Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis

Sandeep Yadav, *Student Member, IEEE*, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, *Fellow, IEEE, Member, ACM*, and Supranamaya Ranjan, *Member, IEEE*

***Abstract*—Recent botnets such as Conficker, Kraken, and Torpig have used DNS-based "domain fluxing" for command-and-control, where each Bot queries for existence of a series of domain names and the owner has to register only one such domain name. In this paper, we develop a methodology to detect such "domain fluxes" in DNS traffic by looking for patterns inherent to domain names that are generated algorithmically, in contrast to those generated by humans. In particular, we look at distribution of alphanumeric characters as well as bigrams in all domains that are mapped to the same set of IP addresses. We present and compare the performance of several distance metrics, including K–L distance, Edit distance, and Jaccard measure. We train by using a good dataset of domains obtained via a crawl of domains mapped to all IPv4 address space and modeling bad datasets based on behaviors seen so far and expected. We also apply our methodology to packet traces collected at a Tier-1 ISP and show we can automatically detect domain fluxing as used by Conficker botnet with minimal false positives, in addition to discovering a new botnet within the ISP trace. We also analyze a campus DNS trace to detect another unknown botnet exhibiting advanced domain-name generation technique.**

***Index Terms*—Components, domain flux, domain names, Edit distance, entropy, IP fast flux, Jaccard index, malicious.**

I. Introduction

RECENT botnets such as Conficker, Kraken, and Torpig have brought in vogue a new method for botnet operators to control their bots: DNS "domain fluxing." In this method, each bot algorithmically generates a large set of domain names and queries each of them until one of them is resolved, and then the bot contacts the corresponding IP address obtained that is typically used to host the command-and-control (C&C) server. Besides for command-and-control, spammers also routinely generate random domain names in order to avoid detection. For instance, spammers advertise randomly generated domain names in spam e-mails to avoid detection by regular expression-based domain blacklists that maintain signatures for recently "spamvertised" domain names.

The botnets that have used random domain-name generation vary widely in the random word-generation algorithm as well as the way it is seeded. For instance, Conficker-A [29] bots generate 250 domains every 3 h while using the current date and time at UTC (in seconds) as the seed, which in turn is obtained by sending empty HTTP GET queries to a few legitimate sites such as *google.com*, *baidu.com*, *answers.com*, etc. This way, all bots would generate the same domain names every day. In order to make it harder for a security vendor to preregister the domain names, the next version, Conficker-C [28] increased the number of randomly generated domain names per bot to 50 K. Torpig [6], [31] bots employ an interesting trick where the seed for the random string generator is based on one of the most popular trending topics in Twitter. Kraken employs a more sophisticated random word generator and constructs English-language-like words with properly matched vowels and consonants. Moreover, the randomly generated word is combined with a suffix chosen randomly from a pool of common English noun, verb, adjective, and adverb suffixes, such as -able, -hood, -ment, -ship, or -ly.

From the point of view of botnet owner(s), the economics work out quite well. They only have to register one or a few domains out of the several domains that each bot would query every day, whereas security vendors would have to preregister *all* the domains that a bot queries every day, even before the botnet owner registers them. In all the cases above, the security vendors had to reverse engineer the bot executable to derive the exact algorithm being used for generating domain names. In some cases, their algorithm would predict domains successfully until the botnet owner would patch all his bots with a repurposed executable with a different domain-generation algorithm [31].

We argue that reverse engineering of botnet executables is resource- and time-intensive, and precious time may be lost before the domain-generation algorithm is cracked and consequently before such domain name queries generated by bots are detected. In this regard, we raise the following question: *Can we detect algorithmically generated domain names while monitoring DNS traffic even when a reverse-engineered domain-generation algorithm may not be available?*

Hence, we propose a methodology that analyzes DNS traffic to detect *if* and *when* domain names are being generated algorithmically as a line of first defense. Our technique for anomaly detection may be applied by analyzing groups of domains extracted from the DNS queries, seen at the edge of an autonomous system. Therefore, our proposed methodology can point to the presence of bots within a network and the network administrator can disconnect bots from their C&C server by filtering out DNS queries to such algorithmically generated domain names.

Our proposed methodology is based on the following observation: Current botnets do not use well-formed and pronounceable language words since the likelihood that such a word is already registered at a domain registrar is very high, which could

Manuscript received December 18, 2010; revised June 12, 2011 and November 04, 2011; accepted January 11, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Allman. Date of publication February 10, 2012; date of current version October 11, 2012.

S. Yadav and A. L. Narasimha Reddy are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: sandeepy@tamu.edu; reddy@ece.tamu.edu).

A. K. K. Reddy is with Microsoft, Redmond, WA 98052 USA (e-mail: ashwathr@microsoft.com).

S. Ranjan is with MyLikes Corporation, San Francisco, CA 94103 USA (e-mail: soups.ranjan@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2012.2184552

be self-defeating as the botnet owner then would not be able to control his bots. In turn, this means that such algorithmically generated domain names can be expected to exhibit characteristics vastly different from legitimate domain names. Hence, we develop metrics using techniques from signal detection theory and statistical learning that can detect algorithmically generated domain names that may be generated via a myriad of techniques: 1) those generated via pseudorandom string-generation algorithms, as well as 2) dictionary-based generators—for instance the one used by Kraken [3]–[5], as well as the Kwyjibo tool [13], which can generate words that are pronounceable, yet not in the English dictionary.

Our method of detection comprises of two parts. First, we propose several ways to group together DNS queries: 1) either by the Top Level Domain (TLD) to which they all correspond; or 2) the IP address to which they are mapped; or 3) the connected component to which they belong, as determined via connected component analysis of the IP-domain bipartite graph. Second, for each such group, we compute metrics that characterize the distribution of the alphanumeric characters or bigrams (two consecutive alphanumeric characters) within the set of domain names. Specifically, we propose the following metrics to quickly differentiate a set of legitimate domain names from malicious ones: 1) information entropy of the distribution of alphanumerics (unigrams and bigrams) within a group of domains. The distribution comparison is made using the Kullback–Leibler (K–L) divergence, which computes the "distance" between two distributions; 2) Jaccard index to compare the set of bigrams between a malicious domain name with good domains; and 3) Edit distance, which measures the number of character changes needed to convert one domain name to another.

We apply our methodology to a variety of datasets. First, we obtain a set of legitimate domain names via reverse DNS crawl of the entire IPv4 address space. Next, we obtain a set of malicious domain names as generated by Conficker and Kraken as well as model a much more sophisticated domain name generation algorithm: Kwyjibo [13]. Finally, we apply our methodology to one day of network traffic from one of the largest Tier-1 ISPs in Asia and South America and show how we can detect Conficker as well as a botnet hitherto unknown, which we call *Mjuyh* (details in Section V).

Our extensive experiments allow us to characterize the effectiveness of each metric in detecting algorithmically generated domain names in different attack scenarios. With our experiments, we observe that, in general, our tool's performance improves with a larger dataset used for analysis. For instance, when 200 domains are generated per TLD, then Edit distance achieves 100% detection accuracy with 8% false positives; when 500 domains are generated per TLD, Jaccard Index achieves 100% detection with 0% false positives. Applying our metrics to a campus DNS trace, we discover an unknown botnet that generates domain names by combining English dictionary words, which we detect with a false-positive rate of 2.56%.

The terminology we use in this work is as follows. For a host name such as *physics.university.edu*, we refer to *university* as the second-level domain label, *edu* as the first-level domain, and *university.edu* as the second-level domain. Similarly, *physics.university.edu* is considered a third-level domain, and *physics* a third-level domain label. The ccTLDs such as *co.uk* are effectively considered as first-level domains.

The rest of this paper is organized as follows. In Section II, we compare our work against related literature. In Section III, we present our detection methodology and introduce the metrics we have developed. In Section IV, we present the various ways by which domains can be grouped in order to compute the different metrics over them. Next, in Section V, we present results to compare each metric as applied to different datasets and trace data. Furthermore, in Section VI, we present the detection of malicious domains in a supervised learning framework. Section VII discusses limitations and improvements, with conclusions in Section VIII.

II. Related Work

Characteristics, such as IP addresses, whois records, and lexical features of phishing and nonphishing URLs have been analyzed by McGrath and Gupta [23]. They observed that the different URLs exhibited different alphabet distributions. Our work builds on this earlier work and develops techniques for identifying domains employing algorithmically generated names, potentially for "domain fluxing." Ma *et al.* [18] employ statistical learning techniques based on lexical features (length of domain names, host names, number of dots in the URL, etc.) and other features of URLs to automatically determine if a URL is malicious, i.e., used for phishing or advertising spam. While they classify each URL independently, our work is focused on classifying a group of URLs as algorithmically generated or not, solely by making use of the set of alphanumeric characters used. In addition, we experimentally compare against their lexical features in Section V and show that our alphanumeric distribution-based features can detect algorithmically generated domain names with lower false positives than lexical features. Overall, we consider our work as complementary and synergistic to the approach in [18]. Reference [10] develops a machine learning technique to classify individual domain names based on their network features, domain-name string composition style, and presence in known reference lists. Their technique, however, relies on successful resolution of DNS domain name query. Our technique, instead, can analyze groups of domain names, based only on alphanumeric character features.

