior is such that it communicates with multiple C&C servers for various purposes, Connection-level will treat each of those connections as unrelated. In addition, incoming and outgoing connections are handled separately, which gives only one-sided information. So, another potential extension is to experiment with Conversation-level granularity where sequences are generated from both incoming and outgoing packet flows.

The feature ranking performed is cursory due to the lack of time. Moreover, since the scope of this research was to cluster malware families together, no comparisons were made between malicious and benign traffic. We hypothesize that with the current feature set, similar Internet-usage behaviors, e.g., downloading a file, would be grouped, irrespective of the source being malicious or benign. Hence, additional research is warranted to identify the feature set that can not only cluster attacking capabilities but is also able to distinguish between malicious and benign traffic. Based on our experience with network traffic analysis, exploring packet fragmentation flags at the Network layer seems like a promising start. In addition, if this technique is to be deployed in real-world network monitoring systems, a noise-tolerance test is required to evaluate its resiliency in the presence of benign traffic.

The distance measure used to consolidate the distances of individual features is elementary. Based on the results of feature ranking and feature importance, each feature should be weighted to allow for a more customized distance measure between sequences. With the current unweighted average, each feature is considered equally important even though we have identified that certain features (e.g., protocol number at Pcap-level) do not contribute much in explaining the network behavior.

The three proposed discretization techniques are all based on percentiles and frequency. Other techniques, such as Kernel Density Estimation (KDE), which fits Gaussian curves based on regions of highest density can be evaluated. In addition, a meta-algorithm that can automatically switch between these different techniques can be developed, which will reduce the effort required to identify the best-suited discretization technique.

The length of the handshake is kept equal to the average sequence length of the whole dataset. For example, at the Connection-level, 20 packets are used to represent numeric features, and trigrams are used to represent categorical features. However, the longer the sequences are, the more

sequential behavior they will capture. For example, numeric features of length 50 were required to capture port scans. Hence, additional research is required to find the optimal length of the handshake.

Although the parameters for HDBScan were selected after experimentation, at both the Pcap- and Connection-level, some clusters overlapped each other, which introduced false positives. Even though the false positive ratio was 2.6% for the whole dataset, this number is expected to grow with the introduction of benign traffic and a larger dataset. Hence, we believe that the clustering can still be improved. Specific characteristics, such as better handling of noise points and avoiding clusters to overlap are good starting points for future research.

Lastly, a limitation of density-based clustering is that it considers lower-density regions as noise. Hence, if a limited number of malware samples exhibit strange (zero-day) attacking capabilities, they are more likely to have large distances with every other Pcap/connection in the dataset and be discarded as noise. Moreover, attacks are typically anomalous and rare events in general traffic. So the proposed technique is more likely to discard rare malicious events and only capture common Internet-usage behaviors (which may potentially all be benign). Hence, a combination of an anomaly detection system is essential to overcome this limitation.

9.3. EVALUATION-RELATED

The evaluation technique used in this research is preliminary and manual. The cluster quality estimation is based on visual analysis, which is prone to be subjective. The comparisons between Pcap- and Connection-level granularity are flimsy since Pcaps are being compared with connections, which are subsets of Pcaps – the comparison is not performed at the same level. However, due to lack of time, formalizing this section was left as future work. Hence, the first task in this area is to develop a cluster quality metric that is able to capture our definition of cluster 'goodness'. In addition, a framework needs to be designed that can merge distances from connection-level to Pcap-level to allow an objective comparison between the two level of granularities.

9.4. FUTURE VISION

The proposed technique was developed as part of a larger research study. Now that it is clear that higher-level sequential features can be used to cluster malware attacking capabilities, the evolution of malware attacking capabilities can be studied in detail using the proposed technique as its foundation. We expect that the evolution study will uncover unprecedented collaborations among malware authors and may also be used to infer attacker motivation. In addition, the proposed technique has the potential to shift the focus from system-level labeling of malware to network-level labeling of attacking capabilities. These set of labels are expected to be more consistent due to the uniformity of the Internet architecture in general.

10

CONCLUSIONS

Malware attacks are becoming more elusive and powerful [1]. In addition, the sheer number of malware samples that are detected on a daily basis is exceeding our capacity to analyze them [55]. Fortunately, a significant fraction of malware samples are variants of each other, which can be grouped in families [5]. In this way, we can discard malware samples belonging to a family that we have already analyzed.

Malware can be clustered using information retrieved from static analysis and dynamic analysis. The data source for dynamic analysis can either be system-level behavior or network-level behavior exhibited by a malware. Network activity shows the core behavior of malware as it captures the malware's interaction with its developer. It also does not induce additional overhead on end hosts, and the existing network monitoring infrastructure can be utilized for this purpose [22].

A number of different clustering approaches exist to group similar-looking and similar-behaving malware. Available literature [29, 33] shows an increasing emphasis towards Deep Packet Inspection, which is a controversial subject because of its severe privacy implications [56]. So, it is likely that it may not be allowed in the future.

Hence, we propose an exploratory technique that utilizes sequences of high-level, non-privacy-intrusive features to cluster malware based on their network behavior. Our intuition is that malware samples that use the same underlying infrastructure will perform actions in a similar order, which should be visible in high-level features, even in the presence of evasion techniques. The proposed approach ends up clustering different Internet-usage behaviors exhibited by multiple malware families, which can also be referred to as their *attacking capabilities*. Clustering similar attacking capabilities allows us to directly defend against that attack, rather than defending a system against a particular malware family. Hence, it provides a new way to categorize malware samples into families, which is in contrast to the labeling system based on static analysis of the binary executable.

We solve a number of challenges during the course of this research project. Our main contributions are:

- Identifying which level of granularity to construct the sequences at,
- Identifying which feature-set to use at those granularities,

- Proposing discretization techniques in order to convert numeric sequences to categorical ones,
- Identifying the appropriate distance measures to measure the distance between numeric and categorical sequences,
- Identifying which clustering algorithm performs best in extracting interpretable clusters,
- Performing a detailed analysis on which behaviors are clustered using the proposed approach, and
- Comparing the results of this research with clusters obtained from non-sequential features and malware family labels.

