# Crossing the Threshold: Detecting Network Malfeasance via Sequential Hypothesis Testing

Srinivas Krishnan, Teryl Taylor, Fabian Monrose
Department of Computer Science,
University of North Carolina at Chapel Hill
{krishnan, fabian, tptaylor}@cs.unc.edu

John McHugh
RedJack
john.mchugh@redjack.com

***Abstract*—The domain name system plays a vital role in the dependability and security of modern network. Unfortunately, it has also been widely misused for nefarious activities. Recently, attackers have turned their attention to the use of algorithmically generated domain names (AGDs) in an effort to circumvent network defenses. However, because such domain names are increasingly being used in benign applications, this transition has significant implications for techniques that classify AGDs based solely on the format of a domain name. To highlight the challenges they face, we examine contemporary approaches and demonstrate their limitations. We address these shortcomings by proposing an online form of sequential hypothesis testing that classifies clients based solely on the non-existent (NX) responses they elicit. Our evaluations on real-world data show that we outperform existing approaches, and for the vast majority of cases, we detect malware before they are able to successfully rendezvous with their command and control centers.**

## I. Introduction

Most administrators of enterprise networks would not be surprised by the discovery of yet another compromise on their networks. Indeed, attacks on computer networks are now an all too familiar event, and so operators are left with little choice but to deploy a myriad of network monitoring devices, and traffic engineering solutions, to ensure dependable and stable service of the networks they operate. However, as networks grow bigger and faster, staying ahead of this constant deluge of attack traffic is becoming increasing difficult. A case in point are the attacks on enterprise name servers that interact with the Domain Name System (DNS). These name servers are a critical cog, translating human readable domain names to IP addresses. As a result, any misuse of this service can have a significant impact on a network's operational health. While some of the attacks attempt to exploit flaws in the resolution process (e.g., cache poisoning attacks [17, 22]), others are more subtle and leverage an enterprise's DNS infrastructure to facilitate their nefarious activities. In this paper, we focus on the latter problem, highlighting a growing abuse of enterprise name servers whereby infected clients use automated domain-name generation algorithms to bypass defenses.

As the name suggests, domain-name generation algorithms are designed to generate names that refer to resources within the DNS namespace while minimizing potential collisions. Examples of malware that exhibit such behavior are botnets such as `conficker` and `kraken` and web-based malware and trojans such as `RunForestRun` [25]. `Conficker` is a sophisticated computer worm that propagates while forming a botnet. Since its discovery in 2008, it has remained surprisingly difficult to counter because of its combined use of advance malware techniques. To date, it has infected millions of computers worldwide. The early variants would reach out to 250 pseudo-randomly generated domain per day from eight Top Level Domains (TLDs) in an attempt to update itself with new code or instructions. In an unprecedented act of coordination, the cybersecurity community organized to block the infected computers from reaching the domains. Once the malware was reverse engineered, the defenders were able to leverage its domain generation algorithm to pre-register domains with the cooperation of the appropriate registries and authorities. The so-called Conficker Working Group sought to register and otherwise block domains before the `Conficker` operators, thereby preventing them from updating the botnet. Unfortunately, the `Conficker` operators responded to the defensive pre-registration practices by increasing the number of domains contacted by the infected computers—from 250 to 500 (of 50,000 possibilities) across 116 different TLDs.

Even more problematic for defenders, algorithmically generated domain names (AGD) are now also used for legitimate purposes. For instance, content distribution networks (CDNs) use such techniques to provide identifiers for short-lived objects within their networks, or to perform latency experiments [6]. Additionally, services like Spamhaus and Senderbase regularly use algorithmically generated domain names to query DNS blacklist information. Unfortunately, the security community has largely dismissed the prevalence of these legitimate uses of such domain names, and in doing so, overlooked their effect on the ability to detect malfeasance based solely on information gleaned from a domain name. Given that most methods to detect malicious algorithmically generated domain names leverage techniques that compare distributions of domain name features extracted from benign and malicious domains, algorithmically generated domain names used in benign applications can have a large impact on the accuracy of these techniques.

We explore techniques for identifying infected clients on an enterprise network and focus on their operational impact in terms of accuracy, timeliness of detection, and scalability

to large networks. First, we explore the efficacy of existing botnet detection techniques that rely solely on the structure of the domain name as a distinguishing feature in malware identification. More specifically, we implement techniques suggested in recently proposed detection mechanisms (e.g., [31, 32]) and evaluate these techniques on traces collected at a large campus network. We also examine the impact that the rise of benign applications (e.g., for performance testing in Web browsers and for location-based services prevalent in CDNs) has on these detection techniques. We show that the application of state-of-the-art detection techniques lead to high false positive rates, even when classifiers are enhanced with a combination of smoothing and whitelisting strategies. Moreover, successful classification only occurs after extended observation periods—which directly impacts the practical utility of these approaches.

To address these shortcomings, we propose an approach that exploits the fact that botnets tend to generate DNS queries that elicit non-existent (NX) responses. In particular, we leverage the fact that a noticeable side-effect of a bot's attempts to evade blacklisting is its tendency to have a wider dispersion of NX responses across DNS zones (compared to benign hosts). Our technique is based on sequential hypothesis testing (popularized by Jung et al. [15]) to classify internal client machines as benign or infected. In doing so, we address some key challenges, including the need to differentiate between benign and malicious DNS queries originating from the same client, and the ability to scale to high traffic loads. We show that our approach meets both of these challenges. Furthermore, one of the unique characteristics of our approach is that by focusing solely on NX traffic (and using novel filtering and domain collapsing techniques), we can achieve high accuracy on a fraction of the overall DNS traffic (e.g., 4%) which allows us to scale to larger networks. By contrast, existing approaches use all DNS traffic during analysis. Lastly, in an effort to reduce the cognitive load on a security analyst (performing a forensic analysis on the hosts flagged as suspicious), we provide an approach to cluster the output of our detector.

The rest of the paper is organized as follows. First, in §II, we explore the background of algorithmically generated domain names and discuss pertinent related work. §III covers our data collection infrastructure and summarizes the data used in our evaluation. In §IV we provide a detailed evaluation of existing techniques using domain name features and their shortcomings. We then introduce our technique in §V, followed by a detailed evaluation on archived data in §VI. To reduce the cognitive load on a security analysts, we also provide a technique for visualizing clients flagged by our technique. We provide operational insights on the deployment of our technique on our campus network in §VII, and conclude in §VIII.

## II. Background and Related Work

Unfortunately, the term algorithmically generated domain has been used in differing contexts in the existing literature. Antonakakis et al. [3], for example, describe an AGD as an "automatically generated pseudo-random domain name" created by a botnet using a domain generation algorithm (DGA), whereas other authors [4, 5, 24, 32] simply refer to the process of generating domains as "domain fluxing." In this paper, we consider an *algorithmically generated domain* as a domain that is generated by an automated process with the key objective of minimizing collisions within the DNS namespace. Consequently, algorithmically generated domains tend to be relatively long pseudo-random strings derived from some global seed. Google Chrome's domain generator, for example, creates three alpha-character strings (each of length ten) upon startup, and these strings are used to test whether the configured DNS server hijacks non-existent (NX) responses. If so, Chrome does not perform prefetching [13] of search terms that are entered into its location bar.