With reference to the practice of "IP fast fluxing," e.g., where the botnet owner constantly keeps changing the IP addresses mapped to a C&C server, [25] implements a detection mechanism based on passive DNS traffic analysis. In our work, we present a methodology to detect cases where botnet owners may use a combination of both domain fluxing with IP fluxing by having bots query a series of domain names and, at the same time, map a few of those domain names to an evolving set of IP addresses. Also, earlier papers [21], [24] have analyzed the inner-working of IP fast flux networks for hiding spam and scam infrastructure. With regards to botnet detection, [15] and [16] perform correlation of network activity in time and space at campus network edges, and Xie *et al.* in [34] focus on detecting spamming botnets by developing regular expression-based signatures for spam URLs.

We find that graph analysis of IP addresses and domain names embedded in DNS queries and replies reveal interesting macro

relationships between different entities and enable identification of bot networks (Conficker) that seemed to span many domains and TLDs. With reference to graph-based analysis, [35] utilizes rapid changes in user–bot graphs structure to detect botnet accounts.

Statistical and learning techniques have been employed by various studies for prediction [11], [14], [26]. We employed results from detection theory in designing our strategies for classification [12], [32].

Several studies have looked at understanding and reverse-engineering the inner workings of botnets [3]–[5], [17], [27], [30], [31]. Botlab has carried out an extensive analysis of several bot networks through active participation [20] and provided us with many example datasets for malicious domains.

III. Detection Metrics

In this section, we present our detection methodology that is based on computing the distribution of alphanumeric characters for groups of domains. First, we motivate our metrics by showing how algorithmically generated domain names differ from legitimate ones in terms of distribution of alphanumeric characters. Next, we present our three metrics, namely K–L distance, Jaccard Index (JI) measure, and Edit distance. Finally, in Section IV, we present the methodology to group domain names.

A. Datasets

We first describe the datasets and how we obtained them.

1) ISP Dataset: We use network traffic trace collected from across 100+ router links at a Tier-1 ISP in Asia. The trace is one day long (21 h long, collected on November 3, 2009) and provides details of DNS requests and corresponding replies. This dataset consumes 66 GB of hard drive space and consists of 355 M packets with 38 M flows. These flows include various protocols such as DNS, HTTP, SMTP, etc. Of these flows, there are about 270 000 DNS name server replies. The queries captured at the ISP edge come from approximately 8400 clients that are mostly recursive DNS resolvers for smaller networks.

2) Nonmalicious DNS Dataset: We performed a reverse DNS crawl of the entire IPv4 address space to obtain a list of domain names and their corresponding IP addresses. The crawl was performed on March 4, 2010, and lasted for approximately 20 h. We perform this crawl through a system utilizing a DNS recursive nameserver for resolving DNS PTR queries issued for IPv4 addresses. We issue DNS PTR requests for various subnets and IP addresses, barring certain subnets belonging to military and unassigned zones. Such a request excludes DNS type-A records where multiple domains map to a single server (CDNs or Web site hosts). We further divided this dataset into several parts, each comprising domains that had 500, 200, 100, and 50 domain labels. The DNS Dataset is considered nonmalicious for the following reasons. Botnets may own only a limited number of IP addresses. Based on our study, we find that a DNS PTR request maps an IP address to only one domain name. The dataset thus obtained will contain very few malicious domain names per analyzed group. In the event that the bots exhibit IP fluxing, the botnet owners cannot change the PTR DNS mapping for IP addresses not owned. However, the malicious name servers may map domain names to any IP address.

3) Malicious Datasets: We obtained the list of domain names that were known to have been generated by recent Botnets: Conficker [28], [29], Torpig [31], Kraken [3], [5], Pushdo, etc. We obtain these domains from BotLab [20], which provides us with URLs used by bots belonging to the respective botnets. Specifically, we extract domains from 10 K URLs for Kraken, 25 K URLs for MegaD, 17 K URLs for Srizbi, 4.8 K URLs for Storm, and 5.4 K URLs for Pushdo. We also leverage our analysis with previous studies to identify the domains for Conficker and Torpig. As described earlier in the Introduction, Kraken exhibits the most sophisticated domain generator by carefully matching the frequency of occurrence of vowels and consonants as well as concatenating the resulting word with common suffixes in the end such as -able, -dom, etc.

4) Kwyjibo: We model a much more sophisticated algorithmic domain-name generation algorithm by using Kwyjibo [13], a tool that generates domain names that are pronounceable yet not in the English language dictionary, and hence much more likely to be available for registration at a domain registrar. The algorithm uses a syllable generator, where they first learn the frequency of one syllable following another in words in the English dictionary, and then automatically generate pronounceable words by modeling it as a Markov process.

B. Motivation

Our detection methodology is based on the observation that algorithmically generated domains differ significantly from legitimate (human) generated ones in terms of the distribution of alphanumeric characters. Fig. 1(a) shows the distribution of alphanumeric characters, defined as the set of English alphabets (*a–z*) and digits (0–9) for both legitimate as well as malicious domains.[1] We derive the following points. 1) First, note that both the nonmalicious datasets exhibit a nonuniform frequency distribution, e.g., letters "m" and "o" appear most frequently in the nonmalicious ISP dataset, whereas the letter "s" appears most frequently in the nonmalicious DNS dataset. 2) Even the most sophisticated algorithmic domain generator seen in the wild for Kraken botnet has a fairly uniform distribution, albeit with higher frequencies at the vowels: "a," "e," and "i." 3) If botnets of the future were to evolve and construct words that are pronounceable yet not in the dictionary, then they would not exhibit a uniform distribution as expected. For instance, Kwyjibo exhibits higher frequencies at alphabets, "e," "g," "i," "l," "n," etc. In this regard, techniques that are based on only the distribution of unigrams (single alphanumeric characters) may not be sufficient, as we will show through the rest of this section.

C. Metrics for Anomaly Detection

The K–L divergence metric is a nonsymmetric measure of "distance" between two probability distributions. The divergence (or distance) between two discretized distributions P and Q is given by $D_{\mathrm{KL}}(P \parallel Q) = \sum_{i=1}^{n} P(i) \log \frac{P(i)}{Q(i)}$, where n is the number of possible values for a discrete random variable. The probability distribution P represents the test distribution, and the distribution Q represents the base distribution from which the metric is computed.

[1]Even though domain names may contain characters such as "-," we currently limit our study to alphanumeric characters only as very few domains in practice use the "-" character. Previous work has also studied lexical features associated with a domain/URL.

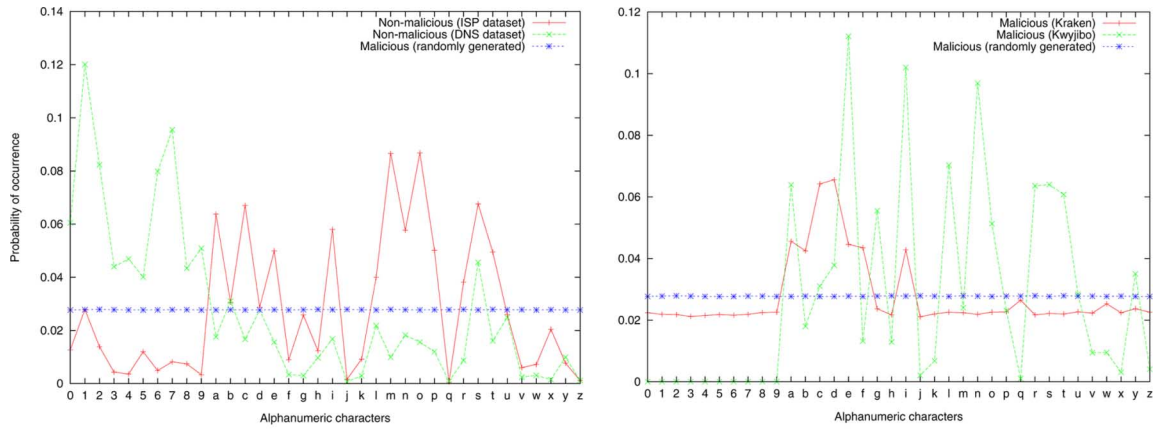

Fig. 1. Distinction between probability distributions of malicious and nonmalicious domains. (a) Nonmalicious and malicious domains. (b) Only malicious entities.

Since the K–L measure is asymmetric in nature, we use a symmetric form of the metric. The modified K–L metric is computed using the formula $D_{\mathrm{sym}}(PQ) = \frac{1}{2}(D_{\mathrm{KL}}(P\,\|\,Q) + D_{\mathrm{KL}}(Q\,\|\,P))$. For cases with singular probabilities of the random variable in the distribution, we apply suitable modifications to compute a well-defined K–L value. The modifications include ignoring computation for singular probabilities for random variables of test/malicious/nonmalicious distributions.