The answers to research questions posed in Section 1.5 are given below:

RQ: Are high-level sequential features effective in characterizing and clustering malware families' network behavior?

The proposed approach utilizes simple, sequential features to capture malware families' network behavior. In order to answer this question, one should define what a *'malware family'* means. A *malware family* is defined as a group of malware samples behaving similarly or originating from the same source. The malware samples present in the available dataset were labeled using static analysis of their binary executables. However, the goal of this project is to cluster the network behavior exhibited by malware samples, which does not have a one-to-one mapping with the static analysis – different families can behave similarly if their objectives are similar, and one malware family can behave in multiple ways corresponding to each of its objective. Therefore, given the definition that a malware family is composed of malware samples whose network behavior is similar, the proposed technique is effective in characterizing those behaviors. The results show that the clusters represent common attacking capabilities among malware families, rather than clustering malware families themselves (Here malware families essentially refer to labels assigned using static analysis). Upon reflection, we believe that for organizations, knowing what kind of attack is being performed on their network would better equip them to defend it, rather than knowing which malware family is attacking them.

RQ1: What level of granularity is best to characterize malware families' network behavior?

In this project, we evaluated two level of granularities – Pcap-level and Connection-level. Upon cluster analysis, we found that Pcap-level captures high-level, bird's eye view of the whole Pcap, while Connection-level drills down to a deeper level and shows a much clearer behavioral pattern. The analysis with connection-level can become resource-intensive due to the sheer amount of connections coming from a few Pcap files. However, we were able to analyze the connections in much more detail and identify attacking capabilities such as port scans, a C&C server controlling different families, and a C&C server behaving differently at different times. We also detected several subnets that seem to host C&C servers. In conclusion, Connection-level granularity, while having its downsides, still outperforms Pcap-level in capturing malware attacking capabilities.

RQ2 (a): Which high-level feature-set characterizes a malware's network traffic?

In order to compensate for the abstractness of high-level features, we decided to leverage the power of sequences-as-features. Our intuition is that the malware samples related to one another will perform actions in a similar order, which will be captured aptly with sequences rather than singular values. Several features were evaluated that are suggested in the literature, extracted from the headers of different layers of the OSI model [30, 31, 33, 37]. We discovered that the feature-set chosen is dependent on the level of granularity being considered for the sequence construction. Hence, at the Pcap-level granularity, a triple sequence of (`packet size, interval between packets, protocol number`) was chosen as the feature set, and at the Connection-level granularity, a quadruple sequence of (`packet size, interval between packets, source port, destination port`) best characterized the malware behavior. In addition, we also identified that discretizing numeric features at the Pcap-level granularity helps clear away noise and results in better-quality clusters.

RQ2 (b): What is the difference between using features that are represented by sequences versus those that are not?

A baseline was created to evaluate the effectiveness of sequence clustering at both levels of granularity. The features at the baseline were represented using aggregated singular values, instead of sequences. The rest of the parameter settings were kept constant. A comparison between the clusters obtained from sequence-as-features versus baseline was performed. The results showed that using sequence clustering allowed robust and lenient clusters, compared to the baseline version. The results also showed overall similar clusters in both settings. However, due to the stubborn nature of feature representation in the baseline version, a lot more Pcaps/connections were discarded as noise, and smaller clusters were formed.

RQ3: Which distance captures the (dis)similarity in sequences at the various granularities considered?

The feature-set at both levels of granularities is a combination of numeric and categorical sequences. A number of different distance measures were evaluated to define the notion of 'similarity' in this regard. For numeric features, we found Dynamic Time Warp distance to aptly measure the distance, while Ngram analysis (N=3) and cosine distance were used to measure the distance between categorical features. In order to consolidate the distances calculated for each Pcap/connection, a simple unweighted average was used.

RQ4 (a): What kind of behaviors are visible in the clustered malware samples resulting from the proposed clustering approach?

The proposed clustering approach is able to cluster similar attacking capabilities exhibited by different malware families. At the Pcap-level, a lot of repetitive behavior is seen in the sequence of packet sizes and the interval between packets. Simple behaviors, such as initiating contact with the C&C server and downloading malicious configuration files are responsible for that periodicity. No specific behavioral insight is gained from the protocol-switching behavior. However, at the Connection-level, the clusters can be divided in a quadrant having incoming and outgoing connections at one axis while HTTP-based and other-protocol-based malware at the other axis. The clusters are generally well-separated, and a false positive rate of 2.6% is observed. Each cluster shows a distinct type of behavior. Incoming and outgoing connections are clustered separately. The

sequence of port numbers shows systematic and randomized port scans being conducted. Several IP addresses, which are potentially C&C servers, are seen behaving differently at different times. Samples from several malware families are also seen contacting the same IP address. These insights suggest that either the malware family label assigned to those samples is incorrect or that those malware families work in collaboration with each other. Lastly, the clusters also revealed a few subnets that are likely to be composed of C&C servers.

RQ4 (b): How different is the malware samples' membership to clusters resulting from network analysis versus static analysis?

Each Pcap was assigned a family label based on its static analysis, which may or may not correspond to its network-level behavior. The benefit of clustering malware samples using their network traffic rather than their family labels is twofold. First, the clustering splits samples from one family across different clusters, showing the variety of behaviors exhibited by that single family. Second, it groups common network behaviors among different malware families, showing relationships between them. On the contrary, a malware family label assigned based on static analysis of the binary does not convey such detailed insights. Hence, dynamic, network-based sequence clustering is far more qualified to identify the many behaviors exhibited by one family, as well as common behaviors among multiple families.

In conclusion, the exploratory study performed in this thesis shows that high-level sequential features are capable of characterizing and clustering the attacking capabilities of malware families. The resulting clusters are robust and can uncover potential collaborations among malware families. However, heavy optimization is required in order to make the clustering method ready to be deployed in the real world.