A recent method for identifying malicious traffic is to take advantage of historical information about the domain name being requested. As DNS-based reputation systems have been more widely deployed, attackers have turned to algorithmically generated domains (with short lifetimes) to circumvent these blacklists. As this cat and mouse game has continued, more timely blacklist and reputation-based systems have emerged (e.g., [1, 2, 4, 9]). Most of these proposals use features that are time-based, answer-based, or TTL-based to detect and model domains involved in malicious activity. Additionally, network-, zone-, and evidence-based features of DNS data are also used. For instance, both Antonakakis et al. [2] and Yadav et al. [32] take advantage of the fact that for high availability, botnets tend to map several domains to an IP address (or vice-versa). Defenders can therefore use the web of domains and IPs to uncover the underlying network infrastructure used by the botnet.

Other auxiliary information can also be used. Hao et al. [11], for example, use the fact that domains are registered "just in time" before an attack. More recent work [3, 14, 27, 31, 32] focuses on the fact that bots tend to generate lookups to hundreds or thousands of random domain names when locating their command and control server. Yadav and Reddy [31] rely on the burstiness of NX responses as well as the entropy of domain name character distributions to classify bot clients. By contrast, Antonakakis et al. [3] use a five-step clustering approach that clusters NX domains based on client-level structural information, and then incorporates network-level information to better classify AGDs. Jiang et al. [14] cluster failed DNS queries and attempt to identify subclusters with specific, presumably malicious, patterns.

Unlike the aforementioned works, we do not rely on domain structure or clustering techniques to identify bots. Rather, we focus entirely on the NX traffic patterns of individual hosts. As a result, our approach is lightweight, and can accurately identify bots upon seeing far fewer unique domain names than prior work. Furthermore, we utilize NX traffic exclusively, thereby enabling realtime analysis by using only a fraction of all DNS traffic observed.

The application of sequential hypothesis tests [28] in security context is by no means new. Jung et al. [15], for example, proposed a threshold random walk (TRW) algorithm to detect bots on a network. The insight behind their approach is that external scanners are more likely to contact inactive IP addresses than benign hosts, and so a sequential hypothesis test can be used to observe success and failure events in such an environment. Each success or

failure event moves a score towards one of two thresholds: one confirming the null hypothesis and another confirming the alternative hypothesis. After a number of events that are largely dictated by the TRW parameters, the score usually crosses a threshold, confirming one of the hypotheses. Similar ideas have been used to detect the propagation of worms [16, 20, 29], to identify opaque traffic [30], and to find node replication attacks in wireless networks [12].

## III. Collection Infrastructure

To aid in our pursuit of understanding AGD-based bot communication and develop an algorithm to detect bots, we collected and analyzed DNS traffic from several name servers at our campus for a week in March 2012. The monitored name servers served as the primary name servers for the entire wireless network as well as student residences and several academic departments around campus. The servers serve approximately 76,000 internal clients on a weekday and 50,000 clients on the weekends.

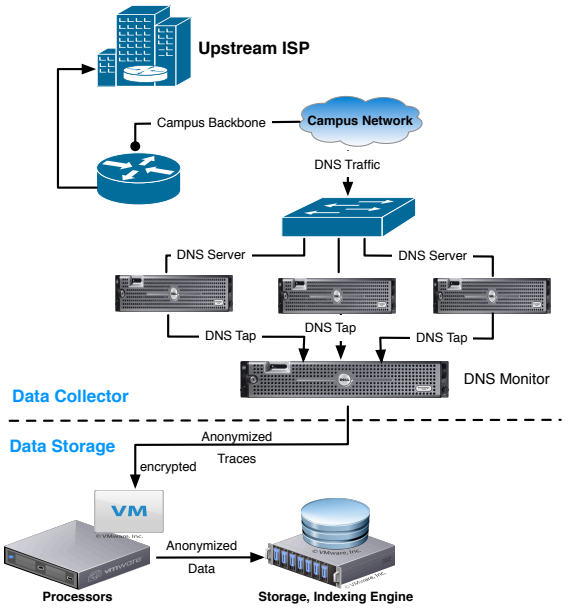


Fig. 1: DNS Monitoring Infrastructure

### A. Data Summary

Our collection infrastructure (see Figure 1) consists of a DNS trace collector and dissector. The DNS servers we monitored sits behind a load balancer, and all wireless clients using the campus network are assigned to one of these name servers during their DHCP registration. DHCP leases on this network are bound to the client's MAC address, and remain in effect for at least a few weeks. The DNS traffic from these servers is processed using a custom DNS engine. The packets in the trace are anonymized and encrypted while resident on disk.

We chose three consecutive days (March 18-20) for analysis. Table I summarizes some of the key statistics. The increase in traffic on March 19th corresponds to the start of the work week. Table I also shows that approximately 3% of all DNS queries result in non-existent or NX responses.

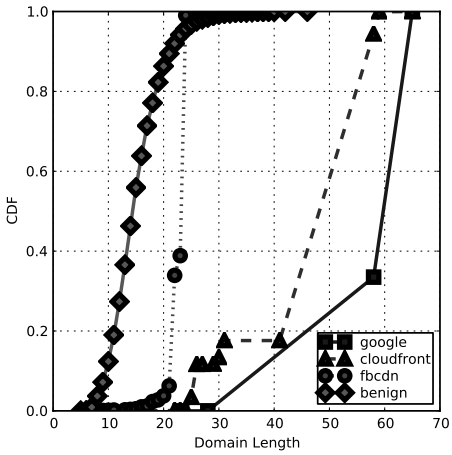


Fig. 2: CDF of domain name lengths for benign domains

A DNS server sends an NX response to a client when an entry for the domain queried by the client does not appear in the global DNS database. A mistyped domain name, for example, will lead to an NX response. Algorithmically generated domains comprise a surprisingly small amount of overall NX traffic, but they can have a large impact on the overall health of an enterprise network.

Apart from the DNS data we collected on campus, we compiled a list of 2,500 known botnet AGDs from publicly available blacklists. In particular, we targeted bots that are known to use DGAs for communication. Table II provides a summary of the bot families and their distribution within our blacklist. Besides the five well-known bot families represented in Table II, we also supplemented our blacklist with a set of newly discovered domains. The discovered domains were found by grouping DNS responses that originated from name servers that were used by the five well-known bot families. The domains in our list are used to study features used by existing techniques to detect DGAs, as well as compare the effectiveness of these techniques to ours.

TABLE I: DNS traffic stats for three days in March 2012.

| | March 18 | March 19 | March 20 |
|---|---|---|---|
| # of DNS Clients | 49.7K | 75.4K | 77.1K |
| # of DNS Queries | 37.3M | 61.2M | 60.3M |
| # of NX response | 1.3M | 1.8M | 1.7M |
| # of distinct domains | 1.5M | 1.8M | 1.8M |
| # of distinct zones | 373.4K | 528.2K | 566.4K |
| # of distinct NX domains | 190.4K | 216.2K | 220.4K |
| # of distinct NX zones | 15.3K | 22.1K | 24.2K |

## IV. Classification based on Features of a Domain Name

Existing techniques focus on properties of the name in order to identify and cluster algorithmically generated domain names. For instance, Antonakakis et al. [3] and Yadav et al. [32] used the length of a domain name as a feature to distinguish malicious domains from benign domains. Figures 2 and 3 show the distribution of the lengths of domain names for a set of benign and malicious domains.

