Understanding Malware's Network Behaviors using Fantasm

Xiyue Deng
xiyueden@isi.edu
Information Sciences Institute

Hao Shi
shihao@isi.edu
Information Sciences Institute

Jelena Mirkovic
mirkovic@isi.edu
Information Sciences Institute

Abstract

Background: There is very little data about how often contemporary malware communicates with the Internet and how essential this communication is for malware's functionality.

Aim: We aim to quantify what fraction of contemporary malware samples are environment-sensitive and will exhibit very few behaviors when analyzed under full containment. We then seek to understand the purpose of the malware's use of communication channel and if malware communication patterns could be used to understand its purpose.

Method. We analyze malware communication behavior by running contemporary malware samples on bare-metal machines in the DeterLab testbed, either in full containment or with some limited connectivity, and recording and analyzing all their network traffic. We carefully choose which communication to allow, and we monitor all connections that are let into the Internet. This way we can guarantee safety to Internet hosts, while exposing interesting malware behaviors that do not show under full containment.

Results. We find that 58% of samples exhibit some network activity within the first five minutes of running. We further find that 78% of these samples exhibit more network behaviors when ran under our limited containment, than when ran under full containment, which means that 78% of samples are environment-sensitive. Most common communication patterns involve DNS, ICMP ECHO and HTTP traffic toward mostly non-public destinations. Likely purpose of this traffic is botnet command and control. We further show that malware's network behaviors can be used to determine its purpose with 85–89% accuracy.

Conclusions. Ability to communicate with outside hosts seems to be essential to contemporary malware. This calls for better design of malware analysis environments, which enable safe and controlled communication to expose more interesting malware behaviors.

1 Introduction

Malware today evolves at an amazing pace. Kaspersky lab [1] reports that more than 300,000 new malware samples are found each day. While many have analyzed malware binaries to understand its purpose [7, 9], little has been done on analyzing and understanding malware communication patterns [17, 22]. Specifically, we do not know how much malware needs outside connectivity and what impact limited connectivity has on malware's functionality. We further do not understand which application and transport protocols are used by contemporary malware, and what is the purpose of this communication. Understanding these issues is necessary for two reasons. First, much malware analysis occurs in full containment due to legal and ethical reasons. If communication is essential to malware, then analyzing it in full containment makes what defenders observe very different from how malware behaves in the wild. Second, understanding malware communication patterns may be useful to understand its functionality, even when malware code is obfuscated or encrypted.

We hypothesize that communication may be essential to malware for multiple reasons. First, contemporary malware is becoming *environment-sensitive* and may test its environment before it reveals its functionality [7, 14]. If constrained environment is detected, malware may modify or abort its behavior. Second, much of malware functionality today relies on a functional network [13, 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 [25]. Without network access such malware is an empty shell, containing no useful code. Third, malware functionality itself may require network access. Advanced persistent threats [15] 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 net-

work access to send malicious traffic to their targets. Without connectivity, such malware will become dormant.

We test our hypothesis by analyzing 2,994 contemporary malware samples, chosen to represent a wide variety of functional behaviors (e.g., key loggers, ransomware, bots, etc.). We analyze each sample under full and under partial containment, for five minutes, and record all network traffic. Our partial containment is designed to carefully allow select malware communication attempts into the Internet, when we believe this is necessary to reveal more interesting behaviors. All traffic is monitored for signs of malicious intent (e.g., DDoS or scanning) and quickly aborted if these are detected. This way we can guarantee safety to the Internet from our experimentation.

We find that 58% of samples exhibit some network behavior, and that 78% of these samples exhibit more network behaviors when ran under our partial containment, than when ran under full containment, which means they are environment-sensitive. Most malware samples send DNS, ICMP ECHO and HTTP traffic, and contact obscure destinations rather than popular servers. Likely purpose of these malware communication attempts is command and control communication, and new binary download. We further show that malware's network behaviors can be used to determine its purpose with 85–89% accuracy. We also show that our partial containment is safe for the Internet. In twelve weeks of running, we have received no abuse complaints and our IP addresses have not been blacklisted.

All the code developed in our work and the materials used in our evaluation are available at our project website: `https://steel.isi.edu/Projects/fantasm/`

2 Related Work

In this section, we summarize related work on understanding malware behaviors.

Most malware analysis works focus on analyzing system traces and malware binaries [20, 21]. There are fewer efforts on analyzing the semantics of malware's network behavior. The Sandnet article [22] provides a detailed, statistical analysis of malware's network traffic. The authors give an overview of the popularity of each protocol that malware employs. However, they do not attempt to understand the high-level semantics of malware's network conversations, and this is the contribution we make. Our work also updates results from [22] with communication patterns of contemporary malware. For example, we observe that ICMP ECHO has become the second most popular protocol used by malware. Morales et al. [17] define seven network activities based on heuristics and analyze malware for prevalence of these behaviors. Yet this work does not provide insight into a malware sample's purpose (e.g., worm, scanner, etc.) and it may miss behaviors other than those seven select ones. Our work complements this work and covers a richer set of behaviors, composed out of some basic communication patterns discussed in Section 5.3.