A

Family label, Hash and IP address Mapping

Family name	Encoded to
Blackmoon	B
Citadel	C
Dridex	D
Dridex-Loader	DL
Gozi-EQ	GE
Gozi-ISFB	GI
Ramnit	R
ZeuS	Z
ZeuS-v1	Z1
Zeus-Action	ZA
ZeuS-P2P	ZP2
Zeus-Panda	ZPA
ZeuS-VM-AES	ZVA
ZeuS-OpenSSL	ZO
Dridex-RAT-FakePin	DRF

Table A.1: Mapping of Malware family names to the first three characters of their names

Hash value	Encoded to
4c7d6	0
2d63f	1
5b537	2
5606c	3
738f1	4
11536	5
6ee08	6
bddf2	7
372f8	8

4c1fa	9
ebe83	10
05509	11
0bc7c	12
6e474	13
4bd77	14
16dad	15
3e636	16
4385a	17
78de2	18
6eaba	19
6b252	20
2d600	21
5ef1e	22
57b2c	23
3604d	24
26d08	25
3c914	26
41b22	27
26135	28
5e34b	29
0bdc3	30
52946	31
34391	32
0fe7b	33
eb578	34
dc1f6	35
6713d	36
5644f	37
11c95	38
1590b	39
16f4a	40
67189	41
20716	42
2c58c	43
1dece	44
5441c	45
6a081	46
5b125	47
24346	48
13a67	49
5a3be	50
34ff3	51
32e63	52
eb4e1	53
45592	54
6ff80	55

786e4	56
2edc3	57
163fd	58
61db2	59
4c3bf	60
564ba	61
6fe95	62
6a7d4	63
6cc35	64
0da1e	65
a1590	66
400af	67
3b639	68
a07d1	69
5d171	70
44547	71
33110	72
0999f	73
0e6dd	74
36688	75
11a32	76
61cef	77
2a05a	78
3ed99	79
a11e1	80
79204	81
30c18	82
5b277	83
44c3c	84
25a27	85
7b1be	86
6743a	87
078ea	88
11286	89
03e0c	90
76b53	91
06fe3	92
1f57a	93
17b0e	94
5dfbe	95
13199	96
4b386	97
5a7d4	98
0fd3b	99
3b588	100
478f5	101
2c52d	102

7754d	103
09d32	104
6263c	105
66312	106
112a1	107
244a6	108
6fa66	109
409ec	110
22a10	111
44cf9	112
1fb3c	113
042f0	114
75b4d	115
5cd55	116
490da	117
7a3ea	118
43b17	119
2d327	120
648a6	121
a3ef9	122
1207f	123
75ed9	124
3feac	125
25210	126
3a042	127
1e1e4	128
334ea	129
461c2	130
1b3b1	131
43ff0	132
15c03	133
6763c	134
063fa	135
35ac7	136
69bf8	137
a0563	138
035d8	139
3e6c3	140
026f4	141
5d931	142
474c1	143
18701	144
31d64	145
53dad	146
1048e	147
06896	148
dcee0	149

74b28	150
6d00f	151
7bdf3	152
e4709	153
a1f87	154
a0152	155
a11b6	156
2db47	157
00e92	158
20660	159
1df9f	160
76170	161
5965b	162
3108e	163
1e6a8	164
360ce	165
3fd36	166
5896b	167
041bf	168
5272c	169
63ede	170
dc3dc	171
716e1	172
08a47	173
4bd80	174
40181	175
53310	176
21f0f	177
0ea2b	178
6896a	179
2cd5c	180
27f0d	181
5f925	182
4a191	183
1d653	184
68e84	185
18782	186
70238	187
ebbb8	188
431b7	189
45f63	190
0eeb1	191
336a7	192
560e2	193
59012	194
59dd1	195
b09ff	196

68029	197
11b75	198
4a84f	199
009ea	200
54f41	201
1e008	202
588c6	203
62c89	204
32503	205
63003	206
72a53	207
5dc20	208
03d2d	209
6791f	210
177fe	211
23bdd	212
75487	213
20ec0	214
6f3ae	215

Table A.2: Mapping of hash values to numbers

IP address	**Encoded to**
94.23.148.62	0
23.219.162.170	1
10.74.23.100	2
148.81.111.121	3
10.74.54.100	4
103.17.117.29	5
10.74.45.100	6
88.221.254.203	7
2.22.22.82	8
13.107.4.50	9
172.217.17.78	10
134.102.200.101	11
47.91.111.171	12
74.125.206.139	13
74.125.206.138	14
62.173.141.42	15
94.73.148.160	16
10.74.34.100	17
188.114.140.245	18
2.22.146.113	19
2.22.146.112	20
31.15.10.10	21
92.48.108.28	22
62.113.203.55	23

103.56.139.8	24
103.244.148.179	25
75.27.136.151	26
92.53.112.89	27
45.79.84.186	28
54.192.203.182	29
23.219.88.153	30
159.203.42.254	31
62.138.144.162	32
23.67.250.139	33
88.221.15.19	34
193.42.156.106	35
13.69.159.30	36
50.7.178.98	37
211.115.106.9	38
211.115.106.8	39
211.115.106.7	40
211.115.106.6	41
211.115.106.5	42
211.115.106.4	43
211.115.106.3	44
211.115.106.2	45
211.115.106.1	46
10.74.33.100	47
67.229.228.218	48
95.100.156.11	49
46.4.99.46	50
78.142.63.63	51
72.246.56.120	52
195.110.124.188	53
193.124.181.50	54
144.217.254.208	55
216.58.198.174	56
209.126.119.168	57
10.74.32.255	58
130.255.73.90	59
216.58.201.36	60
23.62.62.167	61
62.149.144.108	62
23.96.219.115	63
104.93.82.160	64
198.15.236.140	65
198.15.236.141	66
66.102.1.102	67
10.74.18.100	68
74.125.206.102	69
184.51.1.48	70

