Malware Behavior Through Network Trace Analysis

Xiyue Deng[1] and Jelena Mirkovic[2]

[1] xiyueden@isi.edu
[2] mirkovic@isi.edu
Information Sciences Institute, University of Southern California

Abstract. Malware continues to be a major threat to information security. To avoid being detected and analyzed, modern malware is continuously improving its stealthiness. A high number of unique malware samples detected daily suggests a likely high degree of code reuse and obfuscation to avoid detection. Traditional malware detection techniques relying on binary code signatures are greatly hindered by encryption, packing, code polymorphism, and similar other obfuscation techniques. Although obfuscation greatly changes a malware's binary, its functionalities remain intact.

We propose to study malware's network behavior during its execution, to understand the malware's functionality. While malware may transform its code to evade analysis, we contend that its key network behaviors must endure through the transformations to achieve the malware's ultimate purpose, such as sending victim information, scanning for vulnerable hosts, etc. While live malware analysis is risky, we leverage the Fantasm platform on the DeterLab testbed to perform it safely and effectively. Based on observed network traffic we propose an encoding of malware samples. This encoding can help us classify malware flows and samples, identify code reuse and genealogy, and develop behavioral signatures for malware defense. We apply our approach to more than 8,000 diverse samples from the Georgia Tech Apiary project. We find that over 60% of malware is multi-purposed (e.g. downloading new payload and uploading user data). We also illustrate how our encoding and malware flow clustering can be used to identify behavioral signatures for malware defense.

Keywords: malware, network, polymorphic, genealogy

1 Introduction

The Internet is facing increasing threats due to the proliferation of malware. A study by Kaspersky Lab suggests an estimated daily increase of over 360,000 samples in the wild in 2017. [11]. Such high malware production rate suggests that new malware may be created by transforming existing code to evade signature detection.

Malware designers have invested enormous efforts to change their binaries to avoid detection and analysis by defenders. From simple instruction transformation, such techniques have evolved through junk code injection, code obfuscation, to polymorphic and metamorphic engines that can transform malware code into millions of variants [19]. Such obfuscation techniques have greatly undermined traditional signature-based malware detection methods, and they create an enormous workload for code analysis. Yet much of the new malware variants could simply be old malware in a new package, or new malware assembled from pieces of old malware. Clearly, as new malware samples emerge, code analysis and signature-based filtering cannot keep pace.

Besides static analysis, researchers have used dynamic analysis to overcome code obfuscation. Dynamic analysis includes tracking disk changes, analyzing dynamic call graphs, as well as monitoring malware execution using debuggers and virtual machines. However, modern malware is often equipped with anti-debugging and anti-virtualization capabilities [20,21], making dynamic code analysis in a controlled environment difficult.

To complement contemporary static and dynamic *code* analysis, we propose to study malware behavior by observing and interpreting its *network activity*. Much of today's malware relies on the network connectivity to achieve its purpose, such as sending reports to the malware author, joining the botnet, sending spam and phishing emails, etc. We hypothesize that it would be difficult for malware to significantly alter its network behavior and still achieve its purpose. Studying network behavior thus may offer an opportunity to both understand what malware is trying to do in an analysis environment, and to develop behavior or network traffic signatures useful for malware defense.

Live malware analysis, with unrestricted network access, is risky, because malware may inflict damage to other Internet hosts during analysis, and analysts would become unwitting accomplices. We leverage the Fantasm system for safe and productive live malware analysis [6] to minimize this risk. The system enables us to emulate unrestricted connectivity from malware's point of view, while it protects the Internet from unwanted traffic.

Just having network traffic records of a malware run is not enough to understand a malware's purpose, because such data is very rich and unstructured. To overcome this challenge, we develop an *embedding* for a malware sample – a set of flow-record features observed during the analysis. We then perform a two-pronged analysis. First, we devise patterns over the embeddings that help us detect presence and features of high-level network behaviors, such as file download, e-mail sending, scanning, etc. We use a set of these high-level behaviors to infer a malware sample's purpose. We note that malware could have more than one purpose. For example, it could both scan for vulnerable hosts and send unsolicited emails. Thus a malware sample could have more than one label. Second, we use the embeddings to identify malware samples that have the same or very similar network behaviors, and study code reuse and malware genealogy.