TABLE II: Summary of bot samples used in our blacklist.

| Bot Family | # Samples | Sample of generated domain name |
|---|---|---|
| Bobax | 1079 | nghhezqyrfy.dynserv.com |
| Conficker | 728 | rxldjmqogsw.info |
| Cridex | 389 | frevyb-ikav.ru |
| Zeus | 300 | pzpuisexhqc69g33mzpwlyauirdqg43mvdt.biz |
| Flashback | 100 | fhnqskxxwloxl.info |
| *Discovered* | 314 | brmyxjyju.org |

The benign domains shown in Figure 2 include domains for known CDNs and other benign domains from alexa.com. Notice that domain names from alexa.com exhibit uniformly distributed lengths between 5 and 20 characters, while CDNs exhibits longer lengths clustered around a few discrete points. Similarly, the lengths of domain names used by botnet (in Figure 3) also cluster around a few discrete points; likely as a result of the generation processes they use. This similarity between the lengths of botnet domain names and benign CDN domain names suggests that the length of a domain name might not be a strong distinguishing feature.

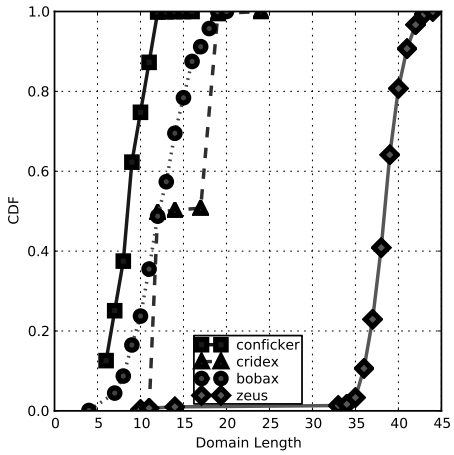


Fig. 3: CDF of lengths for botnet-related domains

Other proposals incorporate the use of similarity metrics for detecting malicious AGDs. In what follows, we revisit three similarity metrics used in current proposals, namely Kullback-Leibler (KL) divergence, Jaccard Index (JI), and Levenshtein distance. We discuss each in turn.

*KL Divergence:* One approach for detecting algorithmically generated domain names is to use the Kullback-Liebler (KL) divergence to compare character frequency distributions. Kullback-Liebler divergence [18] measures the relative entropy between two probability distributions. Yadav et al. [32], for example, use a maximum-likelihood classifier [19]—with KL as its distance metric—for detecting malicious AGDs. The intuition is that malicious algorithmically generated domain names have character and n-gram frequency distributions that are significantly different from character distributions derived from benign domains.

*Jaccard Index:* The Jaccard Index is a similarity metric that counts the bigram occurrences in two strings and measures the amount of overlap between them. The idea is that randomized strings (or supposedly-malicious domain names) should have a set of bigrams that is different than bigrams in a normal (non-malicious) English-based string.

*Levenshtein Distance:* Edit distance is a measure between two strings, which counts the number of insertions, deletions, and substitutions to transform one string to another [19]. In the case of algorithmically generated domains, the assumption is that because a group of malicious domain names are randomly generated, their average edit distance should be higher than a group of non-malicious names.

Each of the similarity metrics operate on a group of domain names in order to achieve detection accuracy. Yadav et al. [31, 32], for example, recommend 200 to 500 domain names for best results. To create the necessary clusters for evaluation, we apply the method suggested in [32] wherein clusters are created by mapping domain names to their corresponding server IP addresses over a specific time window. This is done because botmasters tend to register multiple domains to the same server IP address.

We evaluated 42,870 domain name clusters from March 19, 2012 which contained 13 sink-holed instances (or clusters) of the conficker bot. A sinkhole is a name server that redirects malicious traffic to some address under control of the defender, in order to contain the malware. We manually inspected each cluster to ensure no other bot instances were found and supplemented the ground truth with four clusters (each containing 300 entries) of AGDs sampled from our list of known botnets (see §III-A). Additionally, since the Kullback-Liebler and Jaccard Index based classifiers require both benign and malicious training models, we built the benign training model using the top 10,000 domains from alexa.com and the malicious training model using the list of 2,500 domains from our blacklist.

*Findings:* Table III shows the results of using a Kullback-Leibler-based classifier, which achieved the highest accuracy in our evaluation. The classifier is able to identify the presence of all of the malicious samples, but even then, it has an exceedingly high false positive rate of 28%. A large fraction of CDN traffic is incorrectly classified as malicious, which is one factor contributing to the high false positive rate. A natural way to improve the performance of the classifier would be to whitelist popular CDNs [31]. Figure 4 shows the result of using the different classifiers with varying domain cluster sizes and whitelisted CDNs. We find that, even with filtering, the KL classifier achieves a 12.5% false positive rate with a cluster size of at least 200 domain names. As we show later in Section IV-A, such large cluster sizes have implication on detection rates, processing speeds, and accuracy.

The classifier using Jaccard's Index achieved the second highest accuracy amongst the techniques evaluated.

TABLE III: Results of the KL classifier for Mar.19, 2012.

| Domain Source | Daily | |
|---|---|---|
| | True Positives | False Positives |
| Bot Traffic | 1.0 | 0.28 |
| Facebook (CDN) | 0.65 | 0.35 |
| Cloudfront (CDN) | 0.36 | 0.64 |
| Amazon (CDN) | 0.72 | 0.28 |
| Google IPv6 (CDN) | 0.18 | 0.82 |

However, as Figure 4 suggests, the accuracy came at a high cost—a true positive rate of 92% with corresponding false positive rate of 14%. Furthermore, the Jaccard-based classifier is the slowest of all techniques tested, which limits its ability to be used in an online fashion.

Figure 4 also shows the classification results using an edit distance approach. The plot shows the true and false positive rates when varying the edit distance threshold. In that evaluation, we generated the edit distance values for groups of botnet and benign domain names within our training sets and used that to determine a threshold value that would separate normal traffic from malicious AGDs. We found that 70% of the benign groups had an average edit distance score of eight or below. Of the malicious groups, `conficker` averaged a score of eight, while `bobax` and `cridex` score between nine and eleven. `Zeus` was a consistent outlier with scores above 35.

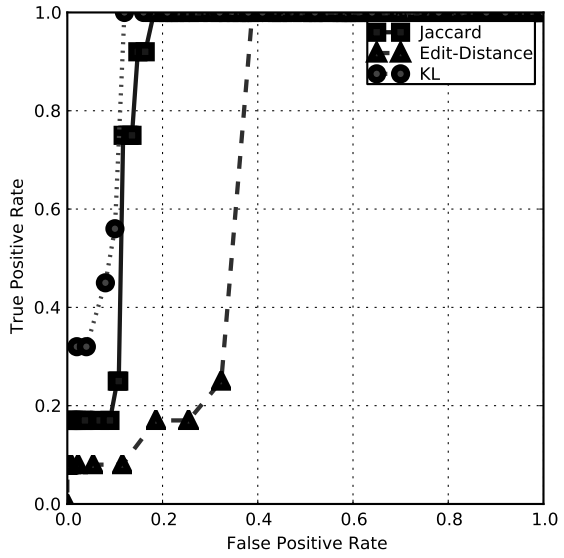


Fig. 4: ROC Curves for Jaccard Index, Edit Distance and KL Divergence using the daily dataset and CDN filtering.