198.41.215.184	71
198.41.215.185	72
184.51.1.41	73
198.41.215.183	74
103.56.139.16	75
103.56.139.17	76
34.194.213.50	77
144.217.56.140	78
10.74.49.100	79
217.182.45.165	80
10.74.20.100	81
23.229.23.2	82
123.110.49.77	83
149.202.2.106	84
104.223.114.11	85
10.74.53.100	86
94.240.170.216	87
5.9.147.226	88
62.210.213.17	89
54.192.203.88	90
2.22.49.14	91
103.226.155.56	92
18.181.5.37	93
108.61.164.218	94
2.20.188.188	95
72.247.184.121	96
185.25.50.5	97
213.136.80.109	98
47.91.121.168	99
23.4.241.89	100
52.174.55.168	101
63.234.248.163	102
151.80.147.153	103
23.4.241.80	104
64.233.184.102	105
5.135.183.146	106
64.233.184.101	107
46.166.167.46	108
88.221.112.131	109
14.17.41.150	110
2.22.48.177	111
141.138.157.53	112
47.91.104.57	113
10.74.5.100	114
95.59.26.76	115
95.46.114.129	116
67.198.227.220	117

62.149.140.138	118
2.22.23.73	119
216.58.208.228	120
10.74.10.255	121
216.58.198.196	122
51.254.202.160	123
10.74.29.255	124
216.58.208.227	125
10.74.7.255	126
10.74.23.255	127
62.149.128.160	128
93.119.123.193	129
10.74.30.255	130
178.63.97.34	131
147.32.5.111	132
87.106.190.153	133
85.195.207.92	134
91.103.125.24	135
203.205.151.50	136
72.14.182.233	137
40.68.38.156	138
192.35.177.195	139
10.74.34.255	140
46.101.6.132	141
81.31.147.141	142
10.74.24.255	143
23.14.84.49	144
74.125.206.100	145
74.125.206.101	146
94.23.204.175	147
2.22.146.123	148
185.129.60.131	149
192.88.99.1	150
10.74.20.255	151
85.159.213.210	152
76.73.17.194	153
163.172.156.198	154
1.164.149.14	155
91.121.230.216	156
74.91.21.2	157
195.154.181.146	158
10.74.3.255	159
2.22.22.218	160
2.22.22.219	161
10.74.29.100	162
5.178.43.18	163
81.176.239.224	164

195.8.222.232	165
54.192.203.236	166
54.192.203.238	167
87.98.175.85	168
46.249.205.50	169
2.16.4.176	170
185.81.0.99	171
185.64.219.6	172
23.81.66.90	173
188.138.75.101	174
2.16.4.178	175
10.74.25.100	176
188.120.243.35	177
10.74.16.100	178
10.74.47.100	179
23.215.133.9	180
10.74.30.100	181
216.58.206.131	182
31.31.73.222	183
50.62.111.1	184
163.172.146.169	185
91.134.135.12	186
46.38.238.142	187
204.188.136.44	188
208.185.118.105	189
31.11.32.99	190
185.156.179.126	191
23.215.130.114	192
37.187.30.78	193
72.167.191.65	194
216.58.213.142	195
66.102.1.138	196
66.102.1.139	197
104.86.110.251	198
10.74.55.255	199
10.74.8.255	200
92.122.122.146	201
92.122.122.147	202
89.223.27.179	203
23.62.6.16	204
2.22.22.105	205
63.243.244.8	206
63.243.244.9	207
185.195.24.6	208
31.11.32.73	209
209.48.71.250	210
88.86.120.181	211

184.51.0.250	212
212.112.245.170	213
64.69.66.200	214
85.204.74.132	215
10.74.40.255	216
78.47.18.110	217
104.93.82.8	218
10.74.19.255	219
5.9.88.74	220
85.214.236.207	221
24.234.152.249	222
80.239.216.34	223
136.243.214.137	224
10.74.39.255	225
217.182.73.92	226
80.231.244.27	227
46.4.34.242	228
10.74.7.100	229
8.253.44.158	230
104.86.110.74	231
111.231.53.226	232
23.219.162.168	233
10.74.36.255	234
144.76.133.38	235
62.149.140.210	236
178.165.72.60	237
8.254.194.174	238
35.187.55.75	239
72.249.193.34	240
23.62.6.171	241
40.74.50.25	242
37.48.122.22	243
10.74.5.255	244
62.210.142.39	245
23.62.6.179	246
54.192.203.68	247
23.55.149.163	248
37.115.202.156	249
2.22.23.25	250
69.181.245.46	251
213.246.56.62	252
89.223.109.60	253
71.17.184.96	254
77.123.108.42	255
239.255.255.250	256
107.167.87.242	257
89.163.224.250	258

2.22.146.40	259
109.74.196.143	260
45.55.8.14	261
10.74.15.100	262
67.131.160.25	263
2.22.146.48	264
10.74.40.100	265
193.23.244.244	266
10.74.13.255	267
90.154.128.74	268
31.11.32.119	269
119.28.86.77	270
64.71.164.5	271
10.74.24.100	272
2.22.22.65	273
198.98.53.5	274
103.244.148.70	275
37.59.236.156	276
195.133.146.57	277
148.251.193.100	278
10.74.37.100	279
109.74.195.149	280
173.223.52.43	281
23.219.88.107	282
198.15.236.121	283
103.24.94.36	284
111.90.140.7	285
23.192.125.168	286
85.227.129.23	287
23.3.96.11	288
88.221.14.168	289
216.58.198.164	290
194.87.237.127	291
85.235.225.239	292
10.74.27.255	293
72.246.56.35	294
40.77.64.160	295
2.16.106.139	296
49.51.38.83	297
45.32.28.232	298
2.21.67.81	299
94.23.144.49	300
104.168.87.167	301
23.215.132.91	302
136.243.176.148	303
104.16.91.188	304
10.74.16.255	305