3 Fantasm

In this section, we describe the goals for our Fantasm system, our partial containment rules and how we ensure safety to the Internet from our experimentation.

3.1 Goals

Our goal in designing the Fantasm system was to support safe and productive malware experimentation. *Safe* means that we wanted to ensure that we do no harm to other Internet hosts with our experiments. *Productive* means that we wanted to ensure that as many as possible outgoing communication requests, launched by malware, receive a reply to that malware may move on to its next activity.

3.2 Partial Containment

One could achieve safety in full containment, without letting any traffic out of the environment. But because malware is environment-sensitive this would not lead to productive experimentation. One could also experiment in an open environment, where all the traffic is let out. But this would not be safe since the analysis environment could become a source of harmful scans, DDoS attacks and worm infections, which harm other Internet hosts. Due to ethical consideration, no organization would support such analysis for long.

To meet our goals we decided to experiment with malware in *partial* containment, where we selectively decide which malware flows to allow to reach into the Internet based on our assessment of their potential risk to the Internet, which is conformant to the ethical principles for information and communication technology research [11]. We also attempt to handle each outgoing flow in full containment first, by impersonating remote servers and crafting generic replies. This further reduces the amount of traffic we must let out and improves experimentation safety. We now explain how we assessed this risk and how we enforced the containment rules.

Based on a malware flow's purpose we distinguish between the following flow categories: benign (e.g., well-formed requests to public servers at a low rate), e-mail (spam or phishing), scan, denial of service, exploit and C&C (command and control). Potential harm to Internet hosts depends on the flow's category. Spam, scans

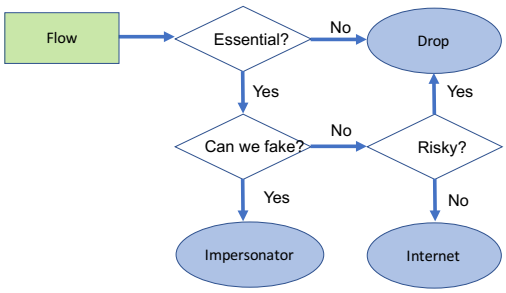

Figure 1: Flow handling: how we decide if an outgoing flow will be let out, redirected to our impersonators or dropped.

and denial of service are harmful only in large quantities – letting a few such packets out will usually not cause severe damages to their targets, but it may generate complaints from their administrators. On the other hand, binary and text-based exploits are destructive, even in a single flow. The C&C and benign communications are not harmful and usually must be let out to achieve productive malware experimentation.

The challenge of handling the outside communication with a fixed set of rules lies in the fact that the flow's purpose is usually not known a priori. For example a SYN packet to port 80 could be the start of a benign flow (e.g., a Web page download to check connectivity), a C&C flow (to report infection and receive commands for future activities), an exploit against a vulnerable Web server, a scan or a part of denial-of-service attack. We thus have to make a decision how to handle a flow based on incomplete information, and revise this decision when more information is available. Our initial decision depends on how essential we believe the flow is to the malware's continued operation, how easy it is for us to fabricate responses without letting the flow out of our analysis environment, and how risky it may be to let the flow out into the Internet. For essential flows whose replies are predictable, we develop generic services that provide these predictable responses and do not allow these flows into the Internet. We call these services "impersonators". Essential flows whose replies are not predictable, and which are not risky, are let out into the Internet, and closely observed lest they exhibit risky behavior in the future. Non-essential flows and essential but risky flows are dropped. Figure 1 illustrates our flow handling.

Traffic that we let out could be misused for scanning or DDoS if we let it out in any quantity. We actively monitor for these activities and enforce limits on the number of suspicious flows that a sample can initiate. We define a *suspicious* flow as a flow, which receives no replies from the Internet. For example, a TCP SYN to port 80 that does not receive a TCP SYN-ACK would be a part of a suspicious flow. Similarly a DNS query that receives no reply is a suspicious flow. Suspicious flows will be present if a sample participates in DDoS attacks or if it scans Internet hosts. If the sample exceeds its allowance of suspicious flows, we abort this sample's analysis.

We summarize our initial decisions and revision rules in Table 1. We consider DNS, HTTP and HTTPS flows as essential and non-risky, whose replies we cannot fake. We make this determination because many benign and C&C flows use these services to obtain additional malware executables, report data to the bot master and receive commands. Among our samples, DNS is used by 62%, HTTP by 35%, and HTTPS by 10% of samples (Section 4).

We consider FTP, SMTP and ICMP flows as essential flows with predictable replies. We forward these to our corresponding impersonators (Figure 1). These are machines in our analysis environment that run the given service, and are configured to provide generic replies to service requests. We redirect ICMP ECHO requests to our service impersonators and fake positive replies. We drop other ICMP traffic.