We perform our analysis on 8,172 malware samples, randomly selected from the Georgia Tech Apiary project [1]. 63.9% of the samples exhibit more than one

behavior, which speaks to the multi-purpose nature of contemporary malware. We further cluster malware flows based on their embeddings and identify groups that have similar or even identical behaviors. We find that only 14 clusters are responsible for over 80% of data-carrying flows, which appear in about 70% of malware samples. Apart from these flows, the samples exhibit diverse other network behaviors, ranging from repeatedly querying `google.com`, to uploading private user information, to utilizing Bittorrent tracker to scan for potential victims. We then show how our clusters can be used to devise behavioral signatures for malware defenses.

2 Background and Related Work

Malware analysis has received continuous increase of research interests [3,17,24]. In this section, we discuss contemporary malware analysis methods and the rationale behind our approach. We also survey related work.

2.1 Static Analysis

A common approach to malware detection is analyzing binaries for *code signatures* – sequences of binary code that are present in malware and are not common in benign binaries. Various static analysis methods like CTPL [12], Generic Virus Scanner [5], etc. have been used to analyze binary code of malware without running it, and identify portions that could be used for signature generation. The identified portion is then synthesized to become the signature of this kind of malware.

Signature-based malware detection has been the most widely used method and has been quite successful. However, malware designers have also been working on countermeasures over the years to undermine such techniques. From junk code generation, malware encryption and oligomorphism, to polymorphic and metamorphic malware [19], such techniques have evolved significantly. Our research complements signature-based detection by identifying common network-level behaviors of malware. These behaviors can be used to develop behavioral signatures of malware, which can be used to detect malware running on compromised hosts. In other words, signature-based detection can *prevent* malware infections, and *behavior signatures* can detect infections that bypass signature-based detection.

2.2 Dynamic Analysis on Host

Dynamic analysis builds signatures of a malware's interaction with its host. Such signatures may include memory access and file access patterns, as well as system call patterns. Those patterns that are likely to be present in malware but not in benign binaries can be used to develop a behavioral signature for malware detection [7].

Dynamic analysis complements static analysis and can overcome malware code obfuscation. Willems et al. [26] proposed CWSandbox that combines static and dynamic techniques for analyzing malware on a contained host. Guarnieri et al. [8]'s Cuckoo sandbox follows similar concepts as CWSandbox and additionally provides a network packet sink. Both works utilize virtual machines to isolate the study environment and the host. However, stealthy malware has another set of techniques to evade dynamic analysis – it detects debuggers and virtual machines, which are often used to speed up and facilitate dynamic analysis, and modifies its behavior. This leads to incorrect or unusable signatures of stealthy malware. Chen et al. [4] found that 39.9% and 2.7% of 6,222 malware samples exhibit anti-debugging and anti-virtualization behaviors respectively. In 2012, Branco et al. [2] analyzed 4 million samples and observed that 81.4% of them exhibited anti-virtualization behavior and 43.21% exhibited anti-debugging behavior.

Our work complements dynamic analysis, by providing another set of features, based on network behavior of malware, that can be used for detection. Our approach allows for feature collection using the network and does not require virtualization (binaries can be run on bare metal machines).

2.3 Dynamic Analysis of Network Behavior

There are a few efforts on analyzing the semantic of malware network behavior. Sandnet [18] provides a detailed, statistical analysis of malware network traffic, and surveys popular protocols employed by malware. Our work aims to understand popular network behavior patterns and the similarity in network behaviors between different malware samples.

Morales et al. [15] studied several network activities, selected using heuristics, which include: (1) NetBIOS name request, (2) failed network connections after DNS or rDNS queries, (3) ICMP-only activity with no replies or with error replies, (4) TCP activity followed by ICMP replies, etc. These activities can be used to detect the likely presence of malware. Morales et al. also report on the prevalence of those behaviors in contemporary malware. While their chosen activities are useful for malware detection, our high-level behavior analysis focuses on understanding malware purposes (e.g., spamming vs scanning). Our work thus complements the work by Morales et al.

Nari et al. [16] proposed an automated malware classification system also focusing on malware network behavior, which generates protocol flow dependency graph based on the IP address being contacted by malware. Our work improves this effort by systematizing detection of different malware behaviors using different network behavior patterns, and includes other information from network flows such as packet size, packet contents, etc.