139.59.36.57	306
50.87.37.56	307
162.246.57.206	308
104.40.208.40	309
23.215.132.235	310
86.105.1.11	311
62.113.203.99	312
5.196.159.173	313
5.196.159.175	314
104.91.166.81	315
137.116.77.120	316
2.16.4.40	317
23.206.242.58	318
10.74.11.255	319
184.83.3.50	320
23.23.170.235	321
103.226.155.43	322
91.207.7.81	323
192.87.28.82	324
88.221.14.193	325
23.215.130.235	326
51.15.135.103	327
216.58.213.131	328
8.253.86.14	329
10.74.22.100	330
10.74.55.100	331
61.74.49.180	332
10.74.44.100	333
176.9.17.142	334
23.61.194.147	335
10.74.15.255	336
2.22.22.137	337
104.155.38.253	338
10.74.37.255	339
185.68.144.62	340
138.197.202.35	341
10.74.9.255	342
173.223.106.227	343
217.73.238.12	344
195.154.255.174	345
64.233.184.113	346
176.31.163.89	347
65.153.18.99	348
2.20.188.172	349
10.74.19.100	350
99.116.60.120	351
67.198.128.252	352

54.192.203.22	353
199.15.250.210	354
151.101.0.233	355
185.107.224.208	356
216.58.208.238	357
10.74.10.100	358
46.173.214.226	359
104.27.138.75	360
10.74.51.100	361
10.74.32.100	362
23.219.162.153	363
192.99.108.183	364
209.170.111.8	365
184.50.238.217	366
10.74.2.255	367
10.74.22.255	368
62.149.140.244	369
5.9.49.12	370
178.16.208.56	371
31.11.34.21	372
89.223.27.249	373
62.149.142.160	374
154.35.32.5	375
198.41.214.184	376
202.218.50.130	377
38.229.70.53	378
172.217.23.14	379
212.111.30.190	380
2.21.67.57	381
2.16.4.112	382
88.221.144.18	383
107.154.113.172	384
66.110.99.19	385
208.67.222.222	386
88.221.144.10	387
88.221.14.121	388
27.133.240.230	389
104.238.186.189	390
70.32.39.218	391
88.221.14.128	392
54.192.203.163	393
2.22.146.75	394
10.74.4.100	395
10.74.9.100	396
87.72.239.187	397
10.74.35.255	398
31.11.32.129	399

23.215.133.64	400
67.131.160.19	401
184.51.0.243	402
172.217.17.142	403
23.74.28.25	404
63.217.21.41	405
216.58.201.238	406
94.242.254.91	407
54.192.203.111	408
99.126.22.157	409
10.74.1.255	410
89.163.247.43	411
5.196.71.24	412
118.214.160.203	413
72.246.56.107	414
69.31.132.16	415
69.31.132.10	416
204.79.197.200	417
77.246.163.142	418
54.192.203.142	419
23.3.96.57	420
2.16.106.51	421
40.113.17.180	422
5.150.221.137	423
75.56.51.58	424
94.23.186.184	425
185.72.178.171	426
88.221.14.16	427
31.11.32.88	428
72.246.56.10	429
198.105.184.30	430
185.141.25.23	431
66.102.1.101	432
23.215.130.179	433
185.133.72.100	434
216.58.212.174	435
198.105.184.26	436
198.105.184.25	437
23.50.225.25	438
217.70.144.15	439
209.85.202.113	440
198.105.184.28	441
122.114.152.92	442
96.90.175.167	443
89.18.27.34	444
204.188.136.221	445
54.72.46.30	446

99.103.223.24	447
10.74.35.100	448
10.74.3.100	449
116.255.164.235	450
54.192.203.65	451
67.225.166.132	452
10.74.50.100	453
88.99.199.87	454
23.219.163.24	455
10.74.4.255	456
118.214.160.195	457
10.74.11.100	458
31.128.74.100	459
184.51.1.10	460
45.63.99.180	461
23.61.194.178	462
94.242.255.112	463
171.25.193.9	464
2.18.65.112	465
217.73.238.20	466
10.74.12.255	467
23.215.130.99	468
10.74.21.100	469
10.74.1.100	470
91.209.77.11	471
216.58.210.35	472
216.18.70.74	473
93.115.91.66	474
144.76.253.229	475
216.58.208.206	476
115.126.14.26	477
47.91.124.165	478
115.126.14.25	479
185.15.72.62	480
198.41.215.182	481
88.221.15.73	482
46.91.215.207	483
137.116.74.190	484
88.221.15.75	485
23.219.162.104	486
10.74.2.100	487
23.54.18.42	488
185.141.25.242	489
155.133.38.226	490
91.229.20.27	491
72.247.9.121	492
23.215.132.11	493

31.11.32.166	494
63.217.21.26	495
138.201.169.12	496
91.121.23.100	497
69.42.64.26	498
99.92.50.150	499
69.42.64.24	500
10.74.28.100	501
103.56.139.15	502
78.156.117.236	503
104.86.110.64	504
51.141.32.51	505
46.235.9.33	506
10.74.46.100	507
62.149.142.84	508
10.74.31.100	509
46.101.100.94	510
83.162.202.182	511
173.212.206.230	512
10.74.48.100	513
198.15.236.139	514
10.74.27.100	515
213.66.28.170	516
79.250.130.36	517
84.201.32.108	518
54.192.203.153	519
68.168.221.222	520
89.163.225.6	521
10.74.21.255	522
2.18.213.59	523
123.58.180.166	524
94.242.58.233	525
92.222.180.87	526
194.109.206.212	527
23.111.11.204	528
72.246.64.177	529
37.191.160.173	530
212.47.229.2	531
188.240.208.89	532
2.16.4.152	533
10.74.12.100	534
92.122.212.27	535
197.2.66.166	536
164.132.77.175	537
10.74.18.255	538
185.159.129.195	539
50.7.151.127	540

23.204.138.56	541
52.166.120.77	542
2.16.4.72	543
194.94.127.98	544
62.149.140.188	545
10.74.17.255	546
168.235.146.20	547
23.61.194.250	548
10.74.26.255	549
194.50.97.16	550
96.126.117.198	551
10.74.33.255	552
10.74.28.255	553
10.74.6.100	554
184.105.192.2	555
122.114.84.167	556
195.225.171.30	557
104.80.88.152	558
66.7.218.146	559
198.105.184.16	560
104.16.93.188	561
89.22.100.64	562
118.214.160.216	563
200.35.156.207	564
93.63.162.60	565
35.198.166.240	566
52.179.17.38	567
5.178.43.17	568
136.243.106.162	569
2.20.188.140	570
23.61.194.10	571
62.233.121.75	572
2.21.67.59	573
31.11.33.18	574
91.134.243.173	575
173.224.119.211	576
173.224.119.212	577
107.180.41.148	578
10.74.38.255	579
188.166.82.61	580
195.123.218.172	581
23.14.90.114	582
103.24.94.38	583
13.65.245.138	584
5.9.25.79	585
216.58.206.142	586
62.149.128.166	587