Our FTP service impersonator is a customized, permissive FTP service that positively authenticates when any user name and password are supplied. This setting can handle all potential connection requests from malware. If malware tries to download a file, we will create one with the same extension name, such as `.exe`, `.doc`, `.jpg`, and others. We save uploaded files for further analysis. For SMTP service, we set up an Email server that can reply with a "250 OK" message to any request. Our ICMP impersonator sends positive replies to any ICMP ECHO request.

4 Experimentation Goals Environment and Design

In this section, we discuss our experimentation goals, environment and experiment design.

4.1 Experimentation Goals

We wanted to observe and analyze communication patterns of malware. This necessitated identification of a relatively recent, representative set of malware binaries and running them in partial containment, while recording their communication. We further needed a way to quickly and automatically restore "clean state" of machines between malware samples

Goal	Action	Targeted Services
Elicit malware behavior	Forward	DNS, HTTP, HTTPS
	Redirect	FTP, SMTP, ICMP ECHO
Restrict forwarded flows	Drop	Other services
	Limit	Number of suspicious flows

Table 1: Flow policies for partial containment

4.2 Experimentation Environment

We experiment with malware samples in the DeterLab testbed [8]. DeterLab [8] enables remote remote experimentation and automated setup. An experimenter gains exclusive access and sudoer privileges to a set of physical machines and may connect them into custom topologies. The machines run an operating system and applications of a user's choice. Experimental traffic is usually fully contained, and does not affect other experiments on the testbed, nor can it get out into the Internet. In our experiments, we leverage a special functionality in the DeterLab testbed, called "risky experiment management", which allows exchange of some user-specified traffic between an experiment and the Internet. We specify that all DNS, HTTP and HTTPS traffic should be let out.

We run malware samples on several machines in a DeterLab experiment, which we will call Inmates. We hijack default route on Inmates and make all their traffic to the Internet pass through a special machine in our experiment, called Gateway. This Gateway implements our partial containment rules. We implement all of the service impersonators on a single physical machine. Each machine has a 3GHz Intel processor, 2GB of RAM, one 36Gb disk, and 5 Gigabit network interface cards.

To hide the fact that our machines reside within DeterLab from environment-sensitive malware we modify the system strings shown in Table 2. For example, we replace the default value ("Netbed User") of "Registered User" with a random name, e.g., – "Jack Linch". Therefore, malware will not detect the existence of DeterLab by searching for such strings.

4.3 Experiment Design

We run each malware sample under a given containment strategy (full or partial) for five minutes and record all network traffic at the Gateway. After analyzing each malware sample, we must restore Inmates to a clean state. We take advantage of the OS setup functionality provided by DeterLab to implement this function. We first perform certain OS optimization to reduce the size of OS image and thus shorten the time needed to load the image when restoring clean state. This modified OS is saved into a snapshot using the disk imaging function of DeterLab. This step takes a few minutes but is carried out only once for our experimentation. Later, whenever we need to restore the system after analyzing a malware sample, we reload the OS image using DeterLab's `os_load` command.

Our environment could also be used to study behavior of benign code, but this is outside of the scope of this research.

4.4 Malware Dataset

We obtained a recent set of malware samples by downloading 29,319 malware samples between March 4th and March 17th, 2017 from OpenMalware [2]. In order to obtain a balanced dataset we establish ground truth about the purposes of these samples by submitting their md5 hashes to VirusTotal [5]. We retrieve 28,495 valid reports. Each report contains the analysis results of about 50∼60 anti-virus (AV) products for a given sample. We keep the samples that were labeled as malicious by more than 50% AV products. This leaves us with 19,007 samples.

Concise Tagging. Each AV product tags a binary with vendor-specific label, for example, "worm.win32.allaple.e.", "trojan.waski.a", "malicious_confidence_100% (d)", or just "benign". As demonstrated in [6], AV vendors disagree not only on which tag to assign to a binary, but also how many unique tags exist. To overcome this limitation, we devise a translation service that translates vendor-specific tags into a nine concise, generic tags, such as: worm, trojan, virus, etc. We learn the translation rules by first taking a union of all the tags assigned by the AV products (74,443 in total), and then manually extracting common keywords out of them that signify a given concise category. Finally, we tag the sample with the concise category that is assigned by the majority of the AV products. Table 3 shows the breakdown of our samples over our concise tags.

We then randomly select 2,994 out of the 19,007 samples, trying to select equal number of samples from each category, to achieve diversity and form a representative malware set. We continue working with this malware set.

Key Name	Default in DeterLab	Our Modification
Registered User	"Netbed User"	Random name, e.g., "Jack Linch"
Computer Name	"pc.isi.deterlab.net"	Random name, e.g., "Jack's PC"
Workgroup	"EMULAB"	"WORKGROUP"

Table 2: Minimizing artifacts of DeterLab.