### A. Shortcomings of Existing Methods

Overall, the application of a KL-based classifier performed reasonably well, providing classification decisions for all the domain clusters within a few minutes. The problem, however, is that it required on the order of a few hundred domain names in *each* cluster to provide accurate results. To see why this is problematic, we note that it may take several hours before a cluster meets the minimum threshold required to achieve the classification results given in Table III; in particular, during a one week period we observed eight `conficker` instances, one `cridex` and one `spambot`. Two of the `conficker` instances queried less than 200 randomly generated domain names, while the other six instances took almost three hours to query 100 domain names, and 3.5 days to query 500 domain names. The `cridex` and `spambot` instances generated less than 10 domain name lookups during 3.5 days. This rate of activity requires many days of monitoring before classification can occur, rendering the technique unusable for detecting and blocking malicious activity from these sources.

From an operational perspective, the Jaccard Index approach is appealing because of its ease of implementation and reasonable performance. The simplicity, however, comes at the cost of computation time: it took several hours to classify all the domain clusters in just one day's worth of DNS traffic. Another disadvantage is the fact that the approach is highly sensitive to the training dataset and the number of domain names in the cluster being evaluated.

Methods based on edit distance, on the other hand, have the advantage of not requiring training data and can operate on small clusters of names. That said, we found the edit distance approach to be the least effective of the techniques we evaluated. Its high false positive rates are tightly coupled with the difficulty of selecting an appropriate threshold value. For real-world deployments, the need to constantly monitor and fine tune these thresholds significantly diminishes its practical utility. This technique was also extremely slow, taking several hours to process the dataset.

Taken as a whole, our analyses indicate that the examined approaches are not robust enough to be used in production environments. This is particularly true if additional auxiliary information (e.g., realtime reputation information from various network vantage points in the DNS hierarchy [3]) is not being used to help address real-world issues that arise when dealing with the complexities of network traffic—where friend or foe can be easily confused. Moreover, these techniques all make the fallacious assumption that anomalous behavior equates to malicious activity and so the use of algorithmically generated names for benign purposes undermines this assumption.

## V. Our Approach

To address the accuracy and performance issues inherent in the aforementioned approaches, we present a lightweight algorithm based on sequential hypothesis testing which examines traffic patterns rather than properties of a domain name in order to classify clients. The intuition behind our approach is that a compromised host tends to "scan" the DNS namespace looking for a valid command and control server. In doing so, it generates a relatively high number of unique second-level domains that elicit more NX responses than a benign host. As a result, the problem lends itself to using sequential hypothesis testing [28] to classify clients as bots based on online observations of unique NX responses.

The general idea is illustrated in Figure 5. In Step ❶, we reduce the amount of data we analyze by over 90%, retaining only NX response packets. Next, we extract the client IP address and zone of the domain name from each packet (Step ❷) and then filter NX responses for well-known

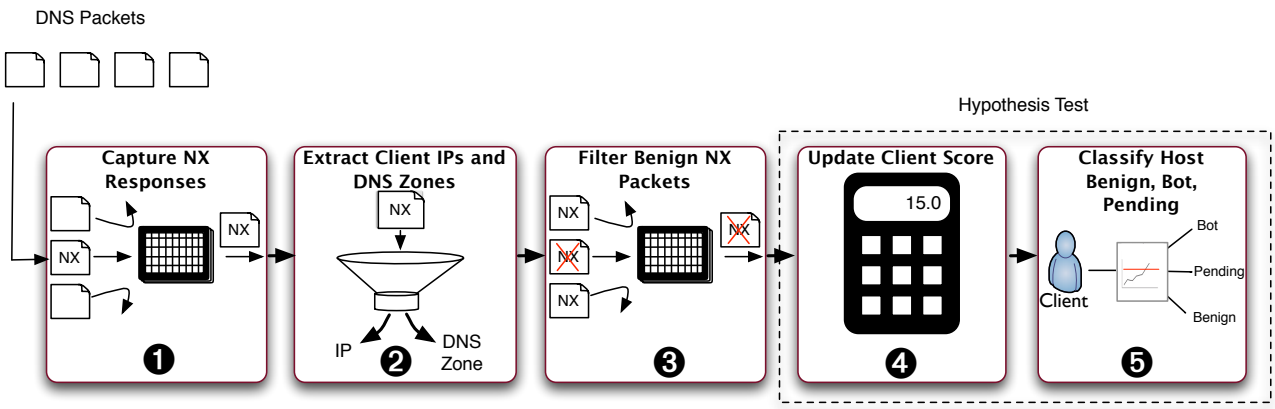


Fig. 5: High-level overview of the our workflow.

(benign) domain names (Step ❸). The zone information of the remaining domain names are used to adjust the client's score. The score is adjusted up or down based on whether the client has seen the zone before (Step ❹). Finally, the new score is compared to both a benign threshold and a bot threshold. If either threshold is crossed, then the client is classified; otherwise, the client remains in the pending state waiting for another NX response (Step ❺).

Our goal is to accurately classify a host as a bot or benign while observing as few outcomes as possible. To that end, we approach the problem by considering two competing hypotheses, defined as follows:

**Null hypothesis** $H_0$ = the local client $l$ is benign.
**Alternative hypothesis** $H_1$ = the local client $l$ is a bot.

A sequential hypothesis test observes success and failure outcomes $(Y_i, i = 1, ...n)$ in sequence and updates a test score after each outcome. A success pushes the score for client $l$ towards a benign threshold while a failure pushes the score towards a bot threshold. In our context, we define a success and failure outcome as follows:

**Success** $Y_i = 1$ Client $l$ receives an NX response for a DNS zone it has already seen.
**Failure** $Y_i = 0$ Client $l$ receives an NX response for a unique DNS zone.

For simplicity, we consider a DNS zone as a portion of the DNS namespace that is administered by a single entity (e.g., the `google.com` zone is administered by Google).

The size of the step taken towards the thresholds is decided by the values $\theta_0$ and $\theta_1$. The value of $\theta_0$ is defined as the probability that a benign host generates a successful event while $\theta_1$ is the probability that a malicious host generates a successful event. More formally, $\theta_0$ and $\theta_1$ are defined as:

$$\begin{aligned} P_r[Y_i = 0|H_0] = \theta_0, P_r[Y_i = 1|H_0] = 1 - \theta_0 \\ P_r[Y_i = 0|H_1] = \theta_1, P_r[Y_i = 1|H_1] = 1 - \theta_1 \end{aligned} \quad (1)$$

Using the distribution of the Bernoulli random variable, we calculate the sequential hypothesis score (or likelihood ratio) as follows:

$$\Lambda(Y) = \frac{Pr[Y|H_1]}{Pr[Y|H_0]} = \prod_{i=1}^{n} \frac{Pr[Y_i|H_1]}{Pr[Y_i|H_0]} \quad (2)$$

where $Y$ is the vector of events observed and $Pr[Y|H_i]$ represents the probability mass function of event stream $Y$ given $H_i$ is true. The score is then compared to an upper threshold $(\eta_1)$ and a lower threshold, $(\eta_0)$. If $\Lambda(Y) \leq \eta_0$ then we accept $H_0$ (i.e., the host is benign) , and if $\Lambda(Y) \geq \eta_1$ we accept $H_1$ (i.e., the host is malicious). If $\eta_0 < \Lambda(Y) < \eta_1$ then we are in the pending state and must wait for another observation.