Given a test distribution $\underline{q}$ computed for the domain to be tested, and nonmalicious and malicious probability distribution over the alphanumerics as $\underline{g}$ and $\underline{b}$, respectively, we characterize the distribution as malicious or not via the following optimal classifier:

$$D_{\mathrm{sym}}(\underline{q}\underline{b}) - D_{\mathrm{sym}}(\underline{q}\underline{g}) \underset{b}{\overset{g}{\gtrless}} 0. \tag{1}$$

We now prove that (1) presents an optimal classifier.

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ denote M the letters of the alphabet from which the domain names are chosen (in our case, this is the English alphabet with spaces and special characters). Let $\underline{g} = [g_1, g_2, \ldots, g_M]$ and $\underline{b} = [b_1, b_2, \ldots, b_M]$ be the distribution of the letters in the good and bad domains, respectively. Let $\underline{x}$ be the actual domain name of length N that has to be classified as being good or bad. Let the letter a_i appear n_i times in $\underline{x}$ such that $\sum_i n_i = N$. Let $\underline{q} = [q_1, q_2, \ldots, q_M]$ be the distribution of the different letters in $\underline{x}$, i.e., $q_i = n_i/N$.

Under the assumption that, *a priori*, $\underline{x}$ can belong to a good or bad domain with equal probability, the classifier that minimizes the probability of error (wrong classification) is given by the maximum-likelihood classifier, which classifies $\underline{x}$ according to

$$P(\underline{x}\,|\,\underline{g}) \underset{b}{\overset{g}{\gtrless}} P(\underline{x}\,|\,\underline{b}). \tag{2}$$

Intuitively, $\underline{x}$ is classified as good if it is more likely to have resulted from the good distribution than from the bad distribution. The above classifier can be specified in terms of the likelihood ratio given by

$$\lambda(\underline{x}) = \frac{P(\underline{x}\,|\,\underline{g})}{P(\underline{x}\,|\,\underline{b})} \underset{b}{\overset{g}{\gtrless}} 1. \tag{3}$$

As we will see later, it is easier to work with an equivalent quantity $\frac{1}{N}\log\lambda(x)$. The classifier is then given according to

$$\frac{1}{N}\log\lambda(x) = \frac{1}{N}\log\frac{P(\underline{x}\,|\,\underline{g})}{P(\underline{x}\,|\,\underline{b})} \underset{b}{\overset{g}{\gtrless}} 0. \tag{4}$$

Under the assumption that the letters in $\underline{x}$ have been generated independently from the same distribution, $P(\underline{x}\,|\,\underline{g})$ is given by

$$\begin{aligned} P(\underline{x}\,|\,\underline{g}) &= \prod_{k=1}^{N} P(x_k\,|\,\underline{g}) = \prod_{i=1}^{M} P(a_i\,|\,\underline{g})^{n_i} \\ &= \prod_{i=1}^{M} g_i^{n_i} = \prod_{i=1}^{M} g_i^{q_i N}. \end{aligned} \tag{5}$$

The second equality follows by grouping all the occurrences of the letters a_i together, and recall that there are n_i such occurrences. Similarly

$$\begin{aligned} P(\underline{x}\,|\,\underline{b}) &= \prod_{k=1}^{N} P(x_k\,|\,\underline{b}) = \prod_{i=1}^{M} P(a_i\,|\,\underline{b})^{n_i} \\ &= \prod_{i=1}^{M} b_i^{n_i} = \prod_{i=1}^{M} b_i^{q_i N}. \end{aligned} \tag{6}$$

Using (5) and (6) in (4), the log-likelihood ratio can be seen to be

$$\frac{1}{N}\log\lambda(x) = \frac{1}{N}\log\frac{P(\underline{x}\,|\,\underline{g})}{P(\underline{x}\,|\,\underline{b})} = \log\frac{\prod_{i=1}^{M} g_i^{q_i}}{\prod_{i=1}^{M} b_i^{q_i}}. \tag{7}$$

Dividing the numerator and the denominator by $\prod_i q_i{}^{q_i}$, we get

$$\frac{1}{N}\log\lambda(x) = \log\frac{\prod_{i=1}^{M}\left(\frac{g_i}{q_i}\right)^{q_i}}{\prod_{i=1}^{M}\left(\frac{b_i}{q_i}\right)^{q_i}} \tag{8}$$

$$= \sum_i q_i \log\frac{g_i}{q_i} - \sum_i q_i \log\frac{b_i}{q_i} \tag{9}$$

$$= D(\underline{q}\,|\,\underline{b}) - D(\underline{q}\,|\,\underline{g}) \tag{10}$$

where $D(\underline{q} \mid \underline{b})$ is the K–L distance between the two distributions. Thus, the optimal classifier given in (4) is equivalent to

$$D(\underline{q} \mid \underline{b}) - D(\underline{q} \mid \underline{g}) \underset{b}{\overset{g}{\gtrless}} 0. \tag{11}$$

This result is intuitively pleasing since the classifier essentially computes the K–L "distance" between $\underline{q}$ and the two distributions and chooses the one that is "closer." Hence, for the test distribution $\underline{q}$ to be classified as nonmalicious, we expect $D_{\mathrm{sym}}(\underline{qg})$ to be less than $D_{\mathrm{sym}}(\underline{qb})$. However, if $D_{\mathrm{sym}}(\underline{qg})$ is greater than $D_{\mathrm{sym}}(\underline{qb})$, the distribution is classified as malicious.

1) Measuring K–L Divergence With Unigrams: The first metric we design measures the K–L divergence of unigrams by considering all domain names that belong to the same group, e.g., all domains that map to the same IP address or those that belong to the same top-level domain. We postpone discussion of groups to Section IV. Given a group of domains for which we want to establish whether they were generated algorithmically or not, we first compute the distribution of alphanumeric characters to obtain the test distribution. Next, we compute the K–L divergence with a good distribution obtained from the nonmalicious datasets (ISP or DNS crawl) and a malicious distribution obtained by modeling a botnet that generates alphanumerics uniformly. The net K–L divergence metric is defined by

$$d = D_{\mathrm{sym}}(\underline{qb}) - D_{\mathrm{sym}}(\underline{qg}). \tag{12}$$

A higher value of d for a group indicates lower maliciousness for the group.

As expected, a simple unigram-based technique may not suffice, especially to detect Kraken- or Kwyjibo-generated domains. Hence, we consider bigrams in our next metric.

2) Measuring K–L Divergence With Bigrams: A simple obfuscation technique that can be employed by algorithmically generated malicious domain names could be to generate domain names by using the same distribution of alphanumerics as commonly seen for legitimate domains. Hence, in our next metric, we consider distribution of bigrams, i.e., two consecutive characters. We argue that it would be harder for an algorithm to generate domain names that exactly preserve a bigram distribution similar to legitimate domains since the algorithm would need to consider the previous character already generated while generating the current character. The choices for the current character will hence be more restrictive than when choosing characters based on unigram distributions. Thus, the probability of test bigrams matching a nonmalicious bigram distribution, becomes smaller. From our datasets, we observe only a few bigrams occurring more often than others, reinforcing our idea. Thus, the classification based on bigram distribution may be used as an additional test for identifying malicious behavior.

Analogous to the case above, given a group of domains, we extract the set of bigrams present in it to form a bigram distribution. Note that for the set of alphanumeric characters that we consider [*a*–*z*, 0–9], the total number of bigrams possible is 36×36, i.e., 1296. Our improved hypothesis now involves validating a given test bigram distribution against the bigram distribution of nonmalicious and malicious domain labels. We use the database of nonmalicious words to determine a nonmalicious probability distribution. For a sample malicious distribution, we generate bigrams randomly. Here as well, we use K–L divergence over the bigram distribution to determine if a test distribution is malicious or legitimate [again using (12)].

3) Using Jaccard Index Between Bigrams: We present the second metric to measure the similarity between a known set of components and a test distribution, namely the *Jaccard index* measure. The metric is defined as

$$\mathrm{JI} = \frac{A \cap B}{A \cup B}$$

where A and B each represent the set of random variables. For our particular case, the set comprises bigrams that compose a domain label or a host name. Note that Jaccard index (JI) measure based on bigrams is a commonly used technique for Web search engine spell-checking [22].

The core motivation behind using the JI measure is the same as that for K–L divergence. We expect that bigrams occurring in randomized (or malicious) host names to be mostly different when compared to the set of nonmalicious bigrams. To elaborate, we construct a database of bigrams where each bigram points to a list of nonmalicious words, domain labels, or host names, as the case may be. We then determine all nonmalicious words that contain at least 75% of the bigrams present in the test word. Such a threshold helps us discard words with less similarity. However, longer test words may implicitly satisfy this criteria and may yield ambiguous JI value. As observed for test words in the DNS PTR dataset, the word sizes for 95% of nonmalicious words do not exceed 24 characters, and hence we divide all test words into units of 24 character strings.

Calculating the JI measure is best explained with an example. Considering a randomized host name such as *ickoxjsov.botnet.com*, we determine the JI value of the domain label *ickoxjsov* by first computing all bigrams (eight in this case). Next, we examine each bigram's queue of nonmalicious domain labels, and short list words with at least 75% of bigrams, i.e., six of the eight bigrams. Words satisfying this criteria may include the*qui**ick**brown**fox**jump**sov**erthelazydog* (35 bigrams). However, such a word still has a low JI value owing to the large number of bigrams in it. Therefore, the JI value is thus computed as $6/(8 + 35 - 6) = 0.16$. The low value indicates that the randomized test word does not match too well with the word from the nonmalicious bigram database.

