Understanding the Prevalence and Use of Alternative Plans in Malware with Network Games

Yacin Nadji
College of Computing
Georgia Institute of Technology
Atlanta, GA 30602, USA
yacin.nadji@cc.gatech.edu

Manos Antonakakis
Damballa Inc.
Atlanta, GA 30308, USA
manos@damballa.com

Roberto Perdisci
Department of Computer Science
University of Georgia
Athens, GA 30602, USA
perdisci@cs.uga.edu

Wenke Lee
College of Computing
Georgia Institute of Technology
Atlanta, GA 30602, USA
wenke@cc.gatech.edu

ABSTRACT

In this paper we describe and evaluate a technique to improve the amount of information gained from dynamic malware analysis systems. By playing *network games* during analysis, we explore the behavior of malware when it believes its network resources are malfunctioning. This forces the malware to reveal its *alternative plan* to the analysis system resulting in a more complete understanding of malware behavior. Network games are similar to multipath exploration techniques, but are resistant to conditional code obfuscation. Our experimental results show that network games discover highly useful network information from malware. Of the 161,000 domain names and over three million IP addresses coerced from malware during three weeks, over 95% *never* appeared on public blacklists. We show that this information is both likely to be malicious and can be used to improve existing domain name and IP address reputation systems, blacklists, and network-based malware clustering systems.

1. INTRODUCTION

Malware authors often implement a variety of techniques to improve the reliability of their malicious network infrastructure. For example, short-lived domain names are used by attackers to act as temporary drop sites for exfiltrated private information. Fast-flux service networks [30] let attackers rapidly change DNS resource records to improve the availability of their malicious network infrastructure. Malicious networks are becoming decentralized to eliminate a central point of failure. All of these techniques improve the reliability of malware by making them more resistant to take-down efforts. Furthermore, malware often makes use of randomness when choosing domain names or IP addresses to use to contact its command-and-control (C&C) server, which makes malware even more resilient and more difficult to analyze.

In particular, this poses a problem for dynamic malware analysis systems, which suffer from observing a single execution trace of a running program. For example, consider a malware family that randomly chooses a C&C domain name from a predefined list of 10 domains to connect to at runtime. MD5 distinct binaries of this malware family could have non-intersecting sets of domain names implying that they are unrelated samples. Furthermore, if a malware sample from this family successfully connected to the first domain name it picked and queried, we would fail to see the remaining nine domain names it could have used. These additional domain names could be instrumental in providing a malware sample a reliable way of contacting its C&C in the event some of its domain names had been remediated since it was initially created. Of existing malware samples seen in the wild, how many employ *alternative plans* in the form of additional domain names or IP addresses to contact in the event of network failure?

In this paper, we present a framework that addresses the primary weakness of dynamic malware analysis from a new point of view using the concept of *network games*. A network game tricks executing malware into revealing additional network information i.e., domain names and IP addresses. By providing fabricated network responses, malware is led to believe its C&C is unavailable forcing it to attempt to connect by whatever means necessary. Gaming malware from the network is complementary to existing approaches, such as multipath exploration [22, 38], but is resistant to evasion through conditional code obfuscation [34]. By using RFC-compliant network responses, games are *indistinguishable* from legitimate network responses from the malware's perspective on the host. Using our framework, GZA[1], we

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA

[1] Named after the Wu-Tang Clan member also known as "The Genius"

design a suite of games to understand the use and prevalence of alternative plans in malware in the wild by exploring malware behavior under a perceived unreliable network. Furthermore, we quantify the usefulness of the additional network information to security practitioners by examining public blacklists for the appearance of this information.

The gains generated from playing network games with malware directly benefits DNS and IP reputation systems [2, 4], network-based malware clustering systems [24], and traditional domain name and IP address blacklists. Furthermore, with these games we can obtain a better understanding of domain name generation algorithms [25, 39] (DGA) to aid in reverse-engineering them.

Specifically, our paper provides the following contributions:

- We design and implement a framework, GZA, for exploratory malware analysis using the concept of network games. Network games can be implemented quickly in as little as 100 lines of Python to respond to developing malware behavior and fit the specific needs of a malware analyst.

- Using GZA, we measure the prevalence of alternative plans in malware. We analyze a large dataset containing 2,191 distinct malware samples, and show that approximately 17% of the samples exhibit alternative plan behaviors.

- We study the long-term benefits of using network games to complement existing malware analysis techniques. Network games coerce, on average, an additional three domain names and two IP addresses from samples with alternative plans. Approximately 95% of the coerced information **never** appears on public blacklists throughout the course of our study. Over 76% of the coerced domain names were flagged as potentially malicious by the domain name reputation system Notos [2].

The remainder of this paper is structured as follows. In Section 2 we explicitly define network games and the specific games we used in our study. In Section 3 we describe the implementation of GZA. We present the methodology for two studies to examine alternative plan prevalence in Section 4, with the subsequent results shown in Section 5. We discuss the implications of network games as well as potential evasion techniques and their solutions in Section 6. We discuss similarities and differences from prior work in Section 7 and conclude the paper in Section 8.

2. PLAYING GAMES WITH MALWARE

Malware uses the same network protocols that benign software uses when performing malicious activity. Despite the fact that many network protocols exist, nearly all communication on the Internet follows one of two patterns:

1. A transport layer (e.g., TCP and UDP) connection is made to an IP address directly, **or**

2. A DNS query is made for a domain name (e.g., `google.com`) and a connection to the returned IP address is made as in #1.

Higher-level protocols leverage these two use cases for nearly all communication. If we can assume malware relies on these two patterns for contacting its C&C servers and performing its malicious activities, these are the patterns we must target during analysis.