The thresholds are calculated based on user selected values $\alpha$ and $\beta$ which represent the desired false positive and true positive rates respectively. The parameters are typically set to $\alpha = 0.01$ and $\beta = 0.99$. The upper bound threshold is calculated as:

$$\eta_1 = \frac{\beta}{\alpha} \quad (3)$$

while the lower bound is computed as:

$$\eta_0 = \frac{1 - \beta}{1 - \alpha} \quad (4)$$

A key challenge in our setting is that because we monitor internal hosts, we see all client-side DNS traffic, including the benign queries (e.g., from web browsing sessions) as well as the malicious queries of the bot. However, since the benign activities mostly result in successful DNS responses, we can safely filter such traffic and focus on NX responses (where the bot has more of an impact). This strategy has the side effect of discarding the vast majority of DNS packets, thereby allowing us to operate at higher network speeds. We further filter the traffic by only processing second level DNS zones, rather than fully qualified domain names (FQDNs). We focus on second-level domains since most bots generate randomized second-level domains in order make it more difficult to blacklist them and to hamper take-down efforts.

We also take advantage of the fact that NX traffic access patterns for benign hosts follows a Zipf's distribution.

Indeed, over 90% of NX responses in our data are to 100 unique zones. The bot DNS queries lie in the tail of the Zipf curve, hidden by the vast amounts of benign traffic. To quickly sift through this mountain of data, we apply a Zipf filter comprising the most popular zones[1] and remove matches using a perfect hash. Finally, each time a client is declared benign its state is reset, forcing it to continuously re-prove itself.

*Limitations:* A straightforward evasive strategy is for a bot to spread its DNS queries across a large time window, essentially implementing a low and slow approach. While this is a viable strategy, we believe that doing so drastically slows a bot's ability to communicate with its command-and-control server — resulting in a clear win for defenders. Another strategy is to attempt to increase our state tracking overhead by making DNS requests from spoofed IPs. That said, in modern networks practical IP spoofing is readily detectable, especially when media access control (MAC) address registration is enforced. Alternatively, if IP spoofing is a significant concern, one could enforce DNS over TCP for local hosts connecting to internal resolvers.

## VI. Evaluation

Unlike the approaches [3, 31, 32] discussed earlier, we classify client IPs based on NX traffic patterns. As such, ground truth in our case is a list of clients exhibiting botnet-like behavior. To attain ground truth for the analyses that follow, we excluded any hosts that did not receive NX responses, and then discarded any connections that received NX responses from white-listed NX zones (e.g., senderbase.org). The white-list was created by manually inspecting the top 100 zones of domain names that elicit NX responses[2]. We then cross-referenced the domain names from the remaining clients against well-known blacklists. While this approach was helpful in identifying known bots, it clearly is of little help in identifying new bots that were yet discovered in the wild on the date of our analysis. To address this possibility, we applied two techniques. First, we performed lookups on domains that received NX responses in March to see if any of those domains were now sink-holed. And second, we hand-labeled the remaining clients on whether they had similar name structure as existing AGDs, generated a sequence of at least two or more domains names that followed a similar structural convention (e.g., character set and length of the domain name), and received NX responses. In the end, we found a total 255 clients: 66 clients on March 18th, 101 on the 19th and 88 on the 20th.

*On Parameter Selection:* Both $\theta_0$ (the probability that a *benign* host sees a success event) and $\theta_1$ (the probability that a *malicious* host sees a success event) are parameters that must be set appropriately in any real-world deployment. Therefore, they must be calculated for each deployment of our sequential hypothesis framework. Fortunately, we show that these parameters can be robustly computed from a relatively small amount of traffic. Recall that in §V, we defined a successful outcome as one where a host receives NX responses for a zone it has already contacted at least once in the past, and a failure outcome every time a NX response is generated for a zone not seen previously. To estimate these parameters, we simply track NX responses on a per-client basis for a set window of time, counting successes and failures.

[1] In our empirical evaluations, we use the top 100 zones

[2] We confirmed the stability of the white-list using historical NX traffic from within our network spanning several months.

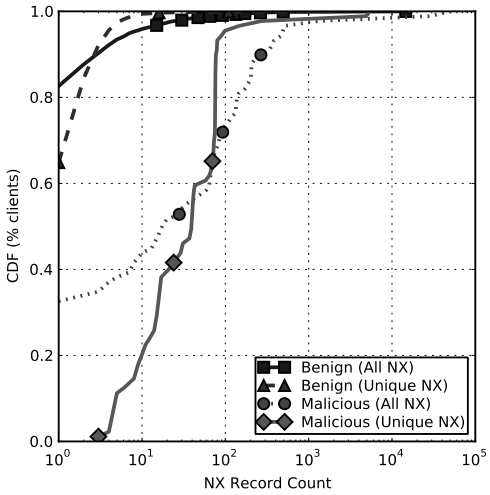


Fig. 6: NX zone counts for benign and malicious clients.

From our empirical analyses, we find that the majority of DNS traffic is in fact benign, and the AGD traffic comprises less than $2\%$ of the overall traffic. We expect this to be true within most enterprise networks, allowing us to estimate $\theta_0$ by simply computing the percent of successful connections for all NX traffic observed in that window of time.

Estimating $\theta_1$, on the other hand, is more difficult. If an operator is fortunate enough to have an oracle by which she could separate benign from malicious hosts and build ground truth for her network, then she could infer $\theta_1$ by computing the percent of successes generated by malicious hosts. However, in the real world, access to such an oracle is difficult, if not impossible; hence, $\theta_1$ must be estimated by other means. In our work, we have found that by discarding all clients that generate less than $\delta$ failure events, we can achieve a reasonable approximation of $\theta_1$ from the remaining traffic. This is based on the fact that bots tend to generate far more failure events than benign hosts.

Figure 6 offers insight into why the application of sequential hypothesis testing makes sense in our setting. Notice that ninety-five percent of benign hosts receive NX responses for four or less unique zones, while 98% of bots receive NX responses for four or more hosts over a day. Hence, by monitoring only NX traffic, we see a clear delineation between benign and infected hosts. Based on this observation, we set $\delta = 4$ for the approximation of $\theta_1$ within our network.

### A. Offline Analysis

In order to evaluate the accuracy of our classifier, we used a $k$-fold cross-validation. Cross-validation is a method typically used to assess the performance of a classifier by partitioning data into $k$-subsets. One subset is used for

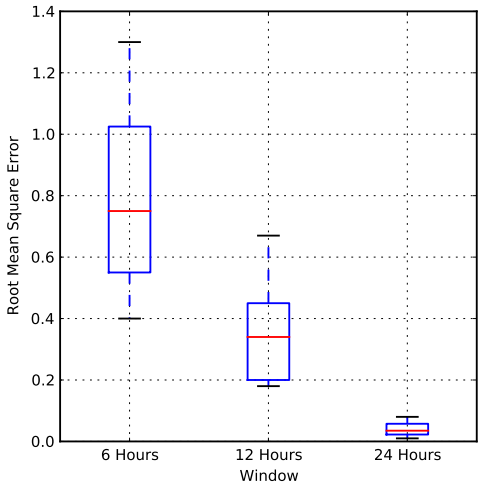


Fig. 7: Box-and-whisker plot of the error estimation for k-fold cross-validation for varying training window sizes

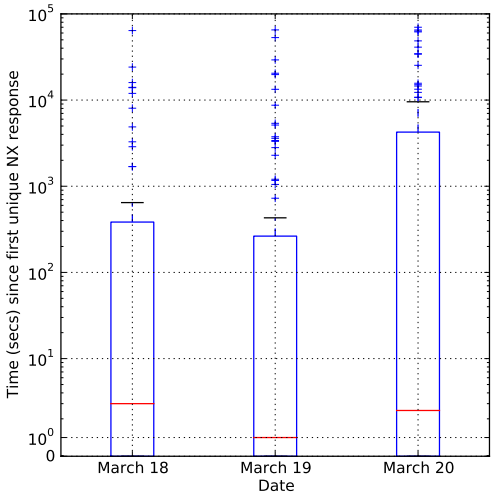


Fig. 8: Classification time after first unique NX response.

training, while the others are used for testing. This process is repeated $k-1$ times until each of the $k$ subsets has served as a training set. In the results that follow, we estimated $\theta_0$ and $\theta_1$ based on the designated training set, then fixed these values on the testing data.