Table 3: Concise Tagging of Malware Samples

Categories	Samples	Categories	Samples
Virus	6,126/32%	Riskware	409/2%
Trojan	6,040/32%	Backdoor	197/1%
Worm	4,227/22%	Bot	45/<1%
Downloader	984/5%	Ransomware	17/<1%
Adware	962/5%	Total	19,007

Protocols	Samples	Protocols	Samples
DNS	1081/62%	1042	65/4%
ICMP echo	818/47%	799	33/2%
HTTP	600/35%	6892	25/1%
65520	237/14%	11110	17/1%
HTTPS	173/10%	11180	17/1%
SMTP	75/4%	FTP	12/1%

Table 4: Top 12 application protocols used by malware, and the number and percentage of samples that use them.

5 Results

In our evaluation, out of 2,994 malware samples in our malware set 1,737 samples exhibited some network activity during a run. The remaining samples may be dormant, waiting for some trigger or may simply exhibit too small communication frequency, which we cannot observe given our experiment duration (5 minutes).

5.1 Partial Containment Exposes More Malware Behavior

We measure the quantity of observable malware behavior by counting the number of network flows recorded during experimentation. Out of 1,737 samples that exhibit any network behavior, 1,354 (78%) generate more flows under partial containment than under full containment. This supports our hypothesis that network connectivity is essential for malware functionality, and that most malware samples are environment-sensitive. Out of 1,737 that exhibit network behavior there were 9,304,083 outgoing flows generated during our 5 minute experimentation interval. Out of these 9,304,083 flows, our impersonators could fake replies to 9,270,831 (99.64%) of them. We had to let 2,295 flows (0.02%) out into the internet because we could not fake their replies and they were deemed essential. Finally 30,957 flows (0.33%) were dropped because we did not have an impersonator for their protocol, but they were deemed too risky to be let out. We hope to develop more impersonators in the future, and thus further reduce risk to the Internet.

As a proof of how safe our experimentation was, during twelve weeks that we ran, we received no abuse complaints. We also analyzed 203 IP blacklists from 56 well-known maintainers (e.g., [3]), which contain 178 million IPs and 34,618 /16 prefixes for our experimentation period. Our external IP was not in any of the blacklists, which further supports our claim that no harmful traffic was let out.

5.2 Malware Communication Patterns

Table 4 shows the top 12 application protocols used by our malware dataset. DNS is used by 62% of samples and its primary use seemed to resolve the IPs of the domains that malware wishes to contact. ICMP was used by 47% of samples, likely to test reachability, either to detect if malware is running in a contained environment or to identify live hosts that may later be infected, if vulnerable. HTTP (35% of samples) and HTTPS (10% of samples) are likely used to retrieve new binaries, as we find many of these connections going out to file-hosting services. Port 65520 is mostly used by a virus that infects executable files and opens a back door on the compromised computers. The SMTP protocol is used to spread spam.

Samples in our malware dataset queried a total of 5,548 different domains, among which `zief.pl` (14%) and `google.com` (11%) are the most popular domains. We query these domains from `alexa.com`[1] , which has the records for 341 (6%) domains, as shown in Figure 2. We find that only 1% of the domains have ranks lower than 10,000, 5% have higher ranks and 94% of domains are not recorded by `alexa`. For the domains whose rank is lower than 10,000, most are web portals, such as YouTube and many are file storage services, like Dropbox. We manually check 20 domains that have no record in `alexa`, and none had a valid DNS record. This suggests that malware may use portal websites either test

[1] In our future work we will look to use a more robust representation of popular domains, like proposed by Metcalf et al in [16]

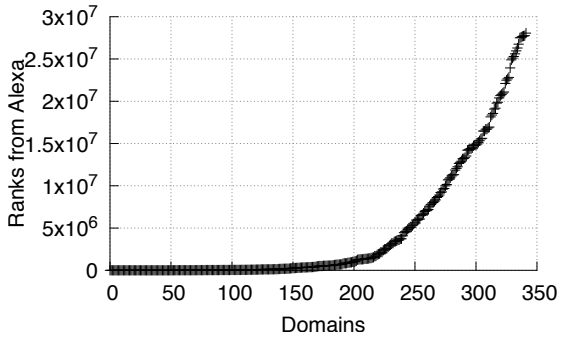

Figure 2: Ranks of domains from `alexa.com`.

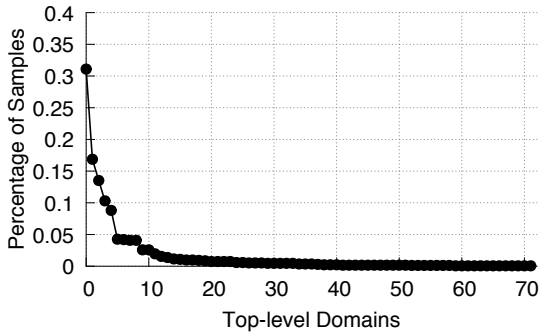

Figure 3: Popularity of top-level domains in our observed malware communications.

network reachability or for file transfer, and it may use private servers for file transfer or for C&C communication.