We define a *network game* to be a set of rules that determine when to inject "false network information" into the communication between a running malware executable and the Internet. More specifically, false information is a forged network packet. Consider the running malware sample m in Figure 1(a). Sample m first performs a DNS query to determine the IP address of its C&C server located at `foo.com`. The returned IP address, `a.b.c.d`, is then used to connect to the C&C and the malware has successfully "phoned home". Sample m could also bypass DNS entirely if it were to hardcode the IP address of its C&C and communicate with it directly, as we see in Figure 1(c). This gives us two opportunities to play games with sample m as shown in Figures 1(b,d): we can say the domain name resolution of `foo.com` was unsuccessful (b) or the direct connection to IP address `a.b.c.d` was unsuccessful (d). At this point, sample m has four possible courses of action:

1. Retry the same domain name or IP address,

2. Remain dormant to evade dynamic analysis and try again later,

3. Give up, or

4. Try a previously unused domain name or IP address.

In (b) and (d), we see the malware samples taking action #4 and querying a *previously unseen* domain name (`bar.com`) and IP address (`e.f.g.c`), respectively. Action #2 is a common problem in dynamic malware analysis systems in general and is further discussed in Section 6.2.

2.1 Notation

Stated more formally, let h be a machine infected with a malware sample m that is currently executing in our analysis system running game G_{name}. G_{name} is a packet transformation function called *name*. Given a packet p, $G_{\text{name}}(p)$ represents G_{name} *gaming* p and its value is either the original packet, or some altered packet p' that changes the intent of p. The implementation details of G_{name} determine when to return p or p'. For example, p could contain the resolved IP address of a queried domain name d, whereas p' says d does not exist. In all other ways, such as type of packet and source and destination IP addresses, p and p' are identical. As h communicates with the outside world, it sends question packets, q_i, in the form of domain name queries and requests to initiate a TCP connection and receives response packets, r_j, in the form of domain name resolutions and initiated TCP connections[2]. False information is provided to the host h by delivering $G_{\text{name}}(r_j)$ in lieu of r_j. A *sample set* for m, $G_{\text{name,m}}$ represents the set of unique domain names and IP addresses queried by m while running under G_{name}. The functions D and I operate on sample sets and return the subset of unique domain names **or** the subset of unique IP addresses, respectively. Given a set of malware sample MD5s, M, a *game set*, G^M_{name} represents the subset of samples that were "successfully gamed" by G_{name}. A game is considered successful if it forces a malware sample to query

[2] More accurately, a TCP response packet is a TCP SYN-ACK packet as part of the TCP connection handshake.

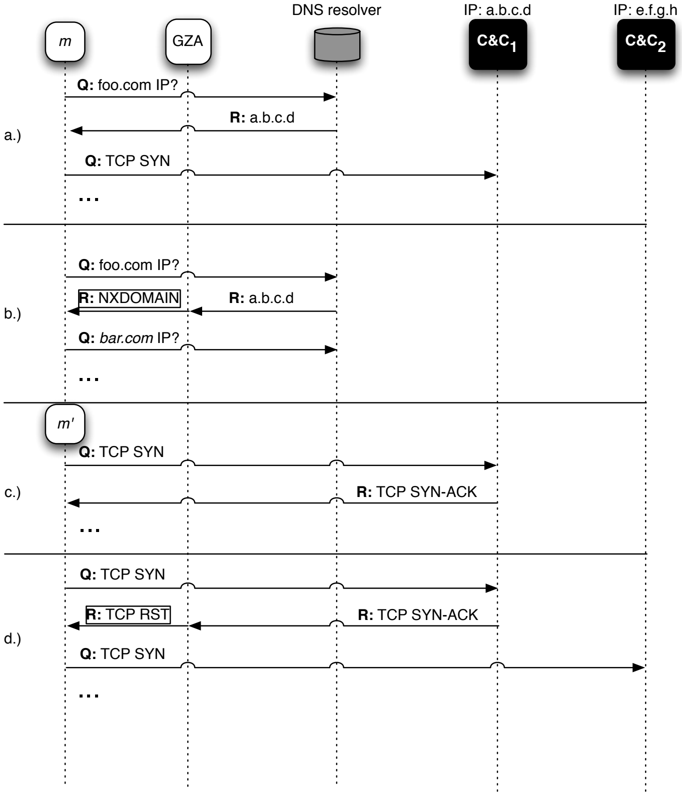


Figure 1: Malware samples m (a-b) and m' (c-d) initiating a connection with the C&C server. m connects by first performing a DNS query to determine the IP address of its C&C server followed by initiating a TCP connection. Sample m' connects directly to the C&C using a hard-coded IP address. Examples (a) and (c) connect without intervention by a game, while (b) and (d) have false information (denoted by boxes) injected.

more network information than under a run without a game present. We formally define this in the following section. The described notation is summarized in Table 1 and will be used throughout the remainder of the paper.

2.2 Designing Games for Interrogating Malware

Crafting games without a priori knowledge of malware network behavior is difficult. Furthermore, a successful game for sample m may be unsuccessful for sample m'. By using generic games "en masse", we improve our chances of successfully gaming malware during analysis. We design a suite of games to coerce a given malware sample into showing its alternative plan during analysis. We apply *all* games to a malware sample to improve the likelihood of success. Each game focuses either on DNS or TCP response packets in an attempt to harvest additional C&C domain names or IP addresses, respectively. For a DNS response packet p_d, p'_d is a modified response packet that declares the queried domain name does not exist, i.e., a DNS `rcode` of `NXDOMAIN`. For a TCP response packet p_t, p'_t is a modified response packet that terminates the 3-way TCP handshake, i.e., a TCP-RESET packet. In this paper, we choose to focus on DNS/TCP packets as they are the predominant protocols used to establish and sustain C&C communication; however, our approach is general and can be adapted to other protocols used less commonly in C&C communication. The design of an individual game is based on anecdotal evidence of how malware samples, in general, communicate. We design seven games to perform our analysis of alternative plan behavior in malware:

G_{name}	A game called *name*.
$G_{\text{name}}(p)$	The result of G_{name}'s transformation on packet p.
$G_{\text{name,m}}$	The set of network information i.e., unique IP addresses and domain names contacted, generated when malware sample m is gamed by G_{name}.
G^{M}_{name}	The subset of malware samples from M that were successfully gamed by G_{name}.
$D(s)$, $I(s)$	Given a sample set, s, return the subset of unique domain names or IP addresses in s, respectively.