We performed a set of experiments to estimate an appropriate training window size. We chose $\Delta = 6$, 12, and 24-hour intervals as window-size candidates, dividing the dataset by each. We then split the ground-truth data based on the clients observed within those time windows. Similarly, $\theta_0$ and $\theta_1$ were estimated for each of the time windows using the technique discussed earlier. We ran a $k$-fold cross-validation for each of the intervals (where $k = 10, 5, 3$) and compared the prediction errors between them.

Figure 7 shows the results of each experiment. The prediction errors are computed as the root mean square error over two repeated runs and plotted as a Box-and-Whisker plot to show the mean and variance within each experiment. Our experiments indicate that a training window of $\Delta = 24$ hours yields the best results with an average root mean square error of 0.034. The accuracy of the classifier is given in Table IV.

TABLE IV: Accuracy for $k$-fold cross validation experiments for varying training window sizes ($\Delta$).

| $k$-fold validation | **Window Size ($\Delta$)** | **TP** | **FP** |
|---|---|---|---|
| $k = 3$ | 24 hours | .94 | .002 |
| $k = 5$ | 12 hours | .86 | .031 |
| $k = 10$ | 6 hours | .81 | .048 |

Intuitively, a window size of 24 hours provides the best results, because it takes into consideration the diurnal patterns in network traffic. Therefore, the remainder of our experiments use 3-fold cross-validation.

*On Classification Speed:* One of the major drawbacks of existing approaches is the amount of time that elapses before a host can be classified (see §IV). Although we do not have definitive information on exactly when a client is infected, we approximate infection time as the moment of the first unique NX response for a particular client. We found that, on average, our technique detects bots within three to four unique NX responses (with a maximum of nine).

Figure 8 shows the time (in seconds) taken to classify a client as a bot. The majority of bots are correctly classified within only a few seconds of seeing the first unique NX response—primarily because they perform tens of queries at once. Some bots, however, take a more delayed approach, making singular queries at uniform time intervals. In this case, it can take several hours to detect them.

That said, since bots must receive instructions from a command-and-control server, a more appropriate measure might be to compute the time elapsed before the bot successfully connects with its command center. We term this connection the "rendezvous point." Obviously, we desire the ability to detect the bot before it makes that connection.

To perform such analyses, we choose a random sample of 20 prominent bots from each of the three days and located their rendezvous point by hand. Figure 9 shows the difference between the rendezvous time and classification time. In 10 [of 60] cases, the rendezvous takes place before the bot is detected.In 16 cases, we detected the bot at the same time as the rendezvous point, while in the remaining cases, we declared the host as a bot seconds before the actual contact with the command-and-control server was made. Overall, in 83% of the cases, we detect bots either shortly before or during the liaison with their command-and-control servers. The differences in detection time from the 19th to the 20th are due to a large AGD-based compromise that occurred on campus on the 20th. The event was detected by our approach and the results were shared with our campus network operators.

*On Hosts Pending Classification:* Next, we consider what fraction of clients remain in the pending state at the end

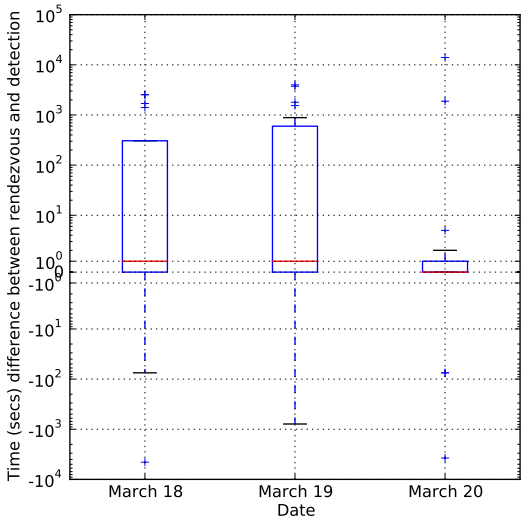


Fig. 9: Time between classification and rendezvous.

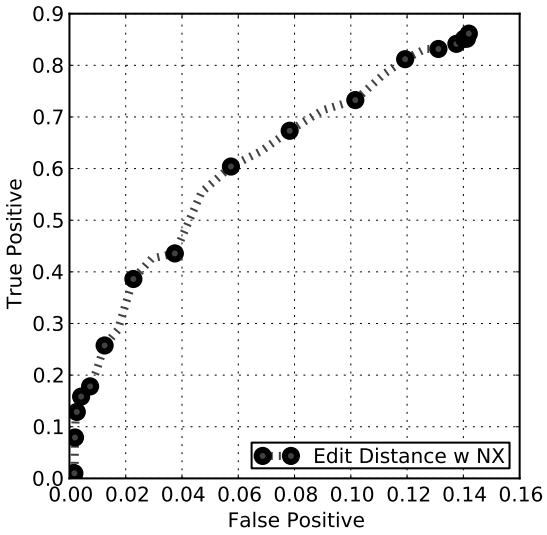


Fig. 10: ROC curve for edit distance using NX responses.

of a given time window. We found that at the end of each day (i.e., $k = 3$), 10% of the hosts were in the pending state. Of those clients, 70% had a response from one (unique) NX zone, 90% two or less, and 99% four or less. All but one of the 18 bots (from the ground truth) that had not been classified by the sequential hypothesis test (6 [of 66] on the 18th, 10 [of 101] on the 19th, and 2 [of 88] on the 20th), were in the pending state. These 18 clients had generated, on average, two or less unique NX responses in the allocated time window.

Upon closer inspection we find that 95% of the pending hosts were in that state for at least 2 1/2 hours and some for almost the entire 24-hour period. This implies that as the pending hosts age, strategies are required for removing these hosts from the pending list in order to reduce our memory footprint. One strategy is to use an approach similar to our Zipf Filter, and generate a filter based on the top $n$ unique zones in the pending host list. With a cursory analysis using the top 100 pending zones, we removed 30% of the hosts in the pending state. Another option is to randomly prune a certain percentage of the pending hosts based on their age or their unique NX response count. We leave such extensions as future work.

*Comparison to Existing Work::* Lastly, to perform a direct comparison to approaches that make use of NX traffic, we implemented an approximation of the time binning algorithm of Yadav and Reddy [31]. The work extends the Edit-Distance technique (see §IV) to individual clients by exploiting the fact that bots tend to make queries in bursts. Their assumption is that by incorporating NX responses, domain samples can be gathered quicker than with successful DNS queries alone.