We classify the queried names based on their top-level domain, e.g., `.com` or `.net`. We find a total of 72 distinct top-level domains, as shown in Figure 3. The Top 3 of these domains are shown in Table 5. The `.com` is the most popular top-level domain, which is queried by 540 (31%) samples. The third column in Table 5 shows the top 3 queried domains in each top-level category. These domains contain 53 country codes, with Poland, Germany, and Netherlands being the top three countries. This means that malware in our dataset predominantly targeted European victims.

5.3 Summarizing Malware Communication

We now explore how to summarize malware communication so we can further investigate common patterns in how malware uses the Internet. Our goal was to create a concise and human-readable digest of malware's communication starting from recorded tcpdump logs. We call this representation *NetDigest*.

We start by splitting a malware's traffic into flows

Top-level	Samples	Second-level	Samples
.com	540/31%	google.com	187/35%
		msrl.com	73/14%
		ide.com	73/14%
.pl	293/17%	zief.pl	244/83%
		brenz.pl	26/9%
		ircgalaxy.pl	22/8%
.net	235/14%	secureserver.net	73/31%
		surf.net	68/29%
		aol.net	65/28%

Table 5: Popularity of domains in malware DNS queries.

Protocol	[Attribute: Value]
All	[LocalPort: `integer`]‡ [NumPktSent: `integer`]‡ [NumPktRecv: `integer`]‡ [PktSentTS: `float_list`]‡ [PktRecvTS: `float_list`]‡ [PayloadSize: `integer_list`]†
DNS	[Server: `IP_address`]‡[QueryType: `string`]‡ [CNAME: `string`]‡ [ResponseType: `IP_address list`]‡
HTTP/FTP	[Server: `IP_address`]‡[Proactive: `boolean`]‡ [GotResponse: `boolean`]‡[Code: `integer`]‡, [Download: `file_type`]†[Upload: `file_type`]†
SMTP	[Server: `IP_address`]‡[EmailTitle: `string`]* [Recipients: `string`]*[BodyLength: `integer`]* [ContainAttachment: `boolean`]* [AttachmentType: `string`]*
ICMP	[RequestIP: `IP_address`]*[NumRequests, `integer`]*

‡ Occur exactly once
† May have zero or more occurrences
* Have at least one occurrence

Table 6: NetDigest of a session.

based on the communicating IP address and port number pairs, and the transport protocol. We call each such flow a "session". Then, for each session, we extract the application protocol employed and devise a list of {attribute: value} pairs for this protocol, as shown in Table 6.

The first row of Table 6 shows the information that we will extract for all types of application protocols. For example, "LocalPort" denotes the local IP port used by malware, which is an integer. This attribute appears only once for a single session, and is derived from the definition of a session. The "NumPktSent" means the total number of packets sent by malware in an individual session. The "PktSentTS" is a list of Unix epoch time of all the packets sent by malware. Finally, we also maintain a list of each packet's payload size.

The DNS protocol has one attribute "Server", which has the value of `IP_address` that the query is sent to. For the domain queried by malware, the `QueryType` can be address record (`A`), mail exchange record (`MX`), pointer record (`PTR`), or others. For the response sent back by DNS server, we first save its canonical name, if any, in

a CNAME field. Then, we extract the response type and corresponding values and assign them to the ResponseType field.

For an HTTP or FTP session, we first take note of the server's IP address in the ServerIP field. Then, we use boolean values to denote if this session is initiated by malware ("Proactive") and if malware receives any response from Internet host ("GotResponse"). If the outside server replies to malware, we classify the following packets as "Download" or "Upload" based on the direction of the bulk volume of data. We also extract the file type being transferred.

For an SMTP message, we extract the server IP address, Email title, recipients, and body length. We also use a boolean value to note whether the message has an attachment and save the attachment's file type in a string.

For the ICMP protocol, we extract the destination IP address into the RequestIP field. We also save the number of requests in NumRequests field.

After we build the lists of attribute-value pairs for all the sessions produced by a malware sample, we sort the lists based on their first timestamps. The final, sorted list of session abstractions is called the *NetDigest*.

One sample NetDigest is shown in Figure 4 for the sample tagged as Trojan by AV products. At the beginning, this sample queries a domain (`ic-dc.deliverydlcenter.com`) using the default DNS server that is part of our impersonator set. Our DNS server acts as a recursive resolver and obtains and returns the actual mapping. Then, this sample downloads a picture and blob files from the first IP address returned. However, for the remaining Internet hosts, this sample just establishes connections with them but does not download or upload any information. For example, the second domain (`www.1-ads.com`) suggests that it is an advertising website, but no payload is downloaded from this website (session starting at timestamp `1488068896.977464`). In addition, some IPs are unreachable at the time of our execution, such as `52.85.83.112`.

5.4 Classifying Malware by Its Network Behavior

We now explore if unknown malware could be classified based on its communication patterns. Current malware classification relies on binary analysis. Yet, this approach has a few challenges. First, malware may use packing or encryption to obfuscate its code, thus defeating binary analysis. Second, malware may be environment-sensitive and may not exhibit interesting behavior and code if ran in a virtual machine or debugger, which are usually used for binary analysis. We thus explore malware classification based on its communication behavior, reasoning that malware may obfuscate its code but it must exhibit certain key behaviors to achieve its basic functionality. For example, a scanner must scan its targets and cannot significantly change this behavior without jeopardizing its functionality.