The JI measure is thus computed for the remaining words. Note that for a group of test words, we compute the JI value only with a benign database, unlike K–L divergence, which is computed with both a benign and a malicious distribution. We compute the JI measure using the equation described above and average it for all test words belonging to a particular group being analyzed. The averaged JI value for a nonmalicious domain is expected to be higher than those for malicious groups.

The scatter plot presented in Fig. 2 indicates the clear separation obtained between nonmalicious and malicious domains. The plot represents the Jaccard measure using a case of 500 test words (details follow in Section V). We highlight the detection of botnet-based malicious domains such as *Kraken*, *MegaD*, *Pushdo*, *Srizbi*, and *Storm*. A few well-known nonmalicious domains such as *apple.com*, *cisco.com*, *stanford.edu*, *mit.edu*, and *yahoo.com* have also been indicated for comparison purposes.

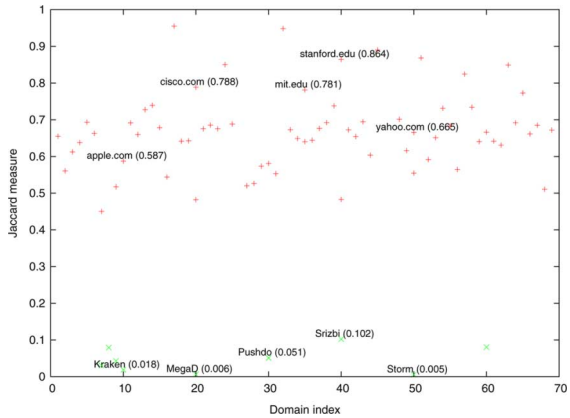

Fig. 2. Scatter plot with Jaccard Index for bigrams (500 test words). Malicious groups have lower Jaccard Index measure value than nonmalicious groups.

As observed via our experiments in Section V, the JI measure is better at determining domain-based anomalies. However, it is also computationally expensive as the database of nonmalicious bigrams needs to be maintained in the memory. Section VII examines the computational complexity of this metric.

4) Edit Distance: Note that the two metrics described earlier rely on definition of a "good" distribution (K–L divergence) or database (JI measure). Hence, we define a third metric, Edit distance, which classifies a group of domains as malicious or legitimate by only looking at the domains within the group and is hence not reliant on definition of a good database or distribution. The Edit distance between two strings represents an integral value identifying the number of transformations required to transform one string to another. It is a symmetric measure and provides a measure of intradomain entropy. The type of eligible transformations are addition, deletion, and modification. For instance, to convert the word *cat* to *dog*, the edit distance is three as it requires all three characters to be replaced. With reference to determining anomalous domains, all domain labels (or host names) that are randomized have, on an average, a higher edit distance value. We use the Levenshtein edit distance dynamic algorithm for determining anomalies [22].

IV. Grouping Domain Names

In this section, we present ways by which we group together domain names in order to compute metrics that were defined in Section III earlier.

A. Per-Domain Analysis

Note that several botnets use several second-level domain names to generate algorithmic subdomains. Hence, one way by which we group together domain names is via the second-level domain name. The intention is that if we begin seeing several algorithmically generated domain names being queried such that all of them correspond to the same second-level domain, then this may be reflective of a few favorite domains being exploited. Hence for all subdomains, e.g., *abc.examplesite.org, def.examplesite.org*, etc., that have the same second-level domain name *examplesite.org*, we compute all the metrics over the alphanumeric characters and bigrams of the corresponding domain labels. Since domain fluxing involves a botnet generating a large number of domain names, we consider only domains that contain a sufficient number of third-level domain labels, e.g., 50, 100, 200, and 500 subdomains.

B. Per-IP Analysis

As a second method of grouping, we consider all domains that are mapped to the same IP address. This would be reflective of a scenario where a botnet has registered several of the algorithmic domain names to the same IP address of a command-and-control server. Determining if an IP address is mapped to several such malicious domains is useful as such an IP address or its corresponding prefix can be quickly blacklisted in order to sever the traffic between a command-and-control server and its bots. We use the dataset from a Tier-1 ISP to determine all IP addresses that have multiple host names mapped to it. For a large number of host names representing one IP address, we explore the above described metrics, and thus identify whether the IP address is malicious or not.

C. Component Analysis

A few botnets have taken the idea of domain fluxing further and generate names that span multiple TLDs, e.g., Conficker-C generates domain names in 110 TLDs. At the same time, domain fluxing can be combined with another technique, namely "IP fluxing" [25], where each domain name is mapped to an ever changing set of IP addresses in an attempt to evade IP blacklists. Indeed, a combination of the two is even harder to detect. Hence, we propose the third method for grouping domain names into connected components.

We first construct a bipartite graph G with IP addresses on one side and domain names on the other. An edge is constructed between a domain name and an IP address if that IP address was ever returned as one of the responses in a DNS query. When multiple IP addresses are returned, we draw edges between all the returned IP addresses and the queried host name.

First, we determine the connected components of the bipartite graph G, where a connected component is defined as one that does not have any edges with any other components. Next, we compute the various metrics (K–L divergence for unigrams and bigrams, JI measure for bigrams, Edit distance) for each component by considering all the domain names within a component.

Component extraction separates the IP-domain graph into components that can be classified into the following classes:

1) *IP fan:* These have one IP address that is mapped to several domain names. Besides the case where one IP address is mapped to several algorithmic domains, there are several legitimate scenarios possible. First, this class could include domain hosting services where one IP address is used to provide hosting to several domains, e.g., Google Sites, etc. Other examples could be mail relay service where one mail server is used to provide mail relay for several MX domains. Another example could be when domain registrars provide domain parking services, i.e., someone can purchase a domain name while asking the registrar to host it temporarily.
2) *Domain fan:* These consist of one domain name connected to multiple IPs. This class will contain components belonging to the legitimate content providers such as Google, Yahoo!, etc.
3) *Many-to-many component:* These are components that have multiple IP addresses and multiple domain names,

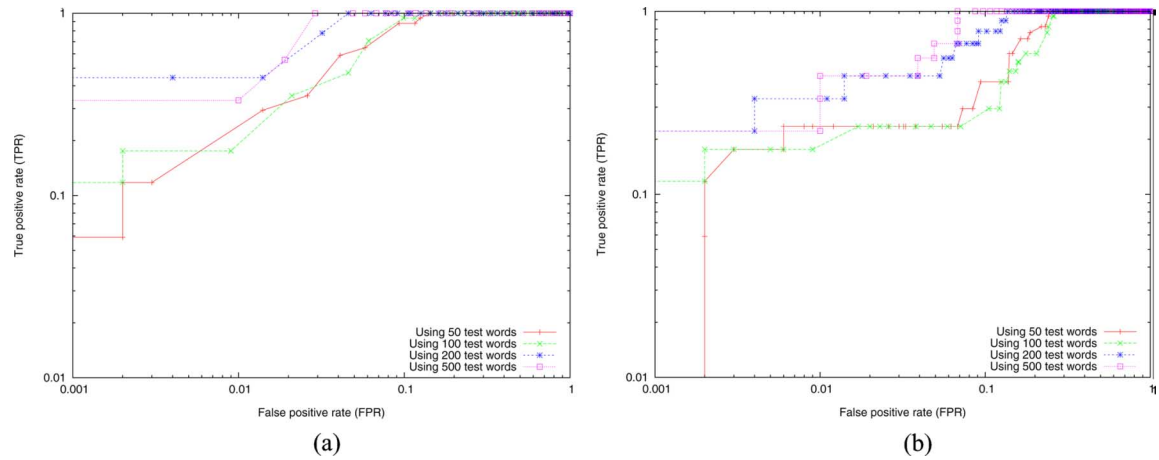

Fig. 3. ROC curves for per-domain analysis. Using 500 test words gives the best performance. (a) K–L metric with unigram distribution (per-domain). (b) K–L metric with bigram distribution (per-domain).

e.g., content distribution networks (CDNs) such as Akamai.

In Section VI, we briefly explain the classification algorithm that we use to classify test components as malicious or not.

V. Results

In this section, we present results of employing various metrics across different groups, as described in Sections III and IV. We briefly describe the dataset used for each experiment.

With all our experiments, we present the results based on the consideration of increasing number of domain labels. In general, we observe that using a larger test dataset yields better results.

A. Per-Domain Analysis

1) Dataset: The analysis here is based only on the domain labels belonging to a domain. The nonmalicious distribution $\underline{g}$ may be obtained from various sources. For our analysis, we use a database of DNS PTR records corresponding to all IPv4 addresses. The database contains 659 second-level domains with at least 50 third-level subdomains, while there are 103 second-level domains with at least 500 third-level subdomains. From the database, we extract all second-level domains that have at least 50 third-level subdomains. All third-level domain labels corresponding to such domains are used to generate the distribution $\underline{g}$. For instance, a second-level domain such as *university.edu* may have many third-level domain labels such as *physics*, *cse*, *humanities*, etc. We use all such labels that belong to trusted domains for determining $\underline{g}$.