Table 1: Notation for describing games and the sets they generate.

G_{null}.

To provide a baseline to compare the effectiveness of future games, this game allows response packets to reach its host without modification. In other words, G_{null} is the identity function.

Note that this does not mean malware communication is allowed to run completely unfettered. We perform standard precautionary measures to prevent malicious activity from harming external systems. However, these measures are not considered part of our network games, but simply good practice when analyzing potentially malicious binaries. We discuss these precautionary measures, which are always performed irrespective of the active game, in detail in Section 3.2.

G_{dns1}.

Malware often immediately connects to its C&C or performs some probing operation to determine the status of its network before doing so. This game assumes the first domain name lookup corresponds to a test of network availability and should be allowed to pass through without modification. Subsequent domain name lookups for domains other than what was queried first will be spoofed. For example, if `google.com` is the first domain queried, all subsequent domains that are not `google.com` will be spoofed. We approximate this behavior in the next game using a whitelist. For a DNS response packet p_d, $G_{\text{dns1}}(p_d)$ returns p_d if its the first DNS request packet and p'_d otherwise. G_{dns1} is successful for m iff $|G_{\text{dns1,m}}| > |D(G_{\text{null,m}})|$.

G_{dnsw}.

A popular domain name, like `google.com`, is unlikely to operate as a C&C server for a botnet. Therefore, DNS queries on popular domain names are unlikely to be concealing additional malicious network information. For a DNS response packet p_d, $G_{\text{dnsw}}(p_d)$ returns p_d if the domain being queried is whitelisted and p'_d otherwise. Our whitelist is comprised of the top 1000 Alexa domain names [1]. G_{dnsw} is successful for m iff $|G_{\text{dnsw,m}}| > |D(G_{\text{null,m}})|$.

G_{tcpw}.
An IP address that resides in a known benign network is also unlikely to function as a C&C, much like a popular domain name. For a TCP response packet p_t, $G_{\text{tcpw}}(p_t)$ returns p_t if the IP being queried is whitelisted and p'_t otherwise. Our whitelist is the dnswl IP-based whitelist [17]. G_{tcpw} is successful for m iff $|G_{\text{tcpw,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp1}.
Malware is often delivered by a *dropper*, a program that downloads, installs and runs the actual malicious binary. If we prevent the dropper from downloading its malicious payload, we will not observe the malicious behavior and fail to unearth alternative plans. We create a class of games that focus on droppers by allowing a variable number of TCP streams to successfully complete before forging response packets. For a TCP response packet p_t, $G_{\text{tcp1}}(p_t)$ returns p_t if the packet is the *first* TCP stream and p'_t otherwise. G_{tcp1} is successful for m iff $|G_{\text{tcp1,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp2}.
Droppers can have multiple *stages* where malicious payloads are downloaded in more than one TCP stream. This games *two stage* droppers. This game is the same as G_{tcp1}, but allows two TCP streams to complete. For a TCP response packet p_t, $G_{\text{tcp2}}(p_t)$ returns p_t if the packet is the *first or second* TCP stream and p'_t otherwise. G_{tcp2} is successful for m iff $|G_{\text{tcp2,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp3}.
While one and two stage droppers are fairly common in the wild, we wanted to test for three stage droppers. We can compare the results for G_{tcp1}, G_{tcp2} and G_{tcp3} to determine when we no longer benefit from increasing the number of allowable TCP streams. This game is the same as G_{tcp1}, but allows three TCP streams to complete. For a TCP response packet p_t, $G_{\text{tcp3}}(p_t)$ returns p_t if the packet is the *first, second or third* TCP stream and p'_t otherwise. G_{tcp3} is successful for m iff $|G_{\text{tcp3,m}}| > |I(G_{\text{null,m}})|$.

3. GZA

In this section, we describe the architecture of GZA and specific implementation details of our system.

3.1 Architecture

GZA contains two components: dynamic malware analysis and gameplay. The first component simply runs malicious code in a fresh virtual machine (VM) and records all network activity that occurs in the VM. All network activity for a VM is routed through one of the games described in Section 2.2 as seen in Figure 2. While the malware sample under analysis initiates communication with its C&C, all packets destined for a VM are routed through a network game. The game decides whether to faithfully route the response packet, or construct and send a spoofed packet, to the sample. Games are run on the host machine in isolation from the VMs, so malware cannot detect that its network activity is being analyzed and modified. As discussed earlier, all spoofed responses are RFC-compliant packets of the protocol currently being gamed making them indistinguishable from legitimate responses. As any analysis technique gains traction, malware authors will begin to attempt to circumvent it. Therefore, we discuss possible evasion techniques and how to mitigate them in Section 6.2.

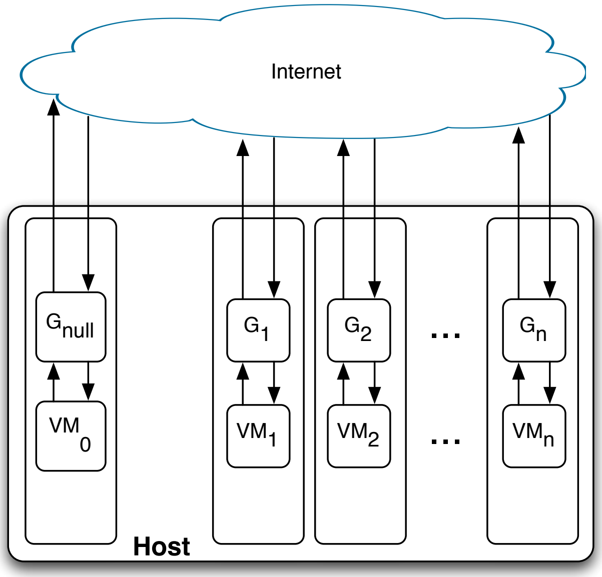


Figure 2: An overview of network traffic routing in GZA. Multiple virtual machines (VM_i) are run on a host using GZA. Each VM is paired up with a game G and a single sample is run against $n + 1$ games. This includes G_{null} to act as a baseline. Each VM's network is isolated from all others to prevent local infection. VM network traffic is routed through its paired game to perform the required packet transformations. There can be multiple groups of these on a single host to perform bulk sample analysis.