Lever et al. [13] experimented with 26.8 million samples collected over five years and showed several findings including that dynamic analysis traces are susceptible to noise and should be carefully curated, Internet services are increasingly filled with potentially unwanted programs, as well as that network traffic provides the earliest indicator of infection. As a slight downside, the data

Service or Protocol	Label
DNS, HTTP, HTTPS	Not risky
FTP, SMTP, ICMP_ECHO	Risky, can be impersonated
Other services or protocols	Risky, cannot be impersonated

Table 1: Flow policies in Fantasm

used for this long period came from different sources which may not be collected in a well controlled environment and could result in noise that is hard to identify and remove. In contrast, our experiment is performed in a well-controlled experiment environment that makes it much easier to sanitize.

3 Capturing Malware Network Behavior

Contemporary malware relies more and more on the network to achieve its ultimate purpose [9, 22]. Malware often downloads binaries needed for its functionality from the Internet or connects into command and control channel to receive instructions on its next activity [10]. Advanced persistent threats [23] and keyloggers collect sensitive information on users' computers but need network access to transfer it to the attacker. DDoS attack tools, scanners, spam, and phishing malware require network access to send malicious traffic to their targets.

We study malware's network activities because they have become essential for modern malware. The first step in this study includes capturing malware's network traffic in an environment, which is transparent to malware, but also minimizes risk to Internet hosts from adversarial malware actions.

3.1 Analysis Environment

We leverage the experimentation platform, called Fantasm [6]. Fantasm is built on the DeterLab testbed [14], which is a public testbed for cyber-security research and education. DeterLab allows users to request several physical nodes, connect them into custom network topologies, and install custom OS and applications on them. Users are granted root access to the machines in their experiments. Fantasm runs on Deterlab with full Internet access, and carefully constrains this access to achieve productive malware analysis, and minimize risk to outside hosts. In our analysis, we run malware on a bare-metal Windows host, without any virtualization or debugger. We capture and analyze all network traffic between this machine and the outside using a separate Linux host, sitting in between the Windows host and the Internet. Both hosts are controlled by the Fantasm framework.

Fantasm makes decisions on which communications to *impersonate*, i.e., intercept and answer itself, which to *forward* and which to *drop*. This decision is made by taking into account each outgoing flow separately, making an initial decision, and revising it later if subsequent observations require this. Fantasm defines a flow as a unique combination of destination IP address, destination port, and protocol. Each flow is initially regarded as *non-essential*, and it is dropped.

If this leads to the abortion of malware activity, Fantasm stops analysis, restarts it, and regards that specific flow as *essential*. Fantasm then considers if it can fake replies to this outgoing connection in a way that would be indistinguishable from the actual replies, should the flow be allowed into the Internet. If Fantasm has an *impersonator* for the given destination and the given service, it will intercept the communication and fake the response. Otherwise, it will evaluate if the outgoing flow is risky, i.e., potentially harmful to other Internet hosts. If so, the flow will be dropped. Otherwise, it will be let out into the Internet. Table 1 illustrates the criteria used by Fantasm to determine if a flow is risky or not and if it can be impersonated. Note that even flows considered not risky and let out are still subjected to further monitoring and may be aborted if they misbehave because they could become part of scanning or DDoS. To minimize the risk of unwitting participation in attacks, Fantasm actively monitors for these activities and enforces limits on the number of *suspicious* flows that a sample can initiate. A suspicious flow receives no replies from the Internet. Many scans and DDoS flows will be classified as suspicious. If the analyzed malware sample exceeds its allowance of suspicious flows (10 in the current implementation), Fantasm aborts its analysis.

4 Sample Embedding

Once each sample is analyzed in Fantasm, we extract flow-level details of the malware's communication with the Internet from the captured traffic traces and create an *embedding* for each flow and each sample. Our selected *flow features* can be categorized into three broad categories:

- **Header information**. This information includes destination address, port, and transport protocol. We use this information to detect when different malware samples contact the same server, or same destination port (and thus may leverage the same service at the destination server).
- **Flow dynamics**. This includes a sequence of application-data-units (ADUs, see Section 4.1) exchanged in each direction of the flow, which corresponds to request and response sizes on the flow. We use this flow dynamics to detect malware flows with similar communication patterns.
- **Payload information**. We use a frequency-based technique (see Section 4.2) to generate an embedding of the flow's content, which can be used for a fast comparison between flows.