To create a malicious base distribution $\underline{b}$, we randomly generate the same number of characters as in the nonmalicious distribution. However, for verification with our metrics, we use domain labels belonging to well-known malware-based domains identified by Botlab, and also a publicly available Webspam database, as malicious domains [1], [9]. Botlab provides us with various domains used by Kraken, Pushdo, Storm, MegaD, and Srizbi [1]. For *per-domain* analysis, the test words used are the third-level domain labels.

We present the results for all the four measures described earlier for domain-based analysis. In later sections, we will only present data from one of the measures for brevity.

2) K–L Divergence With Unigram Distribution: We measure the symmetric K–L distance metric from the test domain to the malicious/nonmalicious alphabet distributions. We classify the test domain as malicious or nonmalicious based on (1). Fig. 3(a) shows the results from our experiment presented as a Receiver Operating Characteristics (ROC) curve.

The figure shows that the different sizes of test datasets produce relatively different results. The area under the ROC is a measure of the goodness of the metric. We observe that with 200 or 500 domain labels, we cover a relatively greater area, implying that using many domain labels helps obtain accurate results. For example, using 500 labels, we obtain 100% detection rate with only 2.5% false-positive rate. We also note a detection rate of approximately 33% for a false-positive rate of 1%. Note that with a larger dataset, we indeed expect higher true positive rates for small false-positive rates, as larger samples will stabilize the evaluated metrics.

The number of domain labels required for accurate detection corresponds to the latency of accurately classifying a previously unseen domain. The results suggest that a domain-fluxing domain can be accurately characterized by the time it generates around 500 names.

3) K–L Divergence With Bigram Distribution: Fig. 3(b) presents the results of employing K–L distance metric over bigram distributions. As before, we compute the bigram distribution of our test sets and compare the K–L divergence measure computed against the good distribution and the uniform distribution (of bigrams). We again observe that using 200 or 500 domain labels does better than using smaller number of labels, with 500 labels doing the best. Experiments with 50/100 domain labels yield similar results.

We note that the performance with unigram distributions is relatively better than using bigram distributions. However, when botnets employ countermeasures to our techniques, the bigram distributions provide an additional layer of defense along with the unigram distributions.

4) Jaccard Measure of Bigrams: The Jaccard Index measure does significantly better in comparison to the previous metrics. From Fig. 4(a), it is evident that using 500 domain labels gives us a clear separation for classification of test domains (and hence an area of 1). Using 50 or 100 labels is fairly equivalent with

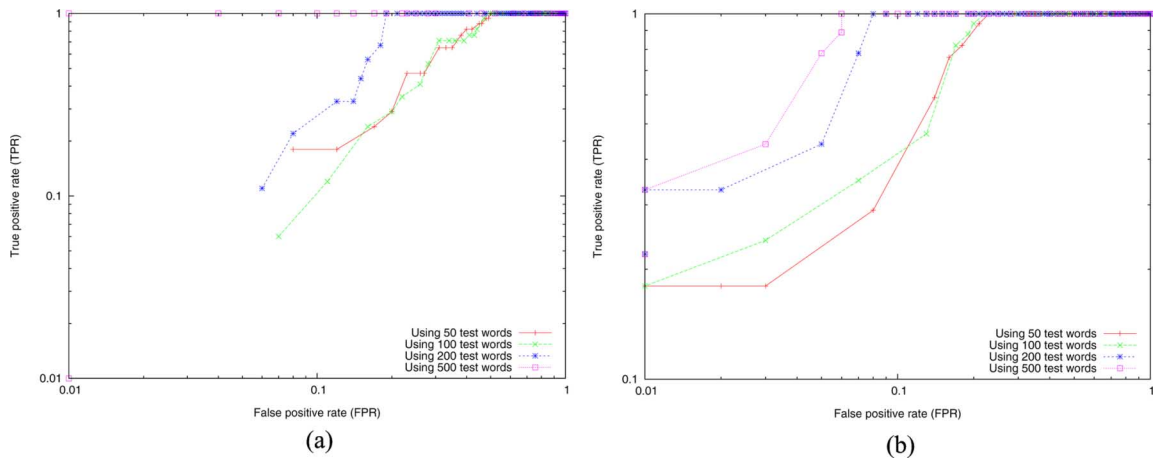

Fig. 4. ROC curves for per-domain analysis. The figure again confirms best performance with 500 test words with JI measure giving the best performance among all metrics. (a) Jaccard measure for bigrams (per-domain). (b) Edit distance (per-domain).

200 labels doing comparatively better. The JI measure produces higher false positives for smaller number of domains (50/100/200) than K–L distance measures.

5) Edit Distance of Domain Labels: Fig. 4(b) shows the performance using edit distance as the evaluation metric. The detection rate for 50/100 test words reaches 1 only for high false-positive rates, indicating that a larger test word set should be used. For 200/500 domain labels, 100% detection is achieved at false-positive rates of 5%–7%. Compare this to the detection rate of about 32% for a 1% false-positive rate.

6) Kwyjibo Domain Label Analysis: Kwyjibo is a tool to generate random words that can be used as domain labels [13]. The generated words are seemingly closer to pronounceable words of the English language, in addition to being random. Thus, many such words can be created in a short time. We anticipate that such a tool can be used by attackers to generate domain labels or domain names quickly with the aim of defeating our scheme. Therefore, we analyze Kwyjibo-based words, considering them as domain labels belonging to a particular domain.

The names generated by the Kwyjibo tool could be accurately characterized by our measures given sufficient names. Example results are presented in Fig. 5 with K–L distances over unigram distributions. From Fig. 5, we observe that verification with unigram frequency can lead to a high detection rate with low false-positive rate. Again, the performance using 500 labels is the best. We also observe a very steep rise in detection rates for all the cases. The Kwyjibo domains could be accurately characterized with false-positive rates of 6% or less.

The initial detection rate for Kwyjibo is low as compared to the per-domain analysis. This is because the presence of highly probable nonmalicious unigrams in Kwyjibo-based domains makes detection difficult at lower false-positive rates. The results with other measures (K–L distance over bigram distributions, JI, and edit distances) were similar: Kwyjibo domains could be accurately characterized at false-positive rates in the range of 10%–12%, but detection rates were nearly zero at false-positive rates of 10% or less.

7) Progressive Demarcation: The earlier results have showed that very high good detection rates can be obtained at low false-positive rates once we have 500 or more host names of a test domain. As discussed earlier, the number of host

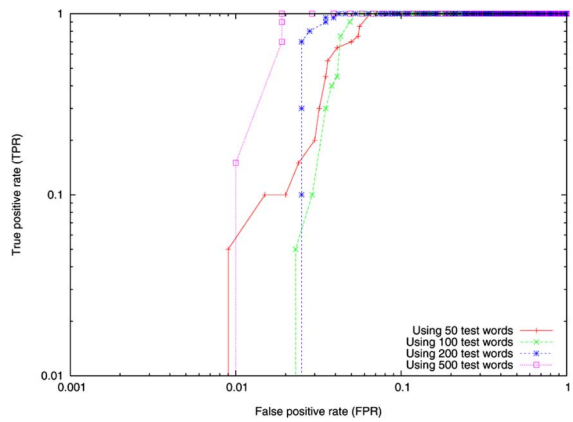

Fig. 5. ROC curve: K–L metric with unigram distribution for Kwyjibo. Kwyjibo domains are relatively difficult to detect owing to pronounceable vowels in the seemingly random words.

names required for our analysis corresponds to latency of accurately characterizing a previously unseen domain. During our experiments, not all the test domains required 500 host names for accurate characterization since the distributions were either very close to the good distribution $\underline{g}$ or bad distribution $\underline{b}$. These test domains could be characterized with a smaller latency (or smaller number of host names).

In order to reduce the latency for such domains, we tried an experiment at progressive demarcation or characterization of the test domains. Intuitively, the idea is to draw two thresholds above one there are clearly good domains, below the second threshold there are clearly bad domains, and the domains between the two thresholds require more data (or host names) for accurate characterization. These thresholds are progressively brought closer (or made tighter) as more host names become available, allowing more domains to be accurately characterized until we get 500 or more host names for each domain. The results of such an experiment using the JI measure are shown in Fig. 6.