3.2 Implementation

GZA[3] is a collection of Python scripts that run malware samples inside a virtualized Windows XP instance in `kvm` and route packets to implement the games described in Section 2.2. All applications and services that could generate DNS or TCP traffic automatically are disabled to ensure that gamed packets are from the analyzed malware only. Before packets are routed for gameplay, precautionary measures are performed to prevent malware from damaging external systems. All SMTP traffic is redirected to a spam trap to prevent spamming and traffic to local systems is dropped to prevent local infection of nearby machines or concurrently running VMs. Each VM has its packets routed through the host running `kvm` using `iptables` [23] with relevant packets being forwarded to a Python script that runs a game. This is done through the `iptables` NFQUEUE interface that redirects each packet to a user-mode process which decides if the packet should be accepted or dropped. If the game returns the original packet, the `NF_ACCEPT` message is returned to the host's kernel and the packet is routed faithfully. If the game returns a spoofed packet, `NF_REJECT` is sent to the

[3] We will make the source code for the GZA framework available in the near future.

host's kernel and a forged packet is created and sent to the VM using the packet manipulation library scapy [5].

Games are very short Python scripts that provide two external functions: `playgame` and `spoof`. `playgame` instructs the host's kernel to route the original packet or to drop it; `spoof` generates and sends a falsified packet to the VM if the original packet was dropped. GZA allows for additional games to be created and removed as its operator sees fit. The implementation of all six of the DNS and TCP games took only 113 lines of code combined.

4. STUDY METHODOLOGY

Using the idea of playing games with malware, we design and run two studies to understand alternative plans in malware. The first goal of this study is to understand the prevalence of alternative plans in malware and determine which games are the best in general; successful games force executing malware to reveal the most additional information. The second attempts to quantify how useful this previously unknown information is by determining how long it takes for newly discovered domain names and IP addresses to appear on publicly available blacklists; coerced network information is more useful the longer it takes to appear. We assume that non-whitelisted network information contacted by malware is malicious. Note that not all games use the whitelist, but during our evaluation we ignore additional network information that is whitelisted. For example, if G_{dns1} caused additional benign domains to be queried, it would not be considered successful. For TCP-based games, we also ignore additional A records returned by DNS requests. We validate this assumption by providing DNS reputation scores for domain names. Furthermore, we show how this increase in network information can improve the accuracy of network-based clustering systems. In both studies presented, all samples are run for five minutes. We discuss timing based evasion further in Section 6.2, but in short the issue is common across all dynamic analysis systems and is orthogonal to the problem we address in this paper.

4.1 Representative Study

We created a dataset, D_R, of 2,191 distinct malware samples obtained between April 2010 and October 2010 from a variety of sources, including: low interaction honeypots, web crawlers, mail filters and user submissions. We used several sources to approximate the general malware population as closely as possible. Additionally, all samples in D_R were flagged as malicious by both Symantec and McAfee. For all malware samples $m \in D_R$, we run m in GZA against each game described in Section 2.2. The astute reader will notice that we run the risk of uncovering new information by chance. Consider a malware sample that is analyzed at two distinct times, t and t' where $t < t'$. It is possible that the malicious network infrastructure changed at some time v where $t < v < t'$, which could taint our results. To eliminate this possibility, a single sample is run against all games *at the same time.*

Malware tend to rely on either domain names *or* IP addresses to communicate with their C&C. Using this assumption, we can increase the throughput of GZA for the long-term study by only using the two most successful games for each protocol.

4.2 Long-term Study

In addition to measuring the prevalence of alternative plans in malware, we want to determine how useful this information is the day a malware sample appears on a malware feed. Detecting malicious domain names and IP addresses before they have appeared in blacklists offers a tangible improvement to companies and researchers that use domain name and IP address reputation systems, perform network-based malware clustering, or maintain domain name and IP address blacklists.

Using the two games chosen from the previous study and G_{null}, we play games with malware samples provided by our daily malware feeds over the course of three weeks. Each day, we analyze all the samples we encounter on our feeds using GZA. Samples that do not generate any network traffic while executing under G_{null} are removed from our results. For each sample that is successfully gamed, we must evaluate the usefulness of the newly obtained information. To do this, we cross-reference the domain name or IP address against eleven public blacklists [12, 20, 15, 16, 21, 35, 19, 13, 27, 29, 26]. The blacklists provide two dates: d_f, the first day a domain name or IP address appeared on the blacklist and d_l, the last day a domain name or IP address appeared on the blacklist. For each additional piece of network information, n_i, and the day it was coerced, t, we place it into one of four categories:

1. `blacklisted`: if n_i appears on any of the blacklists *after* we coerced it on day t i.e., $t < d_f$.
2. `decommissioned`: if n_i appears on any of the blacklists *before* we coerced it but has since been decommissioned i.e., $d_f < d_l < t$.
3. `campaigning`: if n_i appears on any of the blacklists and is *currently* being used i.e., $d_f < t < d_l$.
4. `never`: if n_i does not appear on any of the blacklists.

Each category provides interesting insight into a malware campaign. `blacklisted` network information shows our strategy can coerce domains that other parties eventually flag as malicious. `decommissioned` network information shows that while malware may stop using a network resource, they can quickly and easily resume using one. `campaigning` network information are seen in samples that connect to multiple network resources during *normal* operation. For example, a sample randomly chooses which domain name to use to contact its C&C. `never` network information is perhaps the most interesting. These are domains and IP addresses queried by malware that *never* appear on public blacklists throughout our experiments. `blacklisted` and `never` are the most useful categories of network information and provide the best improvements to systems that rely on such information.