To create an embedding for an entire sample we use all its flow embeddings. A detailed list of features selected for each category is provided in Table 2. We next introduce two metrics we will use to closely compare malware samples.

4.1 Application Data Unit (ADU)

Flow dynamics include sequences of application data units with their sizes and direction. An *application data unit*, or *ADU*, is an aggregation of a flow by the

Feature Category	Feature Selected	Data Type
Header information	Source/Destination address	string
	Source/Destination port	number
	Protocol	string
Flow dynamics	Application data units	list
Payload information	Byte frequency table	dict

Table 2: Features selected for flow and sample embedding

direction that combines all adjacent packets transmitting in the same direction together, while maintains boundary of direction shift. Intuitively, ADU dynamics seeks to encode the length of requests and responses in a connection between a malware sample and a remote host. A transformation of packet trace to ADU is illustrated in Table 3. The ADU sequence is useful to detect similar flows across different malware samples based on their communication dynamics. For example, two different samples may download the same file from two different servers, and the contents may be encrypted with two different keys. However, the ADU sequence of these two flows should be very similar, enabling us to detect that these two flows have a similar or same purpose.

4.2 Payload Byte Frequency

Payload usually stores application-level data. Not all malware flows carry a payload, but if it is present it usually carries high-level logic, such as new instructions or binaries that are important for new functionality in malware. Hence it is important to study payload contents. On the other hand, payload information usually does not have a specific structure, as different malware may organize its data differently. We thus need a way to quickly summarize and compare payloads that may have very different formats.

We transform each flow's payload into a dictionary that encodes each byte's frequency. Keys to this dictionary are all possible byte values, 0–255, and the values being stored are the counts of how many times the given byte value was present in the payload. Finally, we divide each count with the total payload size to arrive at the frequency of byte values. This encoding has two advantages: First, it has a fixed and much smaller size than the actual payload. Second, it simplifies our similarity comparison between flows and samples.

4.3 Malware Similarity

Another useful application of behavior signatures is to study overlap in behaviors between different malware samples, e.g., to detect polymorphic or metamorphic malware, and to understand how malware functionality evolves, and how common it is across samples.

Obviously, our behavior signatures can yield a wide range of similarity measures, depending on how we define what "similar" means, and what weight we assign to features during the comparison.

Our calculation of sample similarity depends on two distance measures:

Pkt ID	Direction	Pkt size
1	incoming	50
2	incoming	60
3	outgoing	50
4	incoming	100
5	outgoing	70
6	outgoing	90
7	incoming	80
8	incoming	100

(a) Packet sequence

Ref. ID	Direction	Pkt size
(1+2)	incoming	110
(3)	outgoing	50
(4)	incoming	100
(5+6)	outgoing	160
(7+8)	incoming	180

(b) ADU sequence

Table 3: ADU transformation from packet sequence

1. **Inverse Jaccard Score for ADU comparison**: For a pair of flows, their ADU sequences are compared by calculating the Jaccard score, taking ADU sizes as the values in the set. The Jaccard score between two ADU sequences X and Y is defined in Equation 1.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{1}$$

   The flow similarity based on the original Jaccard score ranges from 0 to 1, with higher values denoting higher similarity. We convert this score into a *distance-type* metric "inverse Jaccard score" – $\overline{JS} = 1 - JS$, with lower values denoting lower distance and thus higher similarity.
2. **Kullback-Leibler divergence for payload comparison**: We compare payloads of flows P and Q by calculating Kullback-Leibler divergence (KL measure) between their payload frequencies, as described in Equation 2

$$D_{KL}(P||Q) = \sum_{b \in BY} P(b) log(\frac{P(b)}{Q(b)}) \tag{2}$$

   where BY is set of all possible byte values and P(b), Q(b) are frequencies of value b in payloads of the flows P and Q respectively. KL measure ranges from 0 to ∞, with lower values denoting higher similarity, i.e. it is already a distance-type metric.

