Polymorphic Malware Behavior Through Network Trace Analysis

Xiyue Deng
Information Sciences Institute
University of Southern California
Marina Del Ray, California, United States
xiyueden@isi.edu

Jelena Mirkovic
Information Sciences Institute
University of Southern California
Marina Del Ray, California, United States
mirkovic@isi.edu

***Abstract*—Malware continues to be a major threat to information security. To avoid being detected and analyzed, modern malware is continuously improving its stealthiness, including code obfuscation and encryption. On the other hand, a high number of unique malware samples detected daily suggests a likely high degree of code reuse under the layers of stealth. We observe that although obfuscation greatly changes a malware's binary, its functionalities remain intact.**

We propose to leverage malware's network behavior during its execution, to understand the malware's functionality and detect related or even same (polymorphic) malware. 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. We propose an encoding of malware samples that can help us classify samples, identify code reuse and genealogy, and develop behavioral signatures for malware defense based on malware's network behavior. We leverage the same encoding to identify polymorphic malware in a random dataset containing more than 8,000 diverse samples from the Georgia Tech Apiary project. We cluster 6,595 samples which show some network activity based on our embedding features and more than 90% of the cluster contains potentially polymorphic malware with up to 80% of the clusters identify truly polymorphic malware samples, i.e., they have identical network behavior as at least one other sample in our dataset. Such high level of polymorphism indicates a high level of code reuse, and shows how our approach can complement traditional code analysis techniques for malware defense.

***Index Terms*—malware, network, polymorphic, genealogy**

I. Introduction

The Internet is facing increasing threats due to the proliferation of malware. A study by Purple Sec suggests an estimated daily number of 230,000 new malware samples [18]. Such high malware production rate suggests that new malware may be created by transforming existing code to evade signature detection.

Malware designers invest large effort 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 [21]. 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 [22], [23], 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, downloading malicious code, 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.

In our previous papers, we have proposed the Fantasm environment for safe and effective live malware analysis [7], and we analyzed thousands of malware samples to identify partial code reuse [6]. In this work, we turn our focus on detecting polymorphic malware, i.e., samples that have identical network behaviors but different binary code. We analyze 8,172 malware samples randomly selected from the Georgia Tech Apiary project [1]. For each sample, we gather information from anti-virus suite results from VirusTotal [27] as well as our own embedding, which encodes network behavior information. We compare the local behavior reported by VirusTotal and remote network behavior collected by ourselves, and show that our embedding provides better information for identifying polymorphic malware. By clustering 6,595 samples, which show some network activity, using features defined through our embedding, we find that over 90% of clusters contain potentially polymorphic malware, with up to 80% of the

clusters identify truly polymorphic malware. This indicates the added benefit of network behavior encoding, over code analysis, for malware classification and malware defense.

In Section III we describe our environment for safe and effective malware experimentation, which we use to collect information about the malware's network behaviors. In Section IV we detail our embedding of network behavior into a feature vector for machine learning. In Section V we describe our method for detection of polymorphic malware using our embeddings. We show our findings in Section VI. We discuss future work in Section VII and conclude in Section VIII.

II. Background and Related Work

Malware analysis has received increased research interest over the years [3], [19], [26]. In this section, we discuss contemporary malware analysis methods and the rationale behind our approach.

A. Static Binary 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 [5], [12]. Such signatures can be used for malware detection, e.g., when binary code is downloaded over the network. 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 [21], such techniques have evolved significantly.

Researchers have also been improving signature detection methods, such as detecting decryption routines, performing runtime signature detection, etc. It is difficult for researchers to gain advantage in this race, as it will always be cheaper to generate new, obfuscated malware variants, than to analyze them.

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 that bypasses code-based defenses, and runs on compromised hosts. In other words, signature-based detection can *prevent* malware infections, and *behavior signatures* can detect infections that bypass signature-based detection.

B. Dynamic Binary Analysis

Dynamic binary analysis builds behavioral 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. The patterns that are prevalent in malware but not in benign binaries can be used to develop a behavioral signature for malware detection [8].

Dynamic analysis complements static analysis, and can overcome malware code obfuscation [9], [28]. 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 to hide its true purpose [2], [4].