By comparing generated malware sets, we will extract new relationships between malware originally thought to be unrelated. Consider two malware samples, m_1 and m_2 that when run using G_{null} they query domains d_1 and d_2, respectively. However, when run using G_{dnsw}, they *both* query d_1 and d_2. m_1 and m_2 are said to be *strongly related in* G_{dnsw}. Strongly related samples have *distinct* sets of network information when run in G_{null} but *identical* sets when run in any other game. For example, consider a malware family that randomly chooses a C&C domain name to connect to at runtime. MD5 distinct versions of this malware family could have distinct game sets in G_{null} but would have identical game sets in G_{dnsw}. Strongly related samples are likely to be related in some way, for example, they could be members of the same botnet. More formally, two malware samples, m_1 and m_2, are considered strongly related in G_{i} iff:

$$C(G_{\mathrm{i},\mathrm{m}_1}) = C(G_{\mathrm{i},\mathrm{m}_2}) \text{ and } C(G_{\mathrm{null},\mathrm{m}_1}) \neq C(G_{\mathrm{null},\mathrm{m}_2})$$

where C is a function that returns either the subset of unique domain names or unique IP addresses depending on the game type of G_{i} i.e., C is either D or I from Table 1. Samples could be related without being strongly related, however, we focus only on strongly related samples in this paper. Using network games, we can improve malware clustering that uses network features. Furthermore, we use the existing domain name reputation system, Notos [2], to validate that our newly discovered domain names are actually malicious.

5. ANALYSIS

5.1 Representative Study

A summary of the results from the representative study are available in Table 2. Of the 2,191 samples in our dataset, 17% were successfully gamed by at least one of the games described in Section 2.2. Of the two types of games, DNS-based and TCP-based, G_{dnsw} and G_{tcpw} were successful the most often with 6.0% and 7.5% success rate, respectively. In most cases, the increase in network information was between one and three new domain names or IP addresses for alternative plans. A plot of network information gains is shown in Figure 3. Both graphs are heavily skewed to the right which shows that if a malware author had the foresight to include an alternative plan they used few additional network resources. For increases in IP addresses in Figure 3(a), we see little difference between each individual strategy with respect to the amount of information increase. Figure 3(b) is similarly structured, but with a large spike at 14 additional domain names for G_{dnsw}. D_R contained 37 unique samples of the same malware that all queried the same set of domain names.

In addition to understanding the successes of each game individually, examining cases where multiple games were successful on an individual sample yield insight into understanding malware alternative plans. Table 3 shows this overlap by examining pairwise Jaccard Index [36] of the game sets of each game. The large overlap of 0.93 between $G^{D_R}_{\mathrm{dns1}}$ and the more successful $G^{D_R}_{\mathrm{dnsw}}$ show that a naive whitelisting strategy is sufficient and improves upon hard-coding for common patterns in malware. G_{dnsw} generalizes the behavior captured by G_{dns1}. TCP-based games exhibit a much smaller overlap, primarily due to the specific staged dropper the game targets i.e., G_{tcp2} targets two staged droppers. Game performance dropped from G_{tcpw} to G_{tcp1} and G_{tcp1} to G_{tcp2}. This shows that hardcoding for droppers is less effective than a whitelisting approach.

Furthermore, the small overlap of 0.20 between all DNS-based and TCP-based gamesets, $G^{D_R}_{\mathrm{dns}}$ and $G^{D_R}_{\mathrm{tcp}}$ shows that malware authors focus primarily on adding reliability using additional domain names or IP addresses for their C&C servers, but rarely both. Since we can approximate our DNS-based games with G_{dnsw} and G_{tcpw} is the best performer among TCP-based games, we will use these two games in our long-term analysis.

Game	% Gamed	Min Gain	Median Gain	Max Gain
G_{dns1}	4.4%	1	2	28
G_{dnsw}	6.0%	1	3	34
G_{tcpw}	7.5%	1	2	56
G_{tcp1}	6.3%	1	1	54
G_{tcp2}	5.4%	1	1	36
G_{tcp3}	5.4%	1	1	45
Total	17.3%	-	-	-

Table 2: Summary of results of the representative study of alternative plans in malware. The most successful DNS and TCP strategies are highlighted.

	$G^{D_R}_{\mathrm{dns1}}$	$G^{D_R}_{\mathrm{dnsw}}$	$G^{D_R}_{\mathrm{tcpw}}$	$G^{D_R}_{\mathrm{tcp1}}$	$G^{D_R}_{\mathrm{tcp2}}$	$G^{D_R}_{\mathrm{tcp3}}$	$G^{D_R}_{\mathrm{tcp}}$
$G^{D_R}_{\mathrm{dns1}}$	1	**0.93**	-	-	-	-	-
$G^{D_R}_{\mathrm{dnsw}}$	**0.93**	1	-	-	-	-	-
$G^{D_R}_{\mathrm{tcpw}}$	-	-	1	0.50	0.45	0.46	-
$G^{D_R}_{\mathrm{tcp1}}$	-	-	0.50	1	0.36	0.41	-
$G^{D_R}_{\mathrm{tcp2}}$	-	-	0.45	0.36	1	0.43	-
$G^{D_R}_{\mathrm{tcp3}}$	-	-	0.46	0.41	0.43	1	-
$G^{D_R}_{\mathrm{dns}}$	-	-	-	-	-	-	**0.20**

Table 3: Overlap in game strategies represented by the Jaccard Index. $G^{D_R}_{\mathrm{dns}}$ and $G^{D_R}_{\mathrm{tcp}}$ are the union of the DNS and TCP game sets, respectively.