When comparing two samples for similarity, we calculate their distance from flow distance measures in the following way. For each flow in sample A, we compare it with each flow in sample B, and take the lowest distance measure, i.e.

$$d(f_A) = d(f_A, B) = \min_{f_B} d(f_A, f_B) \tag{3}$$

where d is either $\overline{JS}$ or KL measure. We repeat this process for each flow in sample B. Finally, we calculate the **average distance**, which is the average over all flows, i.e.

$$d_{avg}(A,B) = \underset{f_X \in \{f_A, f_B\}}{avg} (d(f_X)) \tag{4}$$

5 Evaluation of Contemporary Malware

We now use features and patterns identified or defined in the previous sections to study malware sample similarity and prevalence of some selected high-level behaviors in contemporary malware.

5.1 Experiment Setup

We build our experiment environment using the Fantasm platform [6] as introduced in Section 3.1. Fantasm utilizes the Deterlab infrastructure to construct a LAN environment with a node running Windows to host the malware, and another node running Ubuntu Linux acting as the gateway. In addition, Fantasm provides necessary services for impersonators, and monitors the network activities by capturing all network packets using *tcpdump*. One round of analysis for a given malware sample consists of the following steps:

- Enable service network monitoring on Linux gateway
- Reload operating system on Windows node and set up necessary network configurations
- Deploy and start the malware binary and continue running it for a given period (we used 5 minutes)
- Kill the malware process and save the captured network trace.

This setting has the advantage that it is immune to current analysis evasion attempts by malware, because it does not use a debugger or a virtual machine. By reloading OS for each run, it ensures that each sample is analyzed in an environment free from any artifacts from the previous analysis rounds.

The malware samples we used in the study were provided by the Georgia Tech Apiary project [1]. We selected malware samples captured throughout 2018 for this research. Next, we submitted each sample to VirusTotal [25] to determine the type of malware, and ensure that the sample is recognized as malicious, from which we randomly picked 8,217 samples. We then analyzed each selected sample in our experiment environment, using the method described above. After the analysis, we have saved traces with all captured communications to and from the Windows node.

6 Clustering Flow Embeddings Using Machine Learning

To evaluate the effectiveness of using flow embeddings and payload byte frequencies on studying malware behavior, we cluster flows based on their metrics and see how well they perform on differentiating traffic behavior. We choose to use OPTICS algorithm (Ordering Points To Identify the Clustering Structure) from Scikit-Learn, which is an algorithm for finding density-based clusters in spatial data. The parameters for OPTICS we use are: $\epsilon = 2$, and min_samples=100.

We cluster malware flows based on two features from our embedding: *10-ADU sequence*, which is useful to identify common communication patterns, and *total*

payload size, payload size of printable characters, and payload size of all other characters. We choose to use these higher-level features instead of our payload frequency maps because they are not sensitive to small payload changes.

In total, we have analyzed 200,236 flows from the 6,595 malware samples out of the 8,172 samples we have selected. The remaining 1,577 samples do not successfully exchange payload with an external host. They send only DNS queries to the local resolver but do not initiate any further contact with the outside. In most cases, this happens because malware appears dormant during our analysis. We exclude these samples from further analysis.

6.1 Clustering by ADU Sequence

The clustering based on ADU results in 53 clusters and one group of unclustered flows, containing 8.71% of the flows. We exclude unclustered flows from further analysis. Based on the behavior, the flow clusters can be further grouped into one of the following larger categories:

- HTTP Downloading – incoming traffic volume is larger than outgoing and the application protocol is HTTP.
- HTTP Uploading – outgoing traffic volume is larger than incoming and the application protocol is HTTP.
- UDP Uploading – outgoing traffic volume is larger than incoming and the transport protocol is UDP.
- ICMP Scanning – various external hosts are contacted using ICMP.
- HTTPS Downloading – incoming traffic volume is larger than outgoing and the application protocol is HTTPS.
- HTTPS Uploading – outgoing traffic volume is larger than incoming and the application protocol is HTTPS.
- Unestablished - the flow attempted to connect to an external host but there was no reply or did not finish the 3-way handshake.

We summarize high-level findings based on these categories in Table 4. In this table, we gather information including several clusters that belong to a behavior category, the ratio of clustered flows for the behavior category, the ratio of samples for the behavior category that its flows belonging to, as well as the average $\overline{JS}$ distance for all flows within this behavior category.