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

C. Dynamic Analysis of Network Behavior

There are a few efforts on analyzing the semantic of malware network behavior. Sandnet [20] provides a detailed, statistical analysis of malware network traffic, and surveys popular protocols employed by malware. Morales et al. [16] 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. Nari et al. [17] 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. 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.

Our work complements the prior efforts by using a larger and more generic set of features to encode a malware's network behavior. In our prior work [6] we have shown how this encoding can be used to study a malware sample's genealogy and trends in malware code. In this work we focus on using the same encoding to detect polymorphic malware.

III. Capturing Malware Network Behavior

Contemporary malware relies more and more on the network to achieve its ultimate purpose [10], [24]. 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 [11]. Advanced persistent threats [25] and key-loggers 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 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 that is transparent to malware, and that also minimizes risk to Internet hosts from adversarial malware actions.

A. Analysis Environment

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 I: Flow policies in Fantasm

We leverage the experimentation platform, called Fantasm, described in [7]. Fantasm is built on the DeterLab testbed [15], 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 XP host, without any virtualization or debugger. We capture and analyze all network traffic between this machine and the outside, using a separate Linux host, which resides 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 I illustrates the criteria used by Fantasm to determine if a flow is risky or not, and if it can be impersonated. Fantasm monitors a given sample's communication with the Internet, and limits the number of *suspicious* flows – flows that receive no replies – that a sample can initiate. 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 the experiment and stops its analysis.

IV. Embedding the Samples

Once each sample is analyzed in Fantasm, we extract flow-level details from the captured traffic traces and create an *embedding* for each flow and each sample. We reuse the flow features as defined in [6].

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 IV-A) exchanged in each direction of the flow, which corresponds to request and response sizes on the flow. We use these features to detect malware flows with similar communication patterns.
- **Payload information**. We use a frequency-based technique [6] to encode the flow's payload into a compressed format, which can be used for fast comparison between flows. 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.

An embedding for an entire sample is the union of its flow embeddings.

A detailed list of features selected for each category is provided in Table II. We next introduce two metrics we use to closely compare malware samples.

A. 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 direction, which combines all adjacent packets sent in the same direction together, while maintaining the 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 III. 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. This will make their payload information different. 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.

B. 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

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

TABLE II: Features selected for flow and sample embedding

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.

V. Polymorphic Malware Detection

Polymorphic malware transforms its binary form, without changing its behavior. This enables malware to avoid being detected by most of the AV suites, because it evades static signature detection.

A typical malware changes its binary form through a polymorphic engine. A common approach used by a polymorphic engine is *packing*, which transforms a binary by encrypting or compressing it, and includes shell code to reverse this transformation in runtime.

In contrast there also exists a type of malware called metamorphic malware, which permanently transforms its binary code into another form while maintaining its behavior by replacing its assembly instructions with instructions that have equivalent functionality. From the attacker's point of view, achieving metamorphic transformation is harder than achieving polymorphism – the first requires transformation of the assembly code, which is non-trivial, while the second just requires encryption with a unique key. Therefore we focus on polymorphic malware in this work.

We identify potential polymorphic malware samples using clustering over parts of our encoding of network behaviors. In Section VI we experiment with ADU sequence feature and with byte frequency feature, and show that byte frequency works better in identifying polymorphic malware. We do not use header information for clustering as a sample could easily change its communication endpoint (e.g., C&C server) and port to evade signature-based network detection. In other cases, the change of communication endpoint occurs because malware contacts a cloud-based service and each sample may communicate with a set of different IP addresses.

Since the ADU sequence feature has variable length, depending on the number of flows and number of ADUs in each flow, we limit the number of flows to 50 and number of ADUs per flow to 10. For shorter flows and smaller samples, we pad their embedding with zeros. Finally, to be robust to flow or ADU reordering we concatenate embeddings for each flow in a sample, and sort the resulting embedding before clustering.

Similarly for byte frequency feature, we limit the number of flows per sample to 50, pad shorter flows with zeros and concatenate their embeddings, then sort to produce the final embedding for the sample.