In our classification we divide our malware set into a training and a testing set. We then apply machine learning to learn associations on the training set between some features of malware communication, which we describe next, and our concise labels denoting malware purpose. Finally, we attempt to classify the malware in the testing set and report our success rate.

Extracting Features. We start with 83 select features, extracted out of the malware's NetDigest, as shown in Table 7.

We abstract malware's network traffic into four broad categories: Packet, Session, Protocol, and Content. For the Packet category, we divide it into three subgroups: Header, Payload, and Statistics. In the Header subgroup, we count the number of distinct IPs that a sample's packets have been sent to. In addition, we also look up the geographical locations of the IPs from the GeoLite [4] database, including the countries and continent they reside in. We chose these features because it is known that certain classes of malware target Internet hosts in different countries. In the Payload subgroup, we calculate the total size of payload in bytes. Furthermore, we compute the following statistics for both sent and received volume, the packet counts and the packet timing: minimum, maximum, mean, and standard deviation.

For the Session category, we consider all packets that are exchanged between malware and a single IP address. For these packets, we divide them into different *sessions* according to the local ports used by malware. For each session, we determine if its direction is proactive or passive, depending on whether the malware initiates the session or not. We say the Result of a session is successful if malware initiates the session and receives any responses from the host. We further calculate the number of TCP SYN packets, which can be used to detect SYN flood attacks. We also record the number of sessions per IP, which can be useful to further establish communication purpose. For example, in our evaluation, we find that one sample launches one short session with the first IP and then initiates multiple sessions with the second one for download. This network behavior indicates that the first IP serves as a master, directing the malware sample to the second, which acts as a file server.

For the Protocol category, we extract features for different types of application protocols. For example, for the DNS we summarize the number of distinct domains queried by malware in their DNS query and response packets. For HTTP, we count the number of packets carrying specific HTTP status codes, such as 200

```
1488068895.052901: DNS - [Server: 10.1.1.3], [A: ic-dc.deliverydlcenter.com],
                         [CNAME: N/A], [A: 52.85.83.81, 52.85.83.112,
                          52.85.83.132, 52.85.83.4, 52.85.83.96, 52.85.83.56,
                          52.85.83.32, 52.85.83.37]
1488068895.154335: HTTP - [Server: 52.85.83.81], [Proactive: True], [GotResponse: True],
                          [Download: blob], [Download: .png], [Download: blob]
1488068895.948346: HTTP - [Server: 52.85.83.81], [Proactive: True], [GotResponse: True]
1488068896.767094: DNS - [Server: 10.1.1.3], [A: www.1-1ads.com], [CNAME: n135adserv.com],
                         [A: 212.124.124.178]
1488068896.977464: HTTP - [Server: 212.124.124.178], [Proactive: True], [GotResponse: True]
1488069110.044756: DNS - [Server: 10.1.1.3], [A: ic-dc.deliverydlcenter.com], [CNAME: N/A],
                         [A: 52.85.83.56, 52.85.83.112, 52.85.83.96, 52.85.83.37,
                          52.85.83.81, 52.85.83.4, 52.85.83.132, 52.85.83.32]
1488069110.049507: DNS - [Server: 10.1.1.3], [A: ic-dc.deliverydlcenter.com], [CNAME: N/A],
                         [A: 52.85.83.32, 52.85.83.37, 52.85.83.56, 52.85.83.112,
                          52.85.83.96, 52.85.83.132, 52.85.83.4, 52.85.83.81]
1488069110.338822: HTTP - [Server: 52.85.83.81], [Proactive: True], [GotResponse: False]
1488069110.342816: HTTP - [Server: 52.85.83.81], [Proactive: True], [GotResponse: False]
1488069131.273458: HTTP - [Server: 52.85.83.112], [Proactive: True], [GotResponse: False]
1488069131.277206: HTTP - [Server: 52.85.83.112], [Proactive: True], [GotResponse: False]
1488069152.304031: HTTP - [Server: 52.85.83.132], [Proactive: True], [GotResponse: False]
1488069152.308025: HTTP - [Server: 52.85.83.132], [Proactive: True], [GotResponse: False]
1488069173.334854: DNS - [Server: 10.1.1.3], [A: ic-dc.deliverydlcenter.com], [CNAME: N/A],
                         [A: 52.85.83.32, 52.85.83.132, 52.85.83.96, 52.85.83.81,
                          52.85.83.4, 52.85.83.56, 52.85.83.112, 52.85.83.37]
1488069173.338605: DNS - [Server: 10.1.1.3], [A: ic-dc.deliverydlcenter.com], [CNAME: N/A],
                         [A:52.85.83.32, 52.85.83.132, 52.85.83.112, 52.85.83.56,
                          52.85.83.81, 52.85.83.4, 52.85.83.37, 52.85.83.96]
1488069173.381571: HTTP - [Server: 52.85.83.4], [Proactive: True], [GotResponse: False]
1488069173.383566: HTTP - [Server: 52.85.83.4], [Proactive: True], [GotResponse: False]
```

Figure 4: Example NetDigest (md5: `0155ddfa6feb24c018581084f4a499a8`).