51.15.177.148	588
23.219.162.146	589
216.58.198.206	590
184.25.56.91	591
46.252.26.2	592
47.90.82.159	593
184.25.56.99	594
10.74.17.100	595
185.86.150.78	596
178.255.83.2	597
178.66.0.167	598
37.187.20.164	599
213.108.41.176	600
54.192.203.251	601
88.221.14.11	602
67.229.104.211	603
180.163.251.231	604
10.74.42.100	605
208.80.154.39	606
193.183.98.154	607
208.83.223.34	608
23.219.162.99	609
10.74.26.100	610
96.6.45.57	611
54.192.203.211	612
95.100.156.8	613
23.215.133.73	614
93.170.96.235	615
23.67.250.121	616
10.74.6.255	617
178.254.9.25	618
10.74.43.100	619
104.20.1.85	620
2.22.22.48	621
104.27.147.174	622
203.130.58.30	623
23.217.129.17	624
8.250.15.254	625
185.45.192.116	626
92.122.122.154	627
92.122.122.137	628
92.122.122.136	629
72.246.56.137	630
104.223.114.3	631
10.74.38.100	632
2.22.22.144	633
23.74.2.66	634

63.234.248.160	635
208.118.235.148	636
10.74.14.100	637
216.58.213.163	638
10.74.41.100	639
72.246.56.186	640
31.11.32.228	641
23.215.130.146	642
66.102.1.113	643
178.16.208.62	644
5.196.20.5	645
8.253.13.110	646
103.17.117.246	647
92.122.212.19	648
64.233.184.100	649
224.0.0.22	650
23.38.103.25	651
84.22.184.152	652
209.85.202.106	653
209.85.202.101	654
209.85.202.100	655
80.83.118.233	656
209.85.202.102	657
85.204.74.158	658
121.42.51.120	659
209.107.217.74	660
192.35.177.64	661
31.193.131.147	662
64.233.184.139	663
162.159.211.43	664
185.20.225.124	665
69.73.130.134	666
119.28.81.30	667
98.139.183.24	668
62.149.142.166	669
103.7.30.86	670
152.195.32.39	671
62.210.254.132	672
23.74.2.120	673
185.61.149.12	674
23.215.132.162	675
198.167.140.243	676
115.126.14.81	677
93.184.220.29	678
185.21.216.157	679
45.123.118.101	680
23.38.103.32	681

10.74.47.255	682
173.223.106.200	683
173.223.106.201	684
217.12.223.202	685
10.74.14.255	686
192.87.28.28	687
212.51.134.123	688
224.0.0.252	689
188.165.200.156	690
23.50.225.17	691
23.219.163.83	692
198.55.107.156	693
198.41.214.185	694
93.170.13.22	695
62.149.128.151	696
10.74.13.100	697
62.149.128.154	698
62.149.128.157	699
142.91.104.107	700
192.42.116.41	701
209.85.203.94	702
23.50.225.9	703
51.255.41.91	704
141.20.33.67	705
95.163.121.162	706
54.91.240.28	707
52.41.240.139	708
23.94.5.133	709
115.159.4.107	710
10.74.8.100	711
185.195.24.139	712
23.97.178.173	713
87.245.242.234	714
194.58.100.73	715
54.202.79.255	716
207.154.244.147	717
51.255.48.78	718
178.62.252.82	719
103.244.148.71	720
38.140.184.17	721
23.43.69.163	722
40.69.25.97	723
62.210.141.69	724
2.18.213.64	725
31.11.32.214	726
2.22.22.209	727
62.149.142.58	728

54.192.203.143	729
104.93.82.19	730
45.63.25.55	731
8.8.8.8	732
72.246.64.187	733
62.149.128.74	734
62.149.128.72	735
88.221.113.49	736
10.74.39.100	737
89.45.67.144	738
209.48.71.121	739
23.111.11.211	740
2.22.146.19	741
10.74.25.255	742
37.139.24.90	743
206.190.36.45	744
138.201.132.17	745
23.204.138.9	746
23.219.162.89	747
95.215.111.125	748
103.52.216.15	749
138.197.133.81	750
217.23.7.103	751
10.74.36.100	752
5.149.216.60	753
185.118.66.84	754
23.229.147.161	755
173.223.52.18	756
144.76.184.104	757
2.16.4.187	758
74.88.96.7	759
37.122.208.220	760
178.255.83.1	761
89.46.105.15	762
54.192.203.136	763
62.210.92.11	764
62.149.128.163	765
80.239.216.58	766
23.219.88.82	767
23.219.88.81	768
10.74.52.100	769
112.90.78.177	770
23.215.130.216	771
23.215.132.138	772
118.214.160.251	773
111.206.66.62	774
86.59.21.38	775

72.246.56.170	776
198.50.146.252	777
23.62.6.27	778
64.233.184.138	779
8.253.129.66	780
104.16.90.188	781
118.166.47.54	782
208.67.183.204	783
198.41.214.183	784
23.61.194.27	785
31.11.32.116	786
198.41.214.187	787
67.229.60.202	788
211.115.106.11	789
211.115.106.10	790
23.43.75.27	791
183.131.79.137	792
89.223.29.55	793
92.122.192.81	794
188.241.58.10	795
165.254.52.106	796
69.156.240.29	797
216.58.209.238	798
195.110.124.133	799
10.74.31.255	800
23.74.28.8	801
62.210.82.244	802
45.56.117.118	803

Table A.3: Mapping of IP addresses to numbers

B

Cover Picture shows a Port scan

The inspiration for the cover picture came from the heatmaps generated during cluster content visualization. The cover page shows a real-life port scan taking place. So, it goes to show that the results of this project are quite useful and the visualizations, artistic.