The largest category is ICMP scanning, containing 5 clusters, 65.02% of total flows, and 2.04% of samples. The second-largest category is unestablished flows, containing 9 clusters, 23.92% of flows, and 67.57% of samples[3]. This suggests that most malware samples have many unsuccessful connections. These two types of flows cover 14 clusters, 88.94% of all flows, and span 69.64% of the samples. HTTPS downloading category consists of 7 clusters, 4% of the flows, and 2.07% of the samples; meanwhile, HTTPS uploading category contains 8 clusters, 2.90% of the flows, and 2.19% of the samples. HTTP downloading category contains 17 clusters, 2.86% of the flows, and 16.46% of the samples, while HTTP

[3] A sample can have multiple flows and thus can appear in multiple categories

Behavior	Cluster Count	Effective Flow ratio	Sample Span Span Ratio	Avg flow $\overline{JS}$ distance (lower is better)	Avg sample $\overline{JS}$ distance (lower is better)
HTTP Downloading	17	2.86%	16.46%	0.1542	0.1717
HTTP Uploading	2	0.21%	1.91%	0.1571	0.2017
UDP Uploading	4	1.17%	1.87%	0.0001	0.1901
ICMP scanning	5	65.02%	2.04%	0.0	0.2120
HTTPS Downloading	7	4.00%	2.07%	0.0749	0.1379
HTTPS Uploading	8	2.90%	2.19%	0.1621	0.1931
Unestablished	9	23.92%	67.57%	0.0034	0.3844

Table 4: High-level summary of clustering result using ADU.

uploading contains 5 clusters, 0.21% of the flows, and 1.91% of samples. This suggests that more malware samples utilize unencrypted HTTP traffic compared to encrypted HTTPS traffic, which allows for their payload analysis and possibly easier detection and filtering. The rarest observed behavior is UDP uploading, relating to 2 clusters, 0.53% of the flows, and 0.24% of the samples. As table 4 shows, the average $\overline{JS}$ distance of flows within categories is very low, which means that flows have similar dynamics (similar sizes of the first 10 ADUs). The ADU sequences of clusters could thus be used as behavioral signatures for malware detection.

6.2 Clustering by Payload Sizes

We next cluster flows by payload sizes, resulting in 37 clusters and one group of unclustered flows containing 14.16% of flows. Note that while we cluster by total, printable and non-printable character length of the payload, the average KL distance for flows in each category is very low (ranging from 0.0 to 0.16). This means that flows in the same category have very similar payloads, not just in length but also with regard to the actual byte values.

We again categorize the clusters using the same criteria as we used for ADU clusters. The categories and their coverage of clusters, flows and samples, as well as the average KL distance between flows in each category are shown in Table 5.

We see a similar trending as in ADU sequence-based clusters. ICMP is the dominating type of flows (69.29%) and is only shown in 0.12% of the samples with 0.0 average KL distance because there is no variation in the payload. This is followed by unestablished that contains 25.86% of the flows from 68.81% of the samples with average KL distance of 0.0034, which reinforces the results from ADU analysis that most samples contain failed connection attempts. We also find that more HTTP traffic is detected than HTTPS traffic for both downloading and uploading flows. Payload size also detects more HTTPS uploading flows from 1.92% of all effective flows from 5.26% of the samples, which covers more samples than its ADU equivalence. UDP is still the rarest type of flows detected here with 0.42% of the effective flows from 1.11% of the samples. As mentioned above, the average KL distance is calculated based on the payload frequency map and the result distances are still very low suggesting very high similarity

Behavior	Cluster Count	Effective Flow Ratio	Sample Span Ratio	Avg flow KL distance (lower is better)	Avg sample KL distance (lower is better)
HTTP Downloading	18	2.05%	9.78%	0.0194	2.557
HTTP Uploading	2	0.28%	2.90%	0.0738	0.1397
UDP Uploading	1	0.42%	1.11%	0.2289	2.480
HTTPS Downloading	1	0.09%	0.94%	0.0161	2.879
HTTPS Uploading	5	1.92%	5.26%	0.1283	0.3195
ICMP Scanning	4	69.29%	0.12%	0.0	0.1732
Unestablished	9	25.86%	68.81%	0.0034	2.149

Table 5: High-level summary of clustering result using payload sizes.

within each cluster. This proves the effectiveness of using the payload frequency map to detect payload patterns.