Categories	**Subgroups**	**Features (83 in total)**
Packet	Header	Distinct number of: IPs, countries, continent, and local ports
	Payload	Total size in bytes; Sent/received: total number, minimum, maximum, mean, and standard variance
	Statistics	Sent/received packets: total number, rate; Sent/received time interval: min, max, mean, and standard variance
Session	Direction	Proactive (initiated by malware) or passive (initiated by Internet servers)
	Result	Succeeded or failed
	Statistics	Total number of SYNs sent; Number of sessions per IP: minimum, maximum, mean, and standard variance
Protocol	DNS	Number of distinct domains queried by malware
	HTTP	Number of replies received per reply code: 200, 201, 204, 301, 302, 304, 307, 400, 401, 403, 404, 405, 409, 500, 501, 503; Method: GET, POST, HEAD
	ICMP	Total number of packets; Number per IP: min, max, mean, and standard variance
	Other	Ports: total number of distinct ports, top three used
Content	Files	php, htm, exe, zip, gzip, ini, gif, jpg, png, js, swf, xls, xlsx, doc, docx, ppt, pptx, blob
	Host info	OS id, build number, system language, NICs
	Registry	Startup entries, hardware/software configuration, group policy
	Keyword	Number of: “mailto”, “ads”, “install”, “download”, “email”

Table 7: Features extracted from a malware’s NetDigest for classification purpose.

Algorithms	Rank 1	Rank 2	Rank 3
Decision Tree	242/89%	257/94%	259/95%
Support Vector	231/85%	259/95%	265/97%
Multi-layer Perception	231/85%	257/94%	262/96%

Table 8: Classification results: Rank 1 – our label was the top label assigned by AV products, Rank 2 – our top label was in the top 2 labels assigned by AV products, Rank 3 – our top label was among top 3 assigned by AV products.

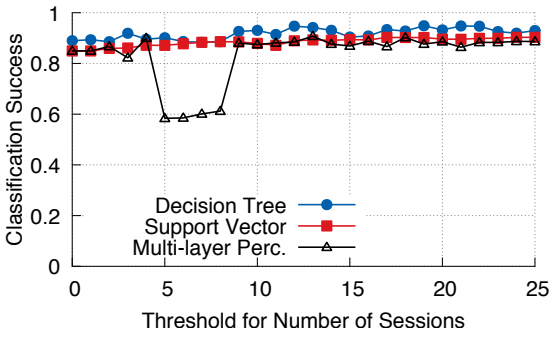

Figure 5: Classification precision as number of sessions grows.

(OK). Some malware samples behave differently based on the returned status code. For ICMP, we calculate minimum, maximum, average and standard deviation of packet counts. For non-standard IP ports, we maintain a set of distinct port numbers and calculate the top three ports targeted by each malware sample.

For the Content category, we investigate the payload content carried in HTTP packets, because this is the top application protocol used by malware in our experiments. We then use regular expressions to extract files from hyperlinks in HTTP content, and interpret their extensions. Sometimes the content is binary, and we tag it as blob. We also attempt to identify, using regular expressions, if payload contains host information and Windows registries that are typically reported to bot masters. Finally, we collect the frequencies of select keywords that are may indicate a malware purpose, such as "ads".

Classification Results. We investigate three popular classification methods in machine learning area – decision trees [10], support vector machines [12], and multi-layer perception [23]. We implement these algorithms and standard data pre-processing (data scaling and feature selection) through a Python package Scikit [18].

We use 80% of this data set for training and the remaining 20% of samples for testing. The results are shown in Table 8. Since malware today has very versatile functionality, it may be possible that a sample exhibits behavior that matches multiple labels. We denote as "Rank 1" the case when our chosen label matches the top one concise label chosen by the majority of AV products. When it matches one of top two labels, we denote this as "Rank 2" and if it matches one of top three labels, we denote it as "Rank 3". Our Rank 1 success rate ranged from 85 % (support vectors and multi-layer perception) to 89% (decision trees), which is very good performance. When we allow for a match between top two labels (Rank 2), our success rate climbs to 94–95%. And if we count match with any of the top three labels as a success (Rank 3), our rate climbs to 95–97%. Based on the typical performance of applying machine learning techniques in malware analysis [19], we conclude that our NetDigest representation can lead to very accurate malware classification, based only on observed communication patterns.

We further investigated the root causes of our misclassifications in Rank 1 that later became a success under Rank 2 or Rank 3 criteria. Toward this goal, we manually examined `pcap` traces of related samples. We find that all these samples exhibit limited network behavior that was not sufficient for classification. For example, one sample queries a domain and then establishes a connection with the HTTP server. However, no payload is downloaded or uploaded, and thus this behavior may match any malware category.