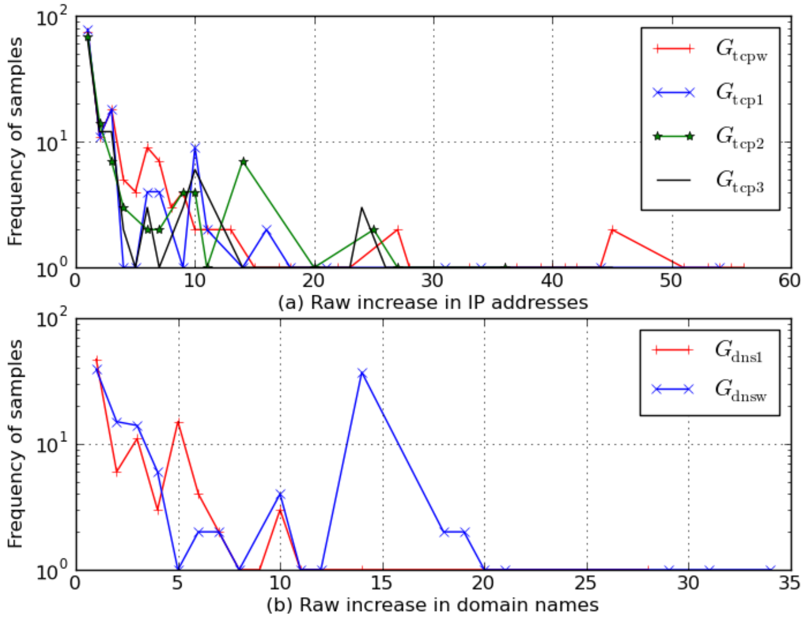
5.2 Long-term Study

We ran approximately 4,000 malware samples a day through GZA using three games $\{G_{\mathrm{null}}, G_{\mathrm{dnsw}}, G_{\mathrm{tcpw}}\}$ from March 11^{th} to March 31^{st}. In general, nearly all coerced network information was never blacklisted (category `never`) during the course of our study. See Table 4 for an example of the output for a single day of analysis that took place on March 15^{th}. Of the unique domains and IP addresses coerced, approximately 96% and 99% never appear on public blacklists by April 2nd, respectively. A small number were considered `blacklisted`, `decommissioned` and `campaigning`. A breakdown of these categories for the entire study are shown in Figure 4. As shown by the plot, almost all coerced network features never appear on public blacklists.

Network feature	Category	Count
Domains	Blacklisted	3
Domains	Decommissioned	15
Domains	In Campaign	10
Domains	Never Blacklisted	669
Domains	Total	697
IP	Blacklisted	1
IP	Decommissioned	6
IP	In Campaign	7
IP	Never Blacklisted	3381
IP	Total	3395

Table 4: Breakdown of coerced unique network information by category and protocol for March 15th.

The additional network information generated by our games makes relationships between malware samples clearer by providing a more complete picture of C&C communication. Consider a graph K where the vertices are malware samples for a given day of our long-term experiment and edges between vertices represent shared network information. For example, two malware samples that both connect to a do-



(a) Raw increase in IP addresses

(b) Raw increase in domain names

Figure 3: Plot of net network information gains for each game.

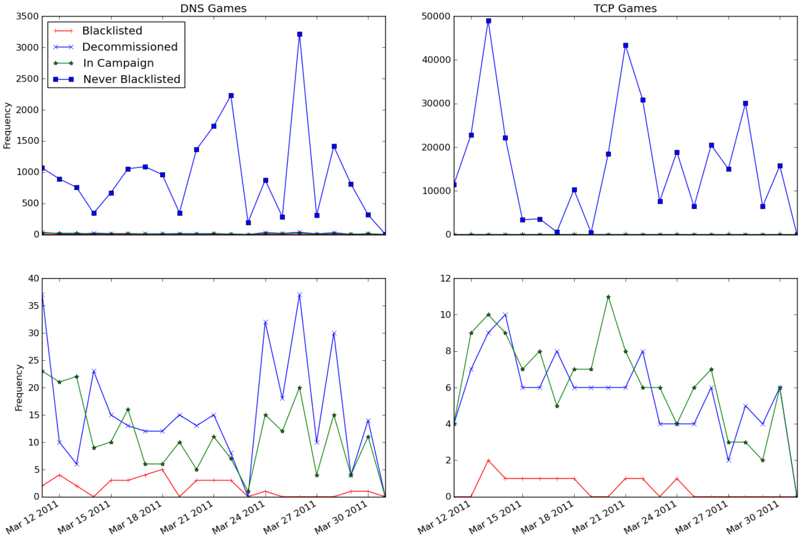


Figure 4: Frequency of network features by category over the course of the entire study. The top row of figures includes all categories, but due to the domination of the `never` category, we also include the other three categories alone on the bottom row. Please note the change in scale.

main name d would have an edge drawn between them in K. As we uncover more information through gameplay, we add additional edges into K. If these additional edges eventually form strongly related connections between malware samples, we should see a decrease in the number of components in graph K. Figure 5 shows that for $G_{\rm dnsw}$ we always see a drop in the number of components in the game graph of its network information compared to the graph under $G_{\rm null}$. $G_{\rm tcpw}$ exhibited no change in the graph from $G_{\rm null}$.

We also used Notos [2] to show the usefulness of our infor-

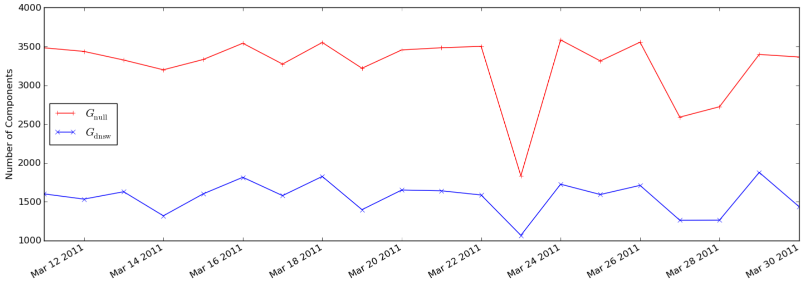


Figure 5: Numbers of components for G_{dnsw} and G_{null} for each day of the long-term experiment.

mation. Given a domain name, Notos classifies the domain as: suspicious, unknown, or whitelisted. Along with a classification, Notos also provides a confidence score. Notos was trained using four weeks of passive DNS data gathered from six ISP-based DNS recursive sensors located across North America. Notos uses the top 2,000 Alexa 2LD domain names and the same blacklists used in this study. Of the 161,000 unique domain names contacted during our long-term study, we ran a simple random sample of 15,050 of them through Notos and over 76% of them were flagged as suspicious (see Table 5). The whitelisted domain names were primarily: mail servers, dynamic DNS providers, and content distribution networks. Notos had high confidence in its classification of our coerced domain names: 80% of suspicious domains and 98% of whitelisted domains had confidences above 95%.