We use OPTICS algorithm (Ordering Points To Identify the Clustering Structure) from Scikit-Learn for clustering, which is an algorithm for finding density-based clusters in spatial data. OPTICS uses Euclidean distance between vectors that are being clustered, and two parameters – *max_eps*, which denotes the maximum distance of a sample from its cluster and *min_samples*, which denotes the smallest allowed cluster size.

VI. Evaluation

We now use features and patterns identified or defined in the previous sections to study the prevalence of polymorphic malware in contemporary malware. It is difficult to evaluate accuracy of our polymorphic malware identification, because there is no public polymorphic malware dataset. Instead, we use a collection of randomly selected malware samples from the Georgia Tech Apiary project [1]. We use our approach to identify clusters of malware, which we believe are polymorphic and then we use observation of malware's local behavior to double-check quality of our findings. This process is detailed in Section VI-B.

A. Experiment Setup

We build our experiment environment using the Fantasm platform [7] as introduced in Section III-A. 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

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 III: ADU transformation from packet sequence

run, it ensures that each sample is analyzed in an environment free from any artifacts from the previous analysis rounds.

We selected malware samples captured throughout 2018 for this research. Next, we submitted each sample to VirusTotal [27] to determine the type of malware, and ensure that the sample is recognized as malicious. We randomly selected 8,217 samples for our evaluation. 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.

In total, we have analyzed 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. Since this low level of network communication is not sufficient to establish a malware sample's purpose, we exclude these samples from further analysis.

B. Cross-verifying Our Findings

To evaluate quality of our clustering and usefulness of our network behavior features for identification of polymorphic malware, we leverage malware's local behavior. We reuse the analysis results from VirusTotal to collect information about a sample's local behavior. Those reports include file system accesses such as files created, opened, deleted, etc., host files changed to manipulate DNS resolution, system mutex created or opened, processes malware spawned, Windows registry changed, runtime DLLs accessed, system services used, etc. To compare local behaviors we need to decide which parameter to use for comparison purpose. We find most of local information are potentially unstable because they can be modified easily by malware, for example, the names of accessed local file may be modified even those they were actually the same file being changed by different malware samples, therefore using those filenames as a metric for finding polymorphic malware samples will be unreliable. The same trick may also be applied to other types of information by malware. We then focus on those parameters that cannot be changed easily by malware, while keeping its functionality. As a result, we select the set of runtime DLLs accessed as a stable representation of local malware behavior. This feature is the feature of the host system, which malware leverages for its own purpose, and thus it is difficult for malware to manipulate this feature while preserving its functionality.

When we evaluate accuracy of a cluster that OPTICS produced using our network behavior embedding, we will count how many different local behaviors (different DLL sets) we observe in each cluster. We label those clusters that have a single DLL set as *truly polymorphic*. On the other hand if a cluster has more than one DLL set, but fewer than the number of samples in the cluster, we call this cluster *potentially polymorphic*. Finally, if each sample in the cluster has a different DLL set we say that this cluster is *not polymorphic*.

C. Calibrating Clustering Parameters

We first perform clustering experiments to identify best-performing values of *max_eps* and *min_samples*. Our accuracy measure is the percentage of clusters that are labeled as truly polymorphic, because all samples in those clusters access one set of local DLLs at runtime. The results are shown in Table IV. We observe that the rate of detecting truly polymorphic improves as we decrease *max_eps* and *min_samples* with diminishing improvements after reaching a certain threshold. When we use ADU sequences, best results are achieved for *max_eps*=10 and *min_samples*=2. These settings produce 408 clusters, out of which 74% are truly polymorphic and additional 22.5% are potentially polymorphic. These settings cluster 4,640 samples or 70% of all samples.

When we use byte frequency best results are achieved for *max_eps*=20 and *min_samples*=2. These settings produce 320 clusters, out of which 81.9% are truly polymorphic and additional 12.5% are potentially polymorphic, which achieves the highest truly polymorphic percentage across all settings that we have tested.. These settings cluster 4,343 samples or 65% of all samples. In the rest of the paper we use *max_eps*=20 and *min_samples*=2 and we use byte frequency to identify clusters of polymorphic malware samples.

D. Network vs local behavior

Since we use local behavior to evaluate quality of our polymorphic malware identification, it may seem that sets of local DLLs could be used, independently of network features to identify polymorphic malware. We explore this direction in this section.