We now take a closer look at the payload of flows in dominant clusters. We skip unestablished and HTTPS encrypted flows as we cannot analyze their payload. We illustrate our findings through three examples.

From ICMP ping packets, we observe three dominant types of payload:

- "Babcdefghijklmnopqrstuvwabcdefghi".
- "\u0000" of 28 bytes.
- "b\u0004\u0000EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE" (with first character being either "b", or "o" or "\u0601").

This suggests that 150 malware samples in our dataset that perform ICMP scanning may reuse only three different code segments to generate scan packets. ICMP packet payloads we identified can be used as payload signatures for malware detection.

The unencrypted HTTP traffic also shows several dominant behaviors that are interesting. We show them in the Table 6. One type of traffic tries to query and access the root of `google.com` using "GET / HTTP/1.1". It receives "HTTP/1.1 301 Moved Permanently" response but ignores it and keeps sending the same request repeatedly. Such behavior is unusual for benign code and could be used as a behavioral signature to detect malware samples. Another interesting type of HTTP traffic is a type of GET request that tries to retrieve a potentially malicious payload from a malicious website but uses `google.com` as Referrer in the header. We suspect that this may be a way to circumvent the detection of some network defense systems by masquerading as an innocent redirection from Google query. Since a benign code could exhibit similar behavior we cannot use this pattern for malware detection, but it could be used to identify suspicious flows and send them to a more sophisticated defense for further scrutiny.

Another common HTTP flow behavior attempts to query many web sites using fixed-length and random-looking domain names, such as `qexylup.com` or `vowyzuk.com`, with the endpoints being `/login.php` or `/key.bin`. These patterns could be used as behavioral signatures for malware detection.

For UDP flows, we found a type of payload similar to what is used in Bittorrent tracker searching and can confirm that some flows even use the canonical

HTTP Behavior	Request Example	Flow ratio over HTTP Downloader
Query google.com	GET / HTTP/1.1 Host: google.com	35.83%
Access potential malicious payload on compromised website	GET /.../ddos.bss HTTP/1.1 Host: migsel.com (Migsel bike parts) Referrer: google.com	14.17%
Access potential malicious websie	GET /key.bin HTTP/1.1 Host: qexylup.com Referrer: google.com	39.76%

Table 6: Notable behaviors of HTTP Downloader

Bittorrent port 6881 for communication. The flows in this cluster have a low average KL distance of 0.2289, indicating a high payload similarity. Thus their payloads could be used to devise a payload signature for malware detection.

6.3 Sample Diversity

We analyze sample diversity based on our findings from flows. We found that 5,223 samples contain flows from different cluster groups from our previous results, which consists of 63.9% of all samples. We also listed the average sample $\overline{JS}$ distance in Table 4 and average sample KL distance in Table 5. We notice that for samples, both the average $\overline{JS}$ distance and average KL distance are higher than their flow counterparts, because those samples may contain flows of different types. We manually inspected some of those samples, and find some types of flows are more likely to be found in the same sample, such as HTTP GET requests for different URLs, HTTP uploading followed by HTTP downloading for potentially malicious payload, etc. These findings suggest that many samples exhibit multiple high-level behaviors and may be multi-purposed.

7 Discussion and Future Work

Our clustering results show that many flows are very similar and that we can use information about their ADU behavior and payload to devise behavioral and payload signatures for contemporary malware. Since the malware ecosystem changes rapidly, the signatures we devise today will likely be obsolete tomorrow. However, our methodology can be used with contemporary malware samples to identify future clusters of behaviors and payloads and to help defenses keep track of malware evolution.

Our current result is still bounded by time and computing power restrictions. Given more time and better infrastructures, we can foresee several future directions to continue our research.

Increase analyzed sample repository. We would like to examine more malware samples and expand our analysis, to balance out samples in each high-level behavior group. We would also like to perform a longitudinal study of malware evolution over time, to quantify how much dominant behaviors change.

Understand sample genealogy. Our results can currently help us identify samples that share similar or identical flows, suggesting that these samples may have a common author or that they may share code. We would like to extend our analysis to map out the evolution of malware samples, e.g., which sample came first, how did the specific behavior (e.g., contacting a C&C channel) change over time, etc.