To investigate the relationship between classification accuracy and the number of sessions observed in malware communication we perform several iterations of the classification experiment. In each iteration filter out samples that launched fewer than N sessions. We then divide the remaining samples into training and testing set in 80%/20% ratio, train on the training set, perform the classification on the testing set, and report the success rate. We vary N from 1 to 25. The evaluation results are shown in Figure 5. The x-axis of Figure 5 denotes our limit on the number of sessions in a given run – N and the y-axis shows the classification success rate for each algorithm, corresponding to our Rank 1 criterion, on the testing set. Figure 6 shows the number of samples that generated N or fewer sessions in the training and the testing set together. Overall, all three of the classification methods performed well and were stable, except for multi-layer perception when session quantity is between 5 to 8. After investigating these sessions, we found that they do not have enough distinguishing feature values for multi-layer perception algorithm. The small variance of the input are further reduced by the intermediate calculation (hidden layers) of the algorithm [18]. The classification success rate increased slightly as the limit on number of sessions increased, from 88% at 1 session to 93% at 25 sessions. Thus longer observations increase

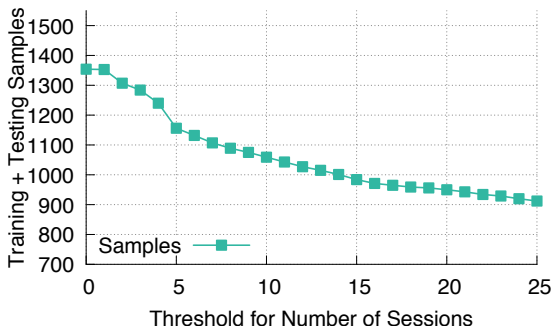

Figure 6: Number of samples as the limit on number of sessions grows.

classification accuracy but not by a lot.

6 Conclusions

In this work, we investigate how essential Internet connectivity is for malware functionality. We find that 58% of diverse malware samples initiate network connections within the first five minutes and that 78% of these samples will become dormant in full containment. We further provide breakdown of popular communication patterns and some evidence as to the purpose of these communications. Finally we show that malware communication behaviors ca be used for relatively accurate (85–89%) inference of a sample's purpose.

As future work, we will extend our framework to include analysis system-level activities for better understanding of a malware's purpose, and will seek to improve our generic impersonators to further reduce the cases when traffic must be let outside of the analysis environment.

7 Acknowledgments

This material is based upon work supported by the Department of Homeland Security under Contract No. HSHQDC-16-C-00024. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Homeland Security.

References

[1] Kaspersky Lab, 323,000 New Malware Samples Found Each Day. `http://www.darkreading.com/vulnerabilities---threats/kaspersky-lab-323000-new-malware-samples-found\\-each-day/d/d-id/1327655`, 2016.

[2] ISC Tech Georgia, Open Malware. `http://oc.gtisc.gatech.edu/`, 2017.

[3] Master Feeds, Bambenek Consulting Feeds. `http://osint.bambenekconsulting.com/feeds/`, 2017.

[4] MaxMind, GeoLite Legacy Downloadable Databases. `http://dev.maxmind.com/geoip/legacy/geolite/`, 2017.

[5] VirusTotal. `https://www.virustotal.com/en/`, 2017.

[6] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *RAID*, volume 4637, pages 178–197. Springer, 2007.

[7] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna. Efficient detection of split personalities in malware. In *NDSS*, 2010.

[8] T. Benzel. The Science of Cyber-Security Experimentation: The DETER Project. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.

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

[10] G. De'ath and K. E. Fabricius. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology*, 81(11):3178–3192, 2000.

[11] D. Dittrich and E. Kenneally. The menlo report: Ethical principles guiding information and communication technology research. *US Department of Homeland Security*, 2012.

[12] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. In *Advances in neural information processing systems*, pages 147–155, 1993.

[13] T. Holz, M. Engelberth, and F. Freiling. Learning more about the underground economy: A case-study of keyloggers and dropzones. *Computer Security–ESORICS*, pages 1–18, 2009.

[14] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[15] S.-T. Liu, Y.-M. Chen, and S.-J. Lin. A novel search engine to uncover potential victims for apt investigations. In *IFIP International Conference on Network and Parallel Computing*, pages 405–416, 2013.

[16] L. B. Metcalf, D. Ruef, and J. M. Spring. Open-source measurement of fast-flux networks while considering domain-name parking. In *Proceedings of the Learning from Authoritative Security Experiment Results Workshop*, 2017.

[17] 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, 2010.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[19] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[20] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125, 2008.

[21] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[22] 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.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[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] K. Thomas, D. Yuxing, H. David, W. Elie, B. C. Grier, T. J. Holt, C. Kruegel, D. McCoy, S. Savage, and G. Vigna. Framing dependencies introduced by underground commoditization. In *Proceedings (online) of the Workshop on Economics of Information Security*, 2015.