We cluster all samples based on their DLL sets, grouping samples with identical sets into the same cluster. We then

Clustering Parameters		ADU Sequence				Byte Frequency			
max_eps	*min_samples*	Cluster Count	Truly Polymorphic	Potentially Polymorphic	Samples Clustered	Cluster Count	Truly Polymorphic	Potentially Polymorphic	Samples Clustered
500	50	27	3 (11.1%)	27 (100%)	3,220 (48.8%)	10	3 (30%)	10 (100%)	3,355 (50.8%)
200	20	55	10 (18.2%)	55 (100%)	3,553 (53.9%)	21	7 (33.3%)	21 (100%)	3,493 (53.0%)
100	10	93	22 (23.7%)	92 (98.9%)	3,907 (59.2%)	51	17 (33.3%)	50 (98.0%)	3,834 (58.1%)
50	5	164	55 (33.5%)	163 (99.3%)	4,192 (63.6%)	94	42 (44.4%)	91 (96.8%)	3,969 (60.2%)
20	2	420	310 (73.8%)	405 (96.4%)	4,693 (71.2%)	320	262 **(81.9%)**	302 (94.4%)	4,343 (65.9%)
10	2	408	302 (74.0%)	394 (96.5%)	4,640 (70.3%)	285	228 (80.0%)	269 (94.4%)	4,230 (64,2%)
5	2	393	288 (73.2%)	379 (96.4%)	4,595 (69.7%)	235	188 (80.0%)	219 (93.2%)	4,067 (61.7%)
2	2	373	273 (73.1%)	361 (96.7%)	4,537 (68.8%)	176	141 (80.1%)	166 (94.3%)	3,888 (59.0%)

TABLE IV: Clustering results using different parameter combinations.

Categorization	Total Cluster Count	Unclustered Sample Count	sub-categorization			
			With a single sub-pattern		With multiple sub-patterns	
			Pattern Count	Sample Count	Pattern Count	Sample Count
Network behavior (byte freq) with local behavior sub-pattern	320	2252	262	1,086 **(25%)**	58	3,257 (75%)
Local behavior with network behavior sub-pattern (byte freq)	1,322	0	798	1,449 (22.0%)	524	5,146 (78.0%)

TABLE V: Comparison of clustering by local and by network behavior

sub-cluster the samples in each cluster based on their network behavior, using *max_eps*=20 and *min_samples*=2 and clustering over byte frequency feature. The results are shown in the second row in Table V, and compared with our network-behavior based clustering, shown in the first row of the same table.

Clustering first on local behavior (DLL sets) leaves no unclustered samples, but only 22% of clustered samples and 60.4% of clusters exhibit same network behaviors. This reflects the fact that many DLL sets are not unique to a single malware sample or malware purpose, but rather used broadly by many samples for a variety of purposes. Table VI shows the top 10 reused DLLs, which are used in 64.2% to 91.7% of all DLL behavior group and 65.9% to 90.4% of all samples. The utility of some DLL files may be clear, e.g. mswsock.dll and dnsapi.dll are clearly used to initiate network activity, while some other DLL files are more general purpose, such as advapi32.dll, secur32.dll, comctl32.dll, etc. and hence don't provide a clear indication of the actual behavior. The high percentage of reuse of general purpose DLL files makes using accessed DLL files to study behavior pattern more challenging. Also DLL set access pattern doesn't provide a clear local access pattern, unlike our embedding which maps directly to human understandable network behavior.

On the other hand, clustering based on network behaviors leaves 34.1% of samples unclustered, but produces more coherent clusters, with 25% of clustered samples and 81.9% of clusters exhibiting same local behaviors (accessing same DLL sets).

E. Large Malware Clusters

We now take a closer look into largest clusters identified by our network-behavior based clustering, and shown in Table VII. For each cluster, we list the domain names queried through DNS, the network communication protocols, accessed remote IP addresses, as well as the sample count.