Following is the original version of the cover, showing the family labels and hashes responsible for the port scan.

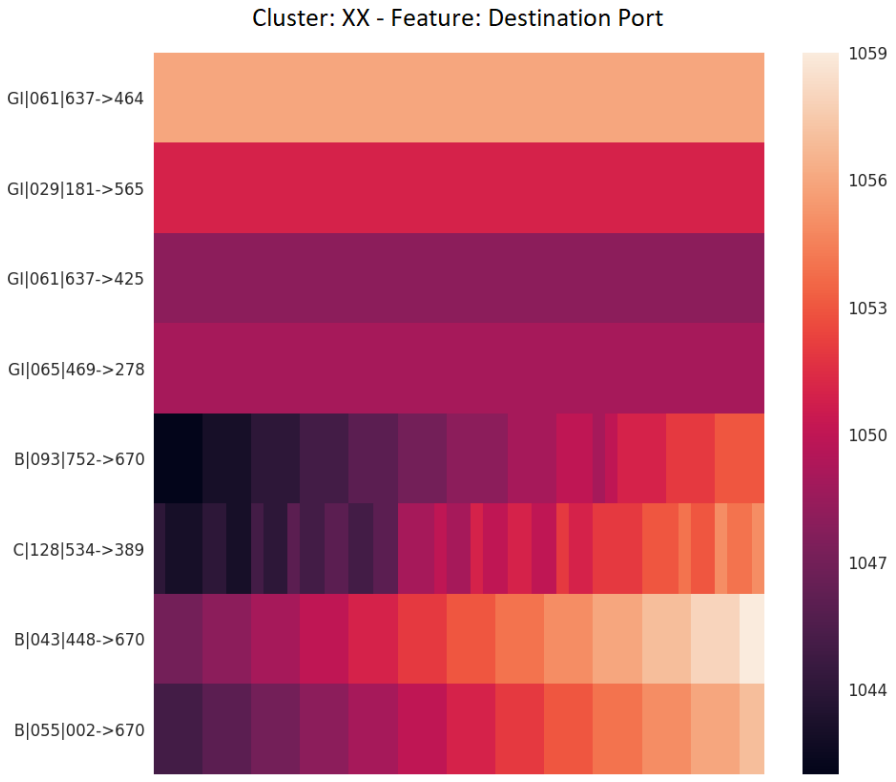


Figure B.1: Cover picture's original heatmap

BIBLIOGRAPHY

[1] J. Crowe, *10 must-know cybersecurity statistics for 2018,* (2018).

[2] R. Benzmüller, *In 2017 every 4.2 seconds a new malware specimen emerges,* (2018).

[3] B. Brenner, *2018 malware forecast: ransomware hits hard, continues to evolve,* (2018).

[4] S. Tajalizadehkhoob, H. Asghari, C. Gañán, and M. Van Eeten, *Why them? extracting intelligence about target selection from zeus financial malware,* in *Proceedings of the 13th Annual Workshop on the Economics of Information Security, WEIS 2014, State College (USA), June 23-24, 2014* (WEIS, 2014).

[5] Y. Li, J. Jang, X. Hu, and X. Ou, *Android malware clustering through malicious payload mining,* in *International Symposium on Research in Attacks, Intrusions, and Defenses* (Springer, 2017) pp. 192–214.

[6] M. Sun, X. Li, J. C. Lui, R. T. Ma, and Z. Liang, *Monet: a user-oriented behavior-based malware variants detection system for android,* IEEE Transactions on Information Forensics and Security **12**, 1103 (2017).

[7] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonell, *The effectiveness of source code obfuscation: An experimental assessment,* (2009).

[8] P. M. Comparetti, G. Salvaneschi, E. Kirda, C. Kolbitsch, C. Kruegel, and S. Zanero, *Identifying dormant functionality in malware programs,* in *2010 IEEE Symposium on Security and Privacy (SP)* (IEEE, 2010) pp. 61–76.

[9] R. Perdisci, W. Lee, and N. Feamster, *Behavioral clustering of http-based malware and signature generation using malicious network traces.* in *NSDI*, Vol. 10 (2010) p. 14.

[10] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, *A static, packer-agnostic filter to detect similar malware samples,* in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (Springer, 2012) pp. 102–122.

[11] T. Ho, S.-J. Cho, and S.-R. Oh, *Parallel multiple pattern matching schemes based on cuckoo filter for deep packet inspection on graphics processing units,* IET Information Security (2018).

[12] C. Yu, J. Lan, J. Xie, and Y. Hu, *Qos-aware traffic classification architecture using machine learning and deep packet inspection in sdns,* Procedia computer science **131**, 1209 (2018).

[13] S. B. Krishnamurthy, *Deep packet inspection with enhanced data packet analyzers,* (2018), uS Patent App. 15/479,306.

[14] H. Ling and Z. Chen, *Method to enable deep packet inspection (dpi) in openflow-based software defined network (sdn),* (2018), uS Patent 9,871,764.

[15] C. Fuchs, *Implications of deep packet inspection (dpi) internet surveillance for society.* (PACT, 2012).

[16] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, *Detection of new malicious code using n-grams signatures.* in *PST* (2004) pp. 193–196.

[17] Q. Zhang and D. S. Reeves, *Metaaware: Identifying metamorphic malware,* in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual* (IEEE, 2007) pp. 411–420.

[18] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, *Dendroid: A text mining approach to analyzing and classifying code structures in android malware families,* Expert Systems with Applications **41**, 1104 (2014).

[19] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, *Detecting code reuse in android applications using component-based control flow graph,* in *IFIP International Information Security Conference* (Springer, 2014) pp. 142–155.

[20] A. Gupta, P. Kuppili, A. Akella, and P. Barford, *An empirical study of malware evolution,* in *First International Communication Systems and Networks and Workshops (COMSNETS 2009)* (2009) pp. 1–10.