We establish the lower bound using the formula $\mu_{\rm b} + \sigma_{\rm b}$, where $\mu_{\rm b}$ is the mean of JI values observed for bad or malicious domains and $\sigma_{\rm b}$ is the standard deviation. Similarly, the upper

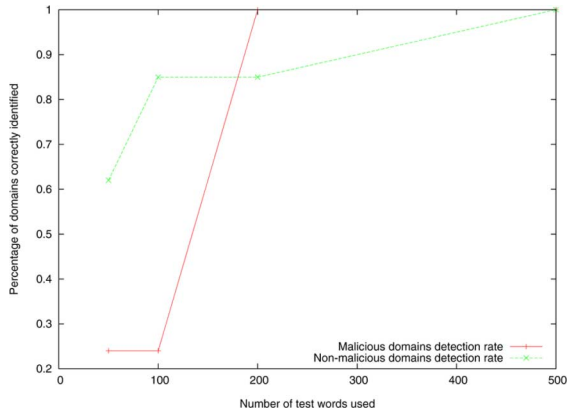

Fig. 6. Illustrating benefits of progressive demarcation with JI measure.

bound is obtained using the expression $\mu_g - \sigma_g$, where the subscript g implies good domains. Fig. 6 shows the detection rate for the considered domains. We see a monotonically increasing detection rate for both good and bad domains. It is observed that 85% of bad domains could be so characterized accurately with only 100 host names, while only about 23% of good domains can be so characterized with 100 host names. In addition, our experiments indicate that only a small percentage of domains require 200 or more host names for their characterization.

B. Per-IP Analysis

1) Dataset: Here, we present the evaluation of domain names that map to an IP address. For analyzing the per-IP group, for all host names mapping to an IP address, we use the domain labels except the TLD as the test word. For instance, for host names *physics.university.edu* and *cse.university.edu* mapping to an IP address, say *6.6.6.6*, we use *physicsuniversity* and *cseuniversity* as test words. However, we only consider IP addresses with at least 50 host names mapping to it. We found 341 such IP addresses, of which 53 were found to be malicious and 288 were considered nonmalicious. The data is obtained from DNS traces of a Tier-1 ISP in Asia.

Many host names may map to the same IP address. Such a mapping holds for botnets or other malicious entities utilizing a large set of host names mapping to fewer C&C servers. It may also be valid for legitimate Internet service such as for CDNs. We first classify the IPs obtained into two classes of malicious and nonmalicious IPs. The classification is done based on manual checking, using blacklists, or publicly available Web of Trust information [7]. We manually confirm the presence of *Conficker*-based IP addresses and domain names [28]. The ground truth thus obtained may be used to verify the accuracy of classification. Fig. 1 shows the distribution of nonmalicious test words, and the randomized distribution is generated as described previously.

We now discuss the results of per-IP analysis. The ROC curve for K–L metric shows that bigram distribution can be effective in accurately characterizing the domain names belonging to different IP addresses. We observe a very clear separation between malicious and nonmalicious IPs with 500, and even with 200, test words. With a low false-positive rate of 1%, high detection rates of 90% or more are obtained with 100 or greater number of test words.

The bigram analysis is found to perform better than unigram distributions. The per-IP bigram analysis performed better than per-domain bigram analysis. We believe that the bigrams obtained from the ISP dataset provide a comprehensive nonmalicious distribution. The second-level domain labels also assist in discarding false anomalies, and therefore provide better accuracy.

The JI measure performs well, even for a small set of test words. The area covered under the ROC curve is 1 for 200/500 test words. For the experiment with 100 test words, we achieve the detection rates of 100% with false-positive rate of about 2%. Edit distance with domains mapping to an IP results in a good performance in general. The experiments with 100 test words result in a false-positive rate of about 10% for a 100% detection rate. However, for using only 50 test words, the detection rate reaches about 80% for a false-positive rate of 20%. Thus, we conclude that for per-IP-based analysis, the JI measure performs relatively better than previous measures applied to this group. However, as highlighted in Section VII, the time complexity for computing Jaccard Index is higher.

2) Detection of Dictionary-Based DGA Botnet: In the previous section, we highlight that the Kwyjibo tool may be used by attackers as a tool for generating randomized words whose alphanumeric character composition may not be clearly distinguishable from the good references used for comparison. We, however, detect Kwyjibo-based domains efficiently. Attackers may also utilize an approach where domain names may be composed of words chosen from a language dictionary in an attempt to exhibit lower information entropy as a group of test words being analyzed. To elaborate, the more frequently used words from the dictionary comprise vowels that may thus resemble the benign distribution more closely [see Fig. 1(a)].

To verify our hypothesis, we analyze a week-long campus DNS trace collected between August 22–29, 2010. The trace is collected by capturing DNS traffic exchanged between the campus' primary DNS recursive resolver and local clients within the autonomous system. Our analysis leads us to identify a new botnet behavior where the domain names mapping to the C&C server IP address are composed of two words from the English language. Some example domain names include *elephantcode.ru, gainpizza.ru, icepipe.ru, licensemoon.ru,* and *networkjack.ru*. A similar behavior is observed with the *Storm* botnet, where the domain names used for spamming were composed of one English language word and a randomized string [1]. Interestingly, all domain names observed for this botnet belong to the *"ru"* TLD. We verify the malfeasance of the observed domain names by cross-verifying with domain reputation services [7] and hence declare the IP address as malicious if the majority of domain names mapping to it are not trustworthy.

We again apply our detection metrics to the above dataset. We compare the alphanumeric character distribution of dictionary-based DGA domain names to the randomly generated (bad) distribution and the alphanumeric distribution from the DNS dataset (good). This is essentially a per-IP analysis where we evaluate the domain names mapping to the C&C server IP addresses. From our experiments, we observe that the edit distance metric provides the lowest false-positive rate (2.56%) for a 100% detection rate. Our findings imply that characters in the test group still exhibit malicious features

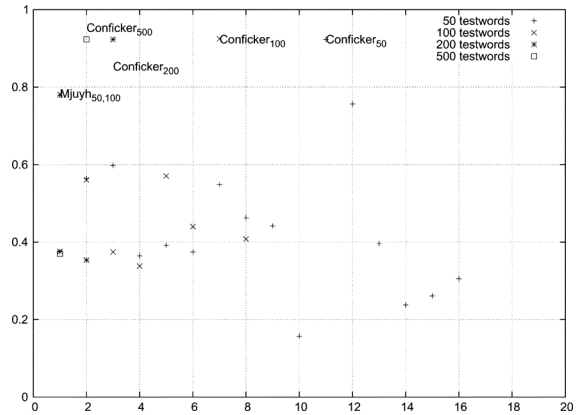

Fig. 7. Evaluation of the ED measure when applied to connected components. The ED measure is useful for identifying malicious connected components.

detectable by our metrics. We also evaluate the usefulness of combining all four metrics when detecting malicious behavior by considering false positives common between the metrics used for detecting the new botnet's domain names. Our experiments yield a relatively lower false-positive rate (2.04%) for a 100% detection rate. Thus, we infer a better performance, however, disregarding the resource complexity associated with every metric concluding that the information entropy exhibited by the botnet's modified DGA, may still be detectable. The average normalized edit distance for the pairs of domains that we observe for dictionary-based domains is in fact 0.8, only marginally lower than observed for Conficker-based domains (0.9 on an average).

C. Applying Individual Metrics to Components

We extend the application of our metrics to components extracted from the Tier-1 ISP dataset from Asia (as utilized for *Per-IP analysis*). We choose components with at least 50 domain names, and then apply each of the metrics—namely, K–L divergence with unigrams, Jaccard Index, and Edit distance—by randomly choosing an appropriate number of test words (domains). In Section VI, we highlight detection of *Mjuyh*, a botnet hitherto unknown, using a supervised learning approach. We find fourth-level domain labels of Mjuyh composed of random alphanumeric characters, thus detectable through a combination of metrics. To analyze Mjuyh based on individual metrics, however, we combine all domains for the botnet into a single connected component.

We apply the Edit distance (ED) metric for the set of four test words, as shown in Fig. 7. For components with at least 50 domains, we randomly choose 50, 100, 200, and 500 domains, discard the TLDs, and apply the metric to the obtained test words. Thus, each point in Fig. 7 represents a component with 50, 100, 200, or 500 test words. We also label the malicious components (Conficker and Mjuyh). When detecting malicious components, we note good performance with the ED measure. ED detects the *Mjuyh* botnet because, although the fourth-level domain label is composed of a restricted set of characters, their combination is still not consistent, highlighting the anomaly. Note that a relatively longer randomized domain label also contributes toward amplifying the ED metric, making detection feasible. We also note false positives with a small number of test words. For instance, the component represented by (12, 0.75) in Fig. 7 refers to a mail server providing e-mailing infrastructure for multiple domains. This behavior is analogous to the higher false positives observed for lower test words with other groups used for evaluation.

On evaluating the K–L divergence metric for various connected components (results not shown here), we observe that the distribution of alphanumeric characters for the Conficker component is distinct even with 50 test words, although there are greater false positives (as concluded through previous sections). Interestingly, we also note that the K–L metric does not help in identifying *Mjuyh*. This is because the 57-character-long fourth-level domain label present in every domain is composed of characters 0–9 and *a–f*, which suggests a hexadecimal encoding. Due to a confined character set for comparing probability distributions, the divergence thus obtained is insufficient to be conclusive.

Similarly, we apply the JI measure to different sets of test words as obtained earlier (results not shown here). We observe that *Mjuyh* has no similarity when compared to the benign database of bigrams. The malicious Conficker component also has a lower correlation. However, we also note several benign components having small (or zero) values for this metric indicating higher false positives. The analysis with JI measure generally leads us to conclude that a good performance is ensured in the case where the benign database is exhaustive, which directly impacts the computational complexity as well.