DLL file	Reuse In Clusters	Reuse In Samples
rpcrt4.dll	1,212 (91.7%)	5,962 (90.4%)
advapi32.dll	1,171 (88.6%)	5,552 (83.7%)
shell32.dll	1,055 (79.8%)	4,772 (72.3%)
mswsock.dll	946 (71.6%)	4,880 (74.0%)
secur32.dll	928 (70.2%)	4,347 (65.9%)
comctl32.dll	914 (69.1%)	4,067 (61.7%)
ole32.dll	895 (67.7%)	4,346 (65.9%)
rasadhlp.dll	875 (66.2%)	4,747 (72.%)
dnsapi.dll	873 (66.0%)	4,737 (71.8%)
wshtcpip.dll	850 (64.2%)	4,405 (66.8%)

TABLE VI: Top 10 most reused DLL files

We first pick the 3 largest truly polymorphic cluster and take a closer look at their behavior. The top cluster contains 115 samples, all of which try to access the domain "migsel.com" with the following GET request:

```
GET /system/classes/alive.php?
  key=Blackshades%5FKey&
  pcuser=<anonymized>&
  pcname=PC118&
  hwid=C405FD41&
  country=United+States
```

which looks like a keep-alive heart beat packet while providing information of the victim machine as part of a potential botnet. The authors tried to access migsel.com through web browser at the time of the writing of this paper with success. The domain name points to an online shopping web site and seems to function normally. On further inspection, all requests to the above URL got HTTP 404 as reply. One possible explanation is that the web site was compromised sometime ago with a botnet control server and had already been fixed so that all bots that were still trying to contact this server would fail like observed in our experiment.

The samples from second largest cluster initiate a proprietary TCP connection to www.baidu.com through port 80, which is unusual. The packet contents seems to be binary

Group	Domain	Comm. Methods	IP/Proto/Port Info	Sample Count
1	migsel.com	GET	95.128.128.129, TCP/80	115
2	www.baidu.com	TCP	104.193.88.77, TCP/80	95
3	N/A	ICMP	72.30.35.10, ICMP 98.137.246.8, ICMP 98.138.219.232, ICMP (etc.)	93

TABLE VII: Top 3 polymorphic malware groups categorized by ADU features.

Group	Domain	Comm. Methods	IP/Proto/Port Info	Sample Count	DLL patterns
1	Accessing zief.pl	N/A	148.81.111.121, TCP/65520	611	91
2	Accessing aa.org	N/A	157.122.62.205, TCP/1379	175	25
3	google.com	GET	172.217.4.174, TCP/80 (etc.)	21	7
4	google.com	GET	172.217.4.142, TCP/80 (etc.)	15	5

TABLE VIII: Prominent potentially polymorphic malware groups categorized by ADU features.

format, but some text are still recognizable and contains some system information like OS version, CPU frequency, etc. This behavior is similar to a backdoor malware, which reports victim system information to malware control center. Similar to the previous cluster, those requests were replied with "HTTP/1.1 400 Bad Request", which suggests there was an HTTP server running on port 80. This suggests that there might be a period of time that the baidu.com server was compromised to serve other purposes and had since been fixed. This cluster consists of 95 samples.

The third largest cluster, consisting of 93 samples, sent out huge amount of ICMP packets trying to detect availability of a given IP address. Such a sample may initiates as much as 15,000+ ICMP packets during our 5 minutes experiment duration. This suggests that ICMP is still the major way for detecting potential victim.

We then take a look at potentially polymorphic groups of interest. We first inspected the top potentially polymorphic groups and found 3 out of the top 5 clusters have their network communication related to "zief.pl" on an unusual TCP port 65520, which is a well-known malicious website and has since been taken down. The largest cluster consists of 611 samples, which also show 91 different DLL access patterns. Similarly, the second largest potentially polymorphic group try to access "aa.org" with another unusual TCP port 1379. As it turns out the malware was taken down from the website so all TCP SYN packets sent from the malware received no response.

The other clusters of interest we observed consists of samples that tried to contact google.com with only a simple "GET /" HTTP request, without trying to submit anything either. We believe such malware may try to use google.com for network availability check. Also note while all samples from group 3 and 4 access google.com, they are clustered differently not because they access different IP addresses, but their flow count differs. On the other hand, samples in the same cluster may access different google.com front end server due to differences in geolocation. Regardless, our embedding features can still cluster them into the same cluster, proving their robustness on identifying similar underlying network communication patterns.