Classification	Count	Percentage
Suspicious	11,519	76.5%
Not Known	2	$< 0.01\%$
Whitelisted	3,529	23.4%
Classification	**Mean Confidence**	-
Suspicious	0.9731[4]	-
Whitelisted	0.9956[5]	-

Table 5: Domain name classification results and mean confidence values from Notos.

6. DISCUSSION

We discuss the implications of our findings as well as potential evasion techniques malware authors may implement to circumvent network games in general.

6.1 Malware Alternative Plans

Our results report that malware is constructed naively and often relies on a single domain name or IP address to initiate and maintain a connection to its C&C server. Malware that does provide an alternative plan infrequently uses its additional network information demonstrated by the vast majority of domain names and IP addresses that do not appear on public blacklists. This may explain why only 17% of samples responded to network games. Another explanation is more specific games, perhaps dynamic ones, must be implemented to realize gains in a larger proportion of samples. We are currently exploring these questions as potential future work.

In general, coerced domain names are more useful than coerced IP addresses. Domain names and abusing the DNS allow for more volatile malicious networks than an attacker could accomplish with IP addresses alone. Furthermore, it is easier to vet the maliciousness of domain names than IP addresses, making them more attractive to security practitioners.

6.2 Evasion

Attackers are always attempting to evade newly created defenses. The most obvious ways to evade our system are through timing attacks, peer-to-peer validation of network resource connectivity, communicating with different protocols, or by evading dynamic analysis entirely with excessive timeouts prior to performing malicious behavior. We discuss these evasion techniques and present methods to address these shortcomings. Since our games use RFC-compliant network responses, malware is unable to determine if it is being gamed at the host-level and subsequently must use the network in clever ways to determine its execution environment.

Dynamic malware analysis systems generally execute malware for a fixed period of time, usually around five minutes per sample. Malware can remain dormant until this time passes to evade detection and analysis. Prior work addresses this limitation by finding these *trigger-based behaviors* and generating inputs to satisfy the triggers at runtime. This limitation applies to all dynamic analysis systems in general and is orthogonal to the problem we are trying to solve of increasing the network information an executing sample attempts to connect to.

Overhead incurred during usermode packet generation could enable a clever malware author to determine if they are being gamed or not. As a performance improvement, GZA will only route packets relevant to the game in question. For example, when G_{dnsw} is being played, `iptables` will only route UDP packets with a port of 53 destined for a VM. If DNS packets take abnormally long, while packets of other types are unaffected this could alert a malware sample that it is being analyzed. Simply routing all packets through its game would apply this overhead uniformly across all packets, removing the signal.

Peer-to-peer (P2P) evasion is when a malware sample verifies the results of a DNS or TCP request by asking another infected machine to perform an identical request. If a sample, m, cannot resolve a domain name d, but fellow infected hosts can resolve d successfully, m has reason to believe it is being run under our system. Communicating this information, however, requires the network. This forces m to succumb to gameplay one way or another; gaming of its initial C&C communication or gaming of verification queries to its peers. By focusing on the building blocks of network communication, we force all network activity to be gamed.

To perform a DNS query, a malware sample could query an HTTP-based DNS tool[6], bypassing the DNS protocol entirely. Furthermore, it could directly connect to a C&C using a non-gamed protocol, such as UDP. These problems are easily addressed by running aggregate games and adding additional protocols. Querying an HTTP-based DNS lookup tool still requires *some* network activity so running DNS and TCP games *simultaneously* would prevent this lookup from succeeding. If an attacker uses another protocol, such as UDP, it is easy to write a new game that targets this new behavior. As malware adapts to the presence of network games, malware analysts can keep pace with malware authors without too much effort.

7. RELATED WORK

Deception through gameplay has been discussed [31, 11, 8, 37], or implemented by hand [10], but little empirical work has been done to demonstrate the usefulness such an approach provides. Prior work traditionally focuses on improving information gain generated by honeypots [37, 8] using game theory to model interactions between an attacker and a honeypot operator. Carroll et al. focus on gains generated by having a honeypot masquerade as a normal machine, or vice-versa, and show in which cases a Nash equilibrium can be reached. Wagener et al. similarly tried to achieve equilibrium, but also played games with live intruders. The honeypot was crafted to randomly fail process spawning system calls to coerce an attacker into attempting workarounds for failing tools, hopefully leading to previously unknown tools and exploits. Gaming the botnet C&C network redundancy mechanism, what we refer to as alternative plans, was discussed and used to improve returns generated by a spamming botnet analysis engine [18]. Anticipation games [7], an extension of attack graphs which are based on game theory, were designed to anticipate malicious interaction with a network and determine the answer to questions such as determining the most effective patching strategy for a given network. We differ from previous gameplay work in that we focus on gathering network intelligence, rather than host-level information, and we quantify the usefulness of this network information to security practitioners.