[21] C. S.-E. T.-R. Johnson, *Improved malware clustering using virustotal metadata,* .

[22] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, *Scalable, behavior-based malware clustering.* in *NDSS*, Vol. 9 (Citeseer, 2009) pp. 8–11.

[23] N. Wong Hon Chan, *Scanner: Sequence clustering of resource access to find nearest neighbors,* (2015).

[24] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, and K. Bobrovnikova, *A technique for the botnet detection based on dns-traffic analysis,* in *International Conference on Computer Networks* (Springer, 2015) pp. 127–138.

[25] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, *Botfinder: Finding bots in network traffic without deep packet inspection,* in *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (ACM, 2012) pp. 349–360.

[26] A. Mohaisen, O. Alrawi, and M. Mohaisen, *Amal: High-fidelity, behavior-based automated malware analysis and classification,* computers & security **52**, 251 (2015).

[27] M. Korczyński and A. Duda, *Markov chain fingerprinting to classify encrypted traffic,* in *Infocom, 2014 Proceedings IEEE* (IEEE, 2014) pp. 781–789.

[28] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, *Disclosure: detecting botnet command and control servers through large-scale netflow analysis,* in *Proceedings of the 28th Annual Computer Security Applications Conference* (ACM, 2012) pp. 129–138.

[29] A. A. Ghorbani and S. Nari, *Automated malware classification based on network behavior,* in *2013 International Conference on Computing, Networking and Communications (ICNC)* (IEEE, 2013) pp. 642–647.

[30] W. Cui, J. Kannan, and H. J. Wang, *Discoverer: Automatic protocol reverse engineering from network traces.* in *USENIX Security Symposium* (2007) pp. 1–14.

[31] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, *Inferring protocol state machine from network traces: a probabilistic approach,* in *International Conference on Applied Cryptography and Network Security* (Springer, 2011) pp. 1–18.

[32] A. Andoni, *Locality sensitive hashing (lsh) home page,* (2018).

[33] X. Jiang and Y. Zhou, *Dissecting android malware: Characterization and evolution,* in *2012 IEEE Symposium on Security and Privacy* (IEEE, 2012) pp. 95–109.

[34] J. Lee and H. Lee, *Gmad: Graph-based malware activity detection by dns traffic analysis,* Computer Communications **49**, 33 (2014).

[35] B. Anderson, S. Paul, and D. McGrew, *Deciphering malware's use of tls (without decryption),* Journal of Computer Virology and Hacking Techniques **14**, 195 (2017).

[36] Y. LI and W. HAO, *Research of encrypted network traffic type identification,* Journal of Computer Applications **29**, 1662 (2009).

[37] P. Black, I. Gondal, and R. Layton, *A survey of similarities in banking malware behaviours,* Computers & Security (2017).

[38] R. Dash, R. L. Paramguru, and R. Dash, *Comparative analysis of supervised and unsupervised discretization techniques,* International Journal of Advances in Science and Technology **2**, 29 (2011).

[39] L. Peng, W. Qing, and G. Yujia, *Study on comparison of discretization methods,* in *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on*, Vol. 4 (IEEE, 2009) pp. 380–384.

[40] S. Kotsiantis and D. Kanellopoulos, *Discretization techniques: A recent survey,* GESTS International Transactions on Computer Science and Engineering **32**, 47 (2006).

[41] G. Pellegrino, Q. Lin, C. Hammerschmidt, and S. Verwer, *Learning behavioral fingerprints from netflows using timed automata,* in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on* (IEEE, 2017) pp. 308–316.

[42] S. F. Chaerul Haviana, *Sistem gesture accelerometer dengan metode fast dynamic time warping (fastdtw),* JURNAL SISTEM INFORMASI BISNIS **5** (2015), 10.21456/vol5iss2pp151-160.

[43] S. M. Harding, W. B. Croft, and C. Weir, *Probabilistic retrieval of ocr degraded text using n-grams,* Research and Advanced Technology for Digital Libraries , 345 (1997).

[44] M. Schonlau and N. Guenther, *Text mining using n-grams,* SSRN Electronic Journal (2016), 10.2139/ssrn.2759033.

[45] G. Volis, C. Makris, and A. Kanavos, *Two novel techniques for space compaction on biological sequences,* Proceedings of the 12th International Conference on Web Information Systems and Technologies (2016), 10.5220/0005801101050112.

[46] R. J. Campello, D. Moulavi, and J. Sander, *Density-based clustering based on hierarchical density estimates,* in *Pacific-Asia conference on knowledge discovery and data mining* (Springer, 2013) pp. 160–172.

[47] P. J. Rousseeuw, *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,* Journal of computational and applied mathematics **20**, 53 (1987).

[48] D. L. Davies and D. W. Bouldin, *A cluster separation measure,* IEEE transactions on pattern analysis and machine intelligence , 224 (1979).

[49] D. Basulto, *Humans are the world's best pattern-recognition machines, but for how long?* (2018).

[50] C. Hennig, *Measurement of quality in cluster analysis,* (2018).

[51] R. Yamada and S. Goto, *Using abnormal ttl values to detect malicious ip packets,* Proceedings of the Asia-Pacific Advanced Network **34**, 27 (2013).

[52] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, and K. Bobrovnikova, *A technique for the botnet detection based on dns-traffic analysis,* in *International Conference on Computer Networks* (Springer, 2015) pp. 127–138.

[53] L. van der Maaten, *t-sne,* (2018).

[54] F. Blogs and Research, *How the rise of cryptocurrencies is shaping the cyber crime landscape:blockchain infrastructure use « how the rise of cryptocurrencies is shaping the cyber crime landscape:blockchain infrastructure use,* (2018).

[55] E. Gandotra, D. Bansal, and S. Sofat, *Malware analysis and classification: A survey,* Journal of Information Security **5**, 56 (2014).

[56] R. T. G. Collins, *The privacy implications of deep packet inspection technology: Why the next wave in online advertising shouldn't rock the self-regulatory boat,* Ga. L. Rev. **44**, 545 (2009).