We created the prerequisite clusters by collecting all queries that elicited an NX response within 64 seconds (before and after) of a successful rendezvous query for each client [31]. The edit distance measure is then applied to the clusters, and the average edit distance value for each cluster is compared with a threshold to determine whether the cluster is malicious or not. We then built clusters for each potential rendezvous point in the March 19th dataset. Clusters that contacted well-known white-listed domains (e.g., `facebook.com`) were filtered using the 100,000 most popular domain names (41,758 zones) from the March dataset. This left 455,500 domain name clusters spanning 10,758 unique client IP addresses.

Figure 10 shows the true and false positive rates when adjusting the edit distance threshold value. As with the other edit distance approaches (see §IV), this extension also resulted in a high false positive rate (of over 14%). Even with the extra domains collected from the NX traffic, we were only able to gather at most 80 AGDs per cluster—far below the 200 domain names required for accuracy [31]. In fact, only 17 of the clients had clusters with more than 50 domain names. An additional limitation is that Yadav and Reddy [31]'s approach requires storage of both successful and NX domain names, which adversely affects its runtime performance. By contrast, we store only the DNS zones for each client, and only require updating a hypothesis test score for each observed event.

### B. Visualizing AGD Traffic

In an enterprise setting, a security analyst usually must investigate the list of hosts declared as bots by any of the aforementioned techniques. After the detection process has completed, and to help reduce the cognitive load on the analyst, we provide a technique for grouping clients based on their AGD traffic. Our technique capitalizes on observations we made while investigating the output of our algorithm, namely that (1) multiple clients tend to be infected with the same type of bot, and (2) the infected hosts generate the same domain lookups because of the use of a global seed.

These observations lend themselves to a natural grouping procedure for a set $S$, where $S$ denotes the clients declared

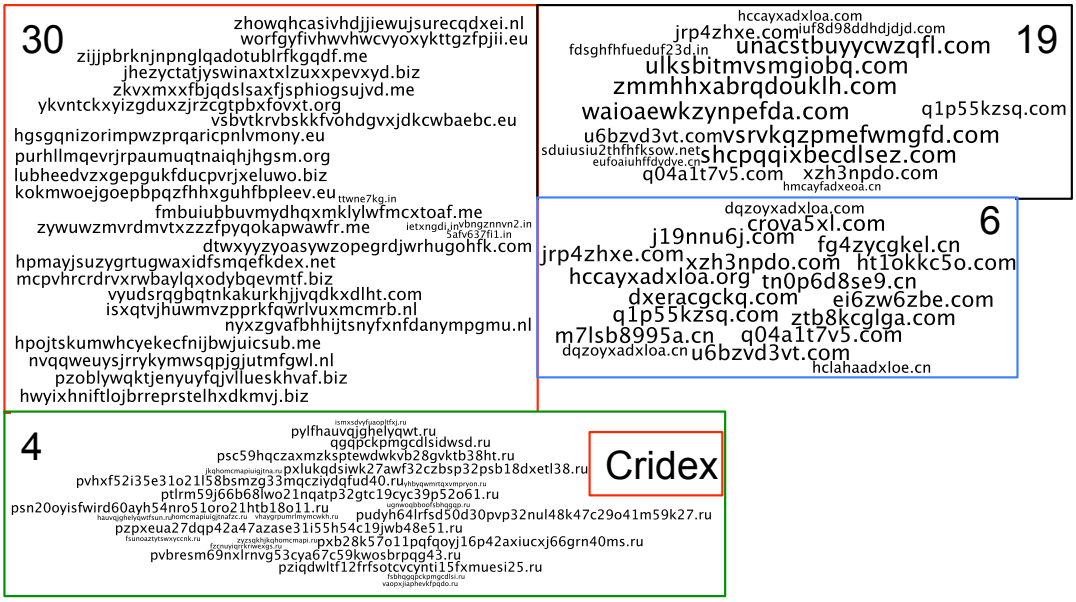


Fig. 11: AGDs found with hierarchical clustering technique. Size of domain name indicates its prevalence in the cluster. Count indicates number of clients found in cluster.

as bots during some time window:

- $\forall i \in S$, let $S_i$ be the tuple $\langle l, n_0, n_1 ... n_m \rangle$ where $l$ is the client's IP, and $n_0, ...n_m$ the list of NX zones queried.
- Let $G = \cup\ n_0, \ldots n_N \in S$.
- For each client $l$, let $b_l$ be a bitmap of length $N$ representing the zones in $G$ and set the bits to 1 for the domains that the client has queried.
- Let the distance between two clients $l_1$ and $l_2$ be $distance(l_1, l_2) = \frac{1}{B_{l1,l2}}$, where $B_{l1,l2}$ is the sum of the number of bits set of the resulting ANDed bitmaps.
- Set $S$ is clustered using hierarchical clustering [8].

Using this approach, we clustered the data for March 20th. The 747 clients were grouped creating 23 clusters of two or more clients. Of those clusters, four contain 59 of the 88 bots found in the ground truth. Figure 11 shows a sampling of the AGDs generated by the clients in each cluster. AGDs in the largest fonts are ones that appear in all clients in the cluster. The smaller the font, the less appearances the domain made.

To attain more information about the botnet families for these clusters, we searched publicly available blacklists and anti-virus websites for information on the domains. We then performed lookups on the domains (e.g., using `dig`) to see if they were sink-holed. Three of the four clusters were sink-holed, and the fourth had known `cridex` AGDs (e.g., aecgrgbjgaofrilwyg.ru).

The remaining 29 bots (in the ground truth) did not cluster. 18 of those hosts generated similarly structured domains, but no two hosts generated the exact domains (see Table V). Little information was found on the origins of these domains. Another 3 clients contain multiple domains that are sink-holed to an address linked to the `TDSS` botnet [10].

TABLE V: AGDs that clustered by domain length.

| IPs | Example AGDs |
|---|---|
| *IP 1* | kt2syggf436dtag458.com |
| *IP 2* | kt2syggf436dtag182.com |
| *IP 1* | jhbvyvuyvuyvuvujvuvrf6r66.com |
| *IP 2* | bbgyujh6uh7i5y67567y5b7.com |
| *IP 3* | csfsdfvdbdbbfbnmcnq8858.com |
| *IP 1* | 27613082671222563732.com |
| *IP 2* | 79735931367645588627.com |
| *IP 3* | 13348318318656728693.com |
| *IP 1* | e7722746d7c642c2a6793cb8935c45da.com |
| *IP 2* | 80b8024c08484f029d1c229f5030c741.com |
| *IP 3* | c62fb768db0c4d179bfb200fcc415c9f.com |

## VII. Analysis of Live Traffic

To further demonstrate the utility of our technique, we implemented an online version and deployed it on our campus network. For the live test, we used an Endace 9.2X2 Data Acquisition and Generation (DAG) card connected to a host machine. This setup was used to monitor DNS traffic at the border of our campus network. The DAG captures DNS packets at line rates and places them in a shared memory buffer without relying on the host. As a result, we can take full advantage of the host (a 2.53 Ghz Intel Xeon core processor with 16GB memory) for packet inspection. As DNS packets are placed into the shared memory buffer by the DAG card, they are assigned to an available core to perform the initial dissection. If the packet requires further processing, it is passed from core to core in a pipeline, where each core is assigned a specific task. This design allows us to easily scale by dynamically assigning packets and tasks across multiple cores.