GZA is similar, but complementary, to other techniques that attempt to coerce malware into revealing useful information. All systems that rely on dynamic binary analysis run into the problem of code coverage, which researchers have addressed by forcing execution of all possible branches [22, 38]. Multipath exploration provides a complete view of possible execution paths of malware but can be evaded with conditional code obfuscation [34] or made impractical due to the exponential explosion in search space. Sharif et. al. describe malware emulators [33], or malware obfuscated by a randomized bytecode instruction set, that would evade multipath exploration. During dynamic analysis, multipath exploration would explore the paths of all possible bytecode programs rather than the execution paths of the malware itself. Since network games do not target binary execution paths, we are resistant to this evasion technique and provide a complementary analysis method. Furthermore, malware increasingly uses external stimuli in the form of *trigger-based behaviors* to determine execution. Malware can determine its execution environment [32, 28, 9] prompting the use of hardware virtualization [14]. More sophisticated techniques include waiting for a specific date to occur or a particular website to be visited. Research has shown how to detect changes in malware behavior as well as determine the underlying cause [3, 6]. We differ from prior work in malware analysis by introducing the concept of evasion-resistant network games. By performing execution path exploration from the network instead of the host, we make it difficult for malware to detect it is being gamed or evade our games.

8. CONCLUSION

In this paper, we designed and built a framework, GZA, to explore malware execution paths using the concept of network games. By playing network games with malware, we described the prevalence of *alternative plans* in malware by examining a large malware corpus of 2,191 samples and performing a long-term study over three weeks of malware samples obtained from malware feeds. Our six network games coerced samples into revealing their alternative plans and the additional network features malware used to enact those plans. We show that while alternative plans have promise to be used to improve malware reliability, they go relatively unused in malware seen in the wild. Only 17% show this behavior. This new network information, however, is *very* useful with approximately 95% never appearing on public blacklists. This directly improves systems that rely on network information, such as blacklist generation, domain name and IP address reputation systems, and malware clustering on network features.

9. REFERENCES

[1] Alexa. Top sites. `http://www.alexa.com/topsites`, (Retrieved) March 2011.

[2] M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee. Building a dynamic reputation system for DNS. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[3] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, and E. Kirda. Efficient detection of split personalities in malware. In *Proceedings of the Symposium on Network and Distributed System Security*, 2010.

[4] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proceedings of the Symposium on Network and Distributed System Security*, Jan 2011.

[5] P. Biondi. Scapy. `http://www.secdev.org/projects/scapy/`, (Retrieved) March 2011.

[6] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin. Automatically identifying trigger-based behavior in malware. *Botnet Detection*, pages 65–88, 2008.

[7] E. Bursztein and J. C. Mitchell. Using strategy objectives for network security analysis. *Information Security and Cryptology*, Jan 2011.

[8] T. Carroll and D. Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, Jan 2009.

[6] `http://www.kloth.net/services/nslookup.php`

[9] 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 *Proceedings of the International Conference on Dependable Systems and Networks DSN*, 2008.

[10] B. Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *Proceedings of the USENIX Security Symposium*, Jan 1990.

[11] F. Cohen and D. Koike. Misleading attackers with deception. In *Proceedings of the 2004 IEEE Workshop on Information Assurance*, Jan 2004.

[12] A. D. Correa. Malware patrol. `http://malwarepatrol.com/`, 2010.

[13] T. Cymru. Bogons. `http://www.cymru.com/Documents/bogon-bn-nonagg.txt`, 2010.

[14] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, Jan 2008.

[15] DNS-BH. Malware prevention through DNS redirection (black hole DNS sinkhole). `http://www.malwaredomains.com`, 2010.

[16] dnsbl.abuse.ch. dnsbl.abuse.ch. `http://dnsbl.abuse.ch`, 2010.

[17] dnswl. DNS whitelist - protect against false positives. `http://www.dnswl.org`, (Retrieved) March 2011.

[18] J. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 291–306, 2009.

[19] M. D. List. Malware domain list. `http://www.malwaredomainlist.com`, 2010.

[20] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. BLADE: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM Conference on Computer and Communiations Security (CCS 2010)*, Jan 2010.

[21] malc0de. Malc0de DNS blacklist. `http://malc0de.com`, 2010.

[22] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, volume 245, 2007.

[23] netfilter team. The netfilter.org "iptables" project. `http://www.netfilter.org/projects/iptables/index.html`, (Retrieved) March 2011.

[24] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[25] H. S. Phillip Porras and V. Yegneswaran. An analysis of conficker's logic and rendezvous points. `http://mtc.sri.com/Conficker/`, 2009.

[26] S. Project. Snort DNS/IP/URL lists. `http://labs.snort.org/iplists/`, 2011.

[27] T. S. Project. Spamhaus drop list. `http://www.spamhaus.org/drop/drop.lasso`, 2011.

[28] T. Raffetseder, C. Krügel, and E. Kirda. Detecting system emulators. In *Information Security Conference*, pages 1–18, 2007.

[29] C. Report. CIDR report bogons. `http://www.cidr-report.org`, 2011.

[30] J. Riden. How fast-flux service networks work. `http://www.honeynet.org/node/132`, 2008.

[31] N. Rowe, E. Custy, and B. T. Duong. Defending cyberspace with fake honeypots. *Journal of Computers*, Jan 2007.

[32] J. Rutkowska. Red pill... or how to detect VMM using (almost) one CPU instruction. `http://invisiblethings.org/papers/redpill.html`, 2004.

[33] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Rotalume: A tool for automatic reverse engineering of malware emulators.

[34] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Impeding malware analysis using conditional code obfuscation. In *Proceedings of the Symposium on Network and Distributed System Security*, Jan 2008.

[35] spyeyetracker.abuse.ch. Spyeye tracker. `https://spyeyetracker.abuse.ch`, 2010.

[36] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

[37] G. Wagener, R. State, A. Dulaunoy, and T. Engel. Self adaptive high interaction honeypots driven by game theory. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Jan 2009.

[38] J. Wilhelm and T. Chiueh. A forced sampled execution approach to kernel rootkit identification. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, Jan 2007.

[39] J. Wolf. Technical details of srizbi's domain generation algorithm. `http://blog.fireeye.com/research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html`, 2008.