Malware detection. Most high-level malware behaviors also occur in benign software, which makes them unreliable for malware detection purposes. However, combinations of these high-level behaviors may be unique to malware and could be useful for detection. For example, a software that scans other hosts and then copies data over is unlikely to be benign, although each of these actions separately could be undertaken by benign software (e.g., probing several servers could look like scanning if servers are unresponsive, and data can be uploaded to a cloud for legitimate reasons) As we extend our malware analysis to more samples, we expect to find more behavior combinations, which can be used for malware detection.

8 Conclusion

In this work, we propose to use malware's network traffic patterns to identify high-level behaviors of malware. We define "application data unit" to study malware traffic behavior and "payload frequency map" to study the payload behavior of malware. We then cluster flows from those malware samples based on these features and then form high-level behavior groups. The results show that each behavior group shows a significant behavior pattern. We confirm that flows from each cluster are very similar based on our distance metrics, suggesting that malware may be created by modifying existing code. We then look deeper into the actual behavior patterns within each cluster and find features that can be used to devise behavioral signatures for malware defense. We further analyze sample-to-flow mapping and confirm that 63.9% of the samples contain flows that belong to different behavior clusters, suggesting that contemporary malware is more likely to be multi-purposed.

References

1. Apiary. `http://apiary.gtri.gatech.edu/`. Accessed: 2018-05-09.
2. R. R. Branco, G. N. Barbosa, and P. D. Neto. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. *Black Hat*, 2012.
3. S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi. A survey on malware analysis and mitigation techniques. *Computer Science Review*, 32:1–23, 2019.
4. X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 177–186. IEEE, 2008.

5. M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. Technical report, WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES, 2006.
6. X. Deng, H. Shi, and J. Mirkovic. Understanding malware's network behaviors using fantasm. *Proceedings of LASER 2017 Learning from Authoritative Security Experiment Results*, page 1, 2017.
7. M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6, 2012.
8. C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser. The cuckoo sandbox. `https://cuckoosandbox.org/`, 2012.
9. T. Holz, M. Engelberth, and F. Freiling. Learning more about the underground economy: A case-study of keyloggers and dropzones. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2009.
10. K. T. D. Y. Huang, D. W. E. B. C. GrierD, T. J. Holt, C. Kruegel, D. McCoy, S. Savage, and G. Vigna. Framing dependencies introduced by underground commoditization. In *Workshop on the Economics of Information Security*, 2015.
11. Kaspersky. Kaspersky lab detects 360,000 new malicious files daily. `https://www.kaspersky.com/about/press-releases/2017_kaspersky-lab-detects-360000-new-malicious-files-daily`, 2017. Accessed: 2018-09-23.
12. J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith. Detecting malicious code by model checking. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 174–187. Springer, 2005.
13. C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis. A lustrum of malware network communication: Evolution and insights. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 788–804. IEEE, 2017.
14. J. Mirkovic and T. Benzel. Deterlab testbed for cybersecurity research and education. *Journal of Computing Sciences in Colleges*, 28(4):163–163, 2013.
15. J. A. Morales, A. Al-Bataineh, S. Xu, and R. Sandhu. Analyzing and exploiting network behaviors of malware. In *International Conference on Security and Privacy in Communication Systems*, pages 20–34. Springer, 2010.
16. S. Nari and A. A. Ghorbani. Automated malware classification based on network behavior. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 642–647. IEEE, 2013.
17. C. Raghuraman, S. Suresh, S. Shivshankar, and R. Chapaneri. Static and dynamic malware analysis using machine learning. In *First International Conference on Sustainable Technologies for Computational Intelligence*, pages 793–806. Springer, 2020.
18. C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. Van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.
19. A. Sharma and S. K. Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. *arXiv preprint arXiv:1406.7061*, 2014.
20. H. Shi, A. Alwabel, and J. Mirkovic. Cardinal pill testing of system virtual machines. In *USENIX Security Symposium*, pages 271–285, 2014.
21. H. Shi and J. Mirkovic. Hiding debuggers from malware with apate. In *Proceedings of the Symposium on Applied Computing*, pages 1703–1710. ACM, 2017.

22. B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. *LEET*, 11:4–4, 2011.
23. C. Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.
24. D. Ucci, L. Aniello, and R. Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.
25. VirusTotal. Virustotal-free online virus, malware and url scanner. `https://www.virustotal.com/en`, 2012.
26. C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2), 2007.