As Sommer et al. [21] note, utilizing multi-core architectures to provide parallelism is important in order to be able to provide online network analysis at line speeds. To that end, our network capture and analysis engine supports

multi-threaded processing and uses two basic thread models: a staged pipeline to stitch together processing stages (dissection, signature matching, statistics etc), and a pool model to parallelize processing within each stage. Each stage is run on a different core and we implement lock-free ring buffers [26] to ensure high throughput across the pipeline buffer and ensure data synchronization. The lock-free data structure was built using Compare-and-Swap (CAS) primitives provided by the underlying x86 architecture. The packet dissection is performed by protocol specific finite state machines (FSMs). Layers within a network packet are modelled as states and transitions between states are modelled as events. Using FSMs allows us to add and remove protocol dissectors easily and provides us with the ability to dynamically assign "processing depth" for an individual packet. For example, our DNS FSM allows the programmer to decide how far into the packet to dissect.

Our online evaluation spans a period of 24 hours in November, 2012. The traffic reflects well-known diurnal patterns, with a large mid-day peak of approximately 80,000 DNS connections per minute. However, NX traffic accounts for less than 10% of the overall traffic, which highlights one of the benefits of using such data for botnet detection. Our throughput analysis shows that we can operate on live traffic with zero packet loss and $< 15\%$ CPU utilization. Note that by using NX traffic, DNS zones (rather than fully qualified domain names), domain name caching, and Zipf filters, we are able to store state information on the order of megabytes versus gigabytes. In larger deployments, one could use space efficient data structures (e.g., bloom filters) to keep track of state for several million IP addresses. We leave this as an exercise for future work.

Analysis of our results show 63 cases of suspected or known malicious traffic. Included in our findings were the `TDSS` and `Z` bots, numerous spambots, an `OSX.FlashFake` trojan and a `FakeAV` trojan. We also detected traffic of `RunForestRun` [25] and `BlackHole` [23]. One noteworthy discovery was that of the so-called Italian typo-squatting trojan [7] that uses domains that are misspellings of existing domains (e.g. `gbazzetta.it,gazzxetta.it`). Interestingly, the domain names used by this trojan would have relatively low edit distance scores making it difficult to detect them using the similarity-based techniques in §IV.

## VIII. Conclusion

In this paper, we study currently available techniques for detecting malicious, algorithmically generated, domain names. Our treatment centers on high accuracy and timeliness, which are key criteria in operational settings. We show that while contemporary techniques can detect the presence of malicious domain names, they incur high false positive rates, and require long observation periods before classification can occur. We address many of the shortcomings of contemporary approaches by presenting a lightweight technique based on sequential hypothesis testing. Our approach takes advantage of the fact that bots generate a relatively high number of unique NX responses when searching for a command-and-control server. Our extensive empirical evaluations show that we are able to classify hosts in as little as three to four NX responses, on average. Moreover, the lightweight nature of our approach makes it well-suited for real-world deployment.

## IX. Acknowledgements

We express our gratitude to Stan Waddell and Alex Everett of the Information Technology Service Office and our networking staff (especially, Murray Anderegg, Bil Hayes and Jim Gogan) for their efforts in deploying the infrastructure for this study. The researchers and the Technology Service Office have a longstanding memorandum of understanding in place to collect anonymized network traffic on campus. The memorandum covers specific uses and types of networking data, as well as conditions for securing and accessing such data. We also thank Jay Aikat, Michael Bailey, Kevin Snow, Andrew White and the anonymous reviewers for their insightful comments. This work is supported in part by the National Science Foundation, under award numbers 0831245 and 1127361 and the Natural Sciences and Engineering Research Council of Canada.

## References

[1] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *USENIX Security Symposium*, 2010.

[2] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *USENIX Security Symposium*, 2011.

[3] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *USENIX Security Symposium*, 2012.

[4] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains using Passive DNS Analysis. In *Symposium on Network and Distributed System Security*, Feb. 2011.

[5] K. Born and D. Gustafson. Detecting DNS Tunnels Using Character Frequency Analysis. In *Proceedings of the Annual Computer Security Conference*, 2010.

[6] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5), Sept. 2002.

[7] A. Eckelberry. Massive italian typosquatting ring foists malware on users. http://goo.gl/4ZzMI, 2007.

[8] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley Series in Probability and Statistics. Wiley, 2011.

[9] M. Felegyhazi, C. Kreibich, and V. Paxson. On the potential of proactive domain blacklisting. In *USENIX Conference on Large-Scale Exploits and Emergent Threats*, 2010.

[10] S. Golovanov and I. Soumenkov. TDL4 Top Bot. See http://goo.gl/23BaA, 2011.

[11] S. Hao, N. Feamster, and R. Pandrangi. Monitoring the Initial DNS Behavior of Malicious Domains. In *ACM SIGCOMM Internet Measurement Conference*, 2011.

[12] J.-W. Ho, M. Wright, and S. Das. Fast detection of mobile replica node attacks in wireless sensor networks using sequential hypothesis testing. *IEEE Transactions on Mobile Computing*, 10(6):767 –782, June 2011.

[13] ISC. Google Chrome and (weird) DNS Requests. http://goo.gl/j48CA, 2011.

[14] N. Jiang, J. Cao, Y. Jin, L. E. Li, and Z.-L. Zhang. Identifying suspicious activities through dns failure graph analysis. In *International Conference on Network Protocols*, pages 144–153, 2010.

[15] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy*, May 2004.

[16] J. Jung, R. Milito, and V. Paxson. On the adaptive real-time detection of fast-propagating network worms. *Journal of Computer Virology*, 4:197–210, 2008.

[17] D. Kaminsky. Black ops 2008–its the end of the cache as we know it. *Black Hat USA*, 2008.

[18] S. Kullback and R. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22 (1):79–86, 1951.

[19] P. H. R.O. Duda and D. Stork. *Pattern Classification*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, Sept. 2007.

[20] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *Symposium on Recent Advances in Intrusion Detection*, pages 59–81, 2004.

[21] R. Sommer, V. Paxson, and N. Weaver. An architecture for exploiting multi-core processors to parallelize network intrusion prevention. *Concurrency and Computation: Practice & Experience*, 21(10):1255–1279, July 2009.

[22] S. Son and V. Shmatikov. The Hitchhiker's Guide to DNS Cache Poisoning. In *International Conference on Security and Privacy in Communication Networks*, Sept. 2010.

[23] Sophos Inc. Exploring the blackhole exploit kit. http://goo.gl/ZhLvp, 2012.

[24] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *ACM Conference on Computer and Communications Security*, pages 635–647, 2009.

[25] Unmask Parasites. Runforestrun and pseudo random domains. http://goo.gl/xRWtw, 2012.

[26] J. Valois. Implementing lock-free queues. In *International Conference on Parallel and Distributed Computing Systems*, pages 64–69, 1994.

[27] R. Villamarn-Salomn and J. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *IEEE Consumer Communications & Networking Conference (CCNC)*, 2008.

[28] A. Wald. *Sequential Analysis*. John Wiley and Sons, Inc., 1947.

[29] N. Weaver, S. Staniford, and V. paxson. Very fast containment of scanning worms, revisited. In *Malware Detection*, pages 113–145. 2007.

[30] A. White, S. Krishnan, M. Bailey, F. Monrose, and P. Parros. Clear and Present Data: Opaque Traffic and its Security Implications for the Future. In *Symposium on Network and Distributed System Security*, Feb. 2013.

[31] S. Yadav and A. N. Reddy. Winning with DNS Failures: Strategies for Faster Botnet Detection. In *International Conference on Security and Privacy in Communication Networks*, 2011.

[32] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *ACM Internet Measurement Conference*, pages 48–61, 2010.