The above analysis leads us to the following question: *Is the Edit distance metric sufficient for detecting anomalies?* We caution that while the ED measure appears to distinguish *Mjuyh* and *Conficker* components, it may be vulnerable to scenarios where CDNs are involved. With CDNs, we expect that different domains hosted on the CDN servers will yield a higher ED measure, thus designating the CDN as a false anomaly. However, the JI similarity will be higher as well. Thus, the supervised learning algorithm (as described in Section VI) with weighted consideration of K–L divergence, Jaccard measure, and Edit distance, yields an accurate analysis where erroneous conclusions due to, say, the ED measure can be compensated by the remaining metrics, such as JI.

D. Summary

For a larger set of test words, the relative order of efficacy of different measures decreases from JI to Edit distance to K–L distances over bigrams and unigrams. However, interestingly, we observe the exact opposite order when using a small set of test words. For instance, with 50 test words used for the per-domain analysis, the false-positive rates at which we obtain 100% detection rates are approximately 50% (JI), 20% (ED), 25% (K–L with bigram distribution), and 15% (K–L with unigram distribution). Even though the proof for (1) indicates that K–L divergence is an optimal metric for classification, in practice it does not hold as the proof is based on the assumption that it is equally likely to draw a test distribution from a good or a bad distribution.

VI. Detection via Supervised Learning

As discussed in Section V-D, the relative merits of each measure vary depending, for instance, on the number of subdomains present in a domain being tested. In this section, we formulate detection of malicious domains (algorithmically generated)

TABLE I
DIFFERENT TYPES OF CLASSES

Type of class	# of components	# of IP addresses	# of domain names	Types of components found
Many-to-many	440	11K	35K	Legitimate services (Google, Yahoo), CDNs, Cookie tracking, Mail service, Conficker botnet
IP fans	1.6K	1.6K	44K	Domain Parking, Adult content, Blogs, small websites
Domain fans	930	8.9K	930	CDNs (Akamai), Ebay, Yahoo, Mjuyh botnet

as a supervised learning problem such that we can combine the benefits afforded by each measure while learning the relative weights of each measure during a training phase. We divide the one-day-long trace from the South Asian Tier-1 ISP into two halves such that the first one of 10 h duration is used for training. We test the learnt model on the remainder of the trace from South Asian ISP as well as over a different trace from a Tier-1 ISP in South America. In this section, we use the grouping methodology of connected components, where all "domain name, response IP address" pairs present during a time window (either during training or test phases) are grouped into connected components.

A. L1-Regularized Linear Regression

We formulate the problem of classifying a component as malicious (algorithmically generated) or legitimate in a supervised learning setting as a linear regression or classification problem. Linear regression-based classifier allows us to attach weights with each feature, making it flexible to analyze the effect of individual features. We first label all domains within the components found in the training dataset by querying against domain reputation sites such as McAfee Site Advisor [2] and Web of Trust [7], as well as by searching for the URLs on search-engines [33]. Next, we label a component as good or bad depending on a simple majority count—i.e., if more than 50% of domains in a component are classified as malicious (malware, spyware, etc.) by any of the reputation engines, then we label that component as malicious.

Define the set of features as F, which includes the following metrics computed for each component: K–L distance on unigrams, JI measure on bigrams, and Edit distance (**thus**, $|F| = 3$). Also define the set of Training examples as T, and its size in terms of number of components as $|T|$. Furthermore, define the output value for each component $y_i = 1$ if it was labeled malicious, or $= 0$ if legitimate. We model the output value y_i for any component $i \in T$ as a linear weighted sum of the values attained by each feature,where the weights are given by β_j for each feature $j \in F$: $y_i = \sum_{j \in F} \beta_j x_j + \beta_0$.

In particular, we use the LASSO, also known as L1-regularized Linear Regression [19], where an additional constraint on each feature allows us to obtain a model with lower test prediction errors than the nonregularized linear regression since some variables can be adaptively shrunk toward lower values. We use 10-fold cross validation to choose the value of the regularization parameter $\lambda \in [0\text{-}1]$ that provides the minimum training error (equation below) and then use that λ value in our tests

$$\arg\min_{\beta} \sum_{i=1}^{|T|} \left(y_i - \beta_0 - \sum_{j \in F} \beta_j x_j \right)^2 + \lambda \sum_{j \in F} |\beta_j|. \quad (13)$$

B. Results

First, note the various connected components present in the South Asian trace as classified into three classes: IP fans, Domain fans, and Many-to-many components in Table I. During the training phase, while learning the LASSO model, we mark 128 components as good (these consist of CDNs, mail service providers, large networks such as Google) and one component belonging to the Conficker botnet as malicious. For each component, we compute the features of K–L divergence, Jaccard Index measure, and Edit distance. We train the regression model using glmnet tool [19] in statistical package R, and obtain the value for the regularization parameter λ as $1e-4$, which minimizes training error during the training phase. We then test the model on the remaining portion of the one-day-long trace. In this regard, our goal is to check if our regression model can not only detect Conficker botnet, but whether it can also detect other malicious domain groups during the testing phase over the trace. During the testing stage, if a particular component is flagged as suspicious, then we check against Web of Trust [7], McAfee Site Advisor [2], as well as via Whois queries and search engines to ascertain the exact behavior of the component. Next, we explain the results of each of the classes individually.

On applying our model to the rest of the trace, 29 components (out of a total of 3 K components) are classified as malicious, and we find 27 of them to be malicious after cross-checking with external sources (Web of Trust, McAfee, etc.), while two components (99 domains) are false positives and comprise Google and domains belonging to news blogs. Note that here we use a broad definition of malicious domains as those that could be used for any nefarious purposes on the Web, i.e., we do not necessarily restrict the definition to only include botnet domain-generation algorithm. Out of the 27 components classified as malicious, one of them corresponds to the Conficker botnet, which is expected since our training incorporated features learned from Conficker. We next provide details on the remaining 26 components that were determined as malicious (see Table II).

Mjuyh Botnet: The most interesting discovery from our component analysis is that of another Botnet, which we call Mjuyh, since it uses the domain name *mjuyh.com* (see Table III). The fourth-level domain label is generated randomly and is 57 characters long. Each of the 121 domain names belonging to this bot network returns 10 different IP addresses on a DNS query for a total of 1.2 K IP addresses. Also, in some replies, there are invalid IP addresses like 0.116.157.148. All the 10 IP addresses returned for a given domain name belong to different network prefixes. Furthermore, there is no intersection in the network prefixes between the different domain names of the

TABLE II
SUMMARY OF INTERESTING NETWORKS DISCOVERED THROUGH COMPONENT ANALYSIS

Comp. type	#Comps.	#domains	#IPs
Conficker botnet	1	1.9K	19
Helldark botnet	1	28	5
Mjuyh botnet	1	121	1.2K
Misspelt Domains	5	215	17
Domain Parking	15	630	15
Adult content	4	349	13

TABLE III
DOMAIN NAMES USED BY BOTS

Type of group	Domain names
Conficker botnet	vddxnvzqjks.ws gcvwknnxz.biz joftvvtvmx.org
Mjuyh bot	935c4fe[0-9a-z]+.6.mjuyh.com c2d026e[0-9a-z]+.6.mjuyh.com
Helldark Trojan	may.helldark.biz X0R.ircdevils.net www.BALDMANPOWER.ORG

mjuyh bot. We strongly suspect that this is a case of "domain fluxing" along with "IP fast fluxing," where each bot generated a different randomized query that was resolved to a different set of IP addresses.

Helldark Trojan: We discovered a component containing five different third-level domains (a few sample domain names are shown in Table III). The component comprises 28 different domain names that were all found to be spreading multiple Trojans. One such Trojan spread by these domains is Win32/Hamweq.CW, which spreads via removable drives such as USB memory sticks. They also have an IRC-based backdoor, which may be used by a remote attacker directing the affected machine to participate in distributed denial-of-service attacks or to download and execute arbitrary files [8].

Misspelled Component: There are about five components (comprising 220 domain names) that used tricked (misspelled or slightly different spelling) names of reputed domain names. For example, these components use domain names such as uahoo.co.uk to trick users trying to visit yahoo.co.uk (since the letter "u" is next to the letter "y," they expect users to enter this domain name by mistake). Dizneyland.com is used to misdirect users trying to visit Disneyland.com (which replaces the letter "s" with letter "z"). We still consider these components as malicious since they comprise domains that exhibit unusual alphanumeric features.

Domain Parking: We found 15 components (630 domain names) that were being used for domain parking, i.e., a practice where users register for a domain name without actually using it, in which case the registrar's IP address is returned as the DNS response. In these 15 components, one belongs to GoDaddy (66 domain names), 13 of them belong to Sedo domain parking (510 domain names), and one component belongs to OpenDNS (57 domain names). Clearly, these components represent something abnormal, as there are many domains with widely disparate algorithmic features clustered together on account of the same IP address to which they are mapped.

Adult Content: We find four components that comprise 349 domains primarily used for hosting adult content sites. Clearly, this matches the well-known fact that in the world of adult-site hosting, the same set of IP addresses are used to host a vast number of domains, each of which in turn may use very different words in an attempt to drive traffic.