Our further inspection through samples in the resulting clusters show that our embedding mechanism is effective on identifying common network behavior and at the meantime retain human readable information to ease further analysis of network behaviors.

Now we compare the network behavior our system captured among truly polymorphic clusters and potentially polymorphic clusters. We observe that for truly polymorphic malware, the network activity accesses lesser known domain, or performs simple tasks like sending ICMP packets. Such malware samples are either more single purposed or specialized to perform a set of specific tasks as defined by a malicious control center resides on a malicious or compromised web site. On the other hand, malware samples from potentially polymorphic cluster are more likely to access a public service or a well-known malicious service. As we detect different DLL set access patterns, we suppose that such samples may perform different local malicious activities while sharing the same network activity. This suggests malware component reuse, which combines different single purpose malicious modules to form new malware samples. As the statistics shown in Table V, more samples fall into clusters with more than one DLL set. This suggests that potential malware module reuse is very common in samples encountered in the wild. As shown previously, our network behavior embedding based clustering mechanism can identify or provide evidences of the existence of such polymorphic malware samples.

VII. Discussion and Future Work

Our clustering results show that many malware samples are very similar with regard to the ADU sequences and byte frequency of the flows contained. In addition to detection of polymorphic malware, these features could be used to form behavioral signatures for malware detection. 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.

Malware Evading Network Analysis. With the existence of techniques to evade signature-based malware detection, we can naturally assume that malware designers will also look to evade our network-communication-based analysis. There are several strategies that malware could use:

(a) appears to be dormant and await a specific trigger
(b) interleave malicious activities with benign traffic to avoid detection

We leave the handling of these evasion techniques for our future work, but sketch some possible solutions here. For trigger-based malware, we could leverage existing work on trigger detection, such as using static analysis and symbolic execution [5]. Since our methodology analyzes each flow separately, malware, which interleaves its malicious behavior with benign behavior will still generate flows that we can recognize as malicious.

Miramirkhani et al. [14] showed that malware authors could detect that a sample is being analyzed by detecting a lack of user-generated files or registry entries. Our current analysis environment would thus be easily detected. In the future we plan to address these challenges by providing artificial artifacts of human-user presence in our analysis environment, such as command line history, registry entries, files in user directories, recently opened file list in popular applications, etc.

VIII. Conclusion

In this work, we propose to use clustering over malware's network traffic patterns to identify polymorphic malware. We show that clustering over our application data unit and byte frequency in malware's network traffic produces a high percentage of clusters, containing samples whose flows are very similar to each other, and whose local behaviors are identical. We show that using local DLL access pattern to identify polymorphic malware samples has limited capability due to common DLL files are heavily reused, while our embedding based pattern clustering results in over 90% of potentially polymorphic clusters and up to 80% of truly polymorphic clusters and in the meantime provides human understandable network patterns to help understand the underlying behaviors of the malware.

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 and J. Mirkovic. Malware behavior through network trace analysis. In *International Networking Conference*, pages 3–18. Springer, 2020.

[7] 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.

[8] 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.

[9] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser. The cuckoo sandbox. https://cuckoosandbox.org/, 2012.

[10] 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.

[11] 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.

[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] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024. IEEE, 2017.

[15] J. Mirkovic and T. Benzel. Deterlab testbed for cybersecurity research and education. *Journal of Computing Sciences in Colleges*, 28(4):163–163, 2013.

[16] 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.

[17] 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.

[18] PurpleSec. 2021 cyber security statistics. https://purplesec.us/resources/cyber-security-statistics/, 2012.

[19] 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.

[20] 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.

[21] A. Sharma and S. K. Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. *arXiv preprint arXiv:1406.7061*, 2014.

[22] H. Shi, A. Alwabel, and J. Mirkovic. Cardinal pill testing of system virtual machines. In *USENIX Security Symposium*, pages 271–285, 2014.

[23] H. Shi and J. Mirkovic. Hiding debuggers from malware with apate. In *Proceedings of the Symposium on Applied Computing*, pages 1703–1710. ACM, 2017.

[24] 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.

[25] C. Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[26] D. Ucci, L. Aniello, and R. Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[27] VirusTotal. Virustotal-free online virus, malware and url scanner. https://www.virustotal.com/en, 2012.

[28] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2), 2007.