In addition, for comparison purposes, we used the lexical features of the domain names such as the length of the domain names, number of dots, and the length of the second-level domain name (for example, *xyz.com*) for training on the same ISP trace, instead of using the K–L divergence, JI measure, and Edit distance measures used in our study. These lexical features were found to be useful in an earlier study in identifying malicious URLs [18]. The model trained on these lexical features correctly labeled four components as malicious (Conficker bot network, three adult content components, and one component containing misspelled domain names) during the testing phase, but it also resulted in 30 components that were legitimate as being labeled incorrectly; compare this against 27 components that were correctly classified as malicious and two that were false positives on using our alphanumeric features.

We also test our model on a trace obtained from a South American-based Tier-1 ISP. This trace is about 20 h long and is collected on a smaller scale as compared to the ISP trace from Asia. The time lag between the capture of the South American Tier-1 ISP trace and the previously used ISP trace from Asia is about 15 days. We use the same training set for the prediction model as we use for the ISP trace from Asia. In the prediction stage, we successfully detect the Conficker component with no false positives. The Conficker component has 185 domain names and 10 IP addresses. Of the 10 IP addresses determined for the Conficker component of the South American trace, nine are common with the Asia ISP trace's Conficker component. We conclude that Conficker-based C&C servers have relatively large TTLs. However, out of the 185 domain names only five domains are common from this component and the component from the ISP trace from Asia. Clearly, the Conficker botnet exhibits rapid domain fluxing. Overall, this experiment shows that a training model learned in one network can successfully detect malicious domain groups when applied to a completely different network.

VII. DISCUSSION

A. Limitations

K–L divergence uses comparison to a good and a bad distribution for distinguishing a test group. However, since our good distribution is based on the publicly available dataset, the attacker may collect such a dataset and modify the DGA to generate the domain names based on the good distribution. The reference comparison to the good database also holds for Jaccard measure computation. However, the edit distance metric does not require comparison to an external reference; rather, it evaluates the intragroup entropy and, hence, is robust against the above mentioned weakness.

As highlighted through the description of the dictionary-based DGA, the botnet owner can generate domain names as a combination of English language words in an attempt to

exhibit lower entropy making it harder to detect such anomalies. However, our analysis with the group of such domains shows the entropy (expressed as normalized edit distance) is high, and hence detectable.

Another weakness of our approach is utilizing a large number of test words for accurate results. Attackers may try to overcome this constraint by timing the domain fluxing such that, say, the number of domains for an IP changes for approximately every 50 domains, without repeating any of the domains, making component analysis difficult as well. We note that this seriously constrains the botnet resources, increasing the difficulty associated with the communication between bots and the C&C server. Also, the latency improvement techniques through the use of DNS failures presented below leverages our approach. Our work utilizes at least 50 test words for analyzing groups for anomaly. However, this is not a strict requirement that the detection system has to adhere to. Through our experiments, we highlight that a larger group of test words helps in achieving higher accuracy of analysis than using smaller number of test words.

B. Latency Improvement Through Failed Domains

The latency of detection is expressed in terms of the number of test words or successful domain names required to analyze and detect a rogue server accurately. From our analysis, it is apparent that we obtain the most accurate results with 200 or more test words. Therefore, we propose supplementing the set of test words (denoted as SQ_{cncip}) with failed query domain names that occur within the vicinity of successful DNS queries. The resulting temporally correlated set may be accumulated much quicker, decreasing the latency of analysis by an order of magnitude. In addition to the temporal association, the quality of failed domains that supplement our set may be improved through entropy-based correlation.

Thus, to realize the faster accumulation of relevant domains, we compute the entropy (normalized edit distance) between a failed domain name and each of the successful DNS domain names (or test words) under analysis. A (failing) domain yielding an entropy value close to the average SEN (or the entropy of successful domains) is considered relevant. The measure of closeness is defined by choosing a bounds parameter σ, which is the standard deviation of the entropy (normalized edit distance) values observed for all pairs of test words observed for a known botnet. We note that such marginally expensive computation for identifying relevant domains through entropy correlation may help improve latency *and* accuracy.

Fig. 8 shows the gain obtained in terms of time taken to collect a set of botnet domain names. Here, we explore the effect of correlating DNS queries temporally only. The figure shows two classes of Conficker botnet IPs that we observe. Class I represents those C&C server addresses where domain fluxing yielded both successful and failed queries. However, for the C&C server in Class II, the bots issued none (or very few) failed DNS queries. In our South Asia ISP trace, with more than 50 domain names mapping to it, we find eight C&C server addresses belonging to Class I and two belonging to Class II.

Fig. 8 shows the improved latency for the average time taken by Class-I or Class-II addresses. From the figure, we observe that when failed domain names are correlated with successful

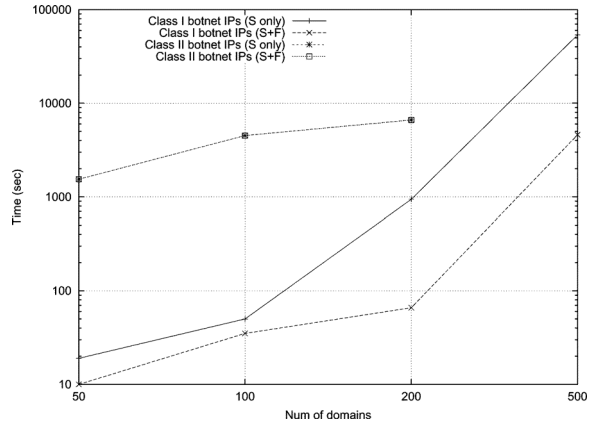

Fig. 8. Latency comparison for different number of test words. The graph shows that using failed DNS queries can reduce the initial lag required to collect the input dataset for application of anomaly detection metrics presented here.

DNS domains, the time taken to collect 50, 100, 200, or 500 domain names is considerably reduced. Especially for 500 domain names, we see a gain of more than an order of magnitude. With Class II, however, we do not observe any gain since no failures help in supplementing the set SQ_{cncip}. Thus, we infer that in context of applying statistical techniques for anomaly detection, the proposed correlation mechanism can significantly reduce the time to collect input required for analysis of Class-I C&C addresses. Note that we do not obtain 500 successful domains for the Class-II address. While speeding up the detection through methods presented here, the worst-case detection latency is the same as the original latency where only domains from SQ_{cncip} are used. We also transform botnet detection to a real-time detection approach through the speeding mechanism presented above.

C. Usefulness of Component Analysis

We note that the component analysis allowed us to detect Conficker domains that would not have been detectable with our approach when applied to domain names alone since some of these domains contained fewer than 50 names needed for accurate analysis. Similarly, some of the IP addresses in the component hosted fewer than 50 names and would not have been detected with the IP address-based analysis either. However, these domains will be included in the component analysis as long as the component has altogether more than 50 names.

Let D_{c} be the number of host names and I_{c} be the number of IP addresses in the component. If D_{d}, I_{d} are the number of host names and corresponding IP addresses detected through domain-level analysis, we define domain-level completeness ratios as $D_{\mathrm{d}}/D_{\mathrm{c}}$ and $I_{\mathrm{d}}/I_{\mathrm{c}}$. Similarly, we can define the completeness ratios for IP-based analysis as $D_{\mathrm{i}}/D_{\mathrm{c}}$ and $I_{\mathrm{i}}/I_{\mathrm{c}}$, where D_{i} and I_{i} correspond to the total number of host names and IP addresses of the Conficker botnet detected by the IP-based analysis.

For the Conficker botnet, these completeness ratios for IP-based analysis were 73.68% for IP addresses and 98.56% for host names. This implies that we are able to detect an additional 26.32% of IP addresses and a relatively small fraction of 1.44% of host names for those IP addresses. The completeness ratios for domain-based analysis were found to be 100% for

TABLE IV
SYMBOLIC NOTATION USED IN COMPUTATION COMPLEXITY ANALYSIS

Notation	
A	Alphabet size
W	Maximum word size
K	Number of test words
K'	Number of test words in a component
$S_{\underline{g}}$	Number of words in non-malicious database

TABLE V
COMPUTATIONAL COMPLEXITY

Grp.	K-L unigram	K-L bigram	JI	ED
dom.	$O(KW+A)$	$O(KW+A^2)$	$O(KW^2S_{\underline{g}})$	$O(K^2W^2)$
IP	$O(KW+A)$	$O(KW+A^2)$	$O(KW^2S_{\underline{g}})$	$O(K^2W^2)$
Com.	$O(K'W+A)$	$O(K'W+A^2)$	$O(K'W^2S_{\underline{g}})$	$O(K'^2W^2)$

IP addresses and 83.87% for the host names. Therefore, we do 16.13% better in terms of determining the host names using the per-domain analysis. This shows that the component-level analysis provides additional value in analyzing the trace for malicious domains.

D. Complexity of Various Measures

The symbols used in the computational complexity analysis are described in Table IV. Table V identifies the computational complexity for every metric and for all groups that we use. We observe that K–L metrics analyzing unigram and bigram distributions can be computed fairly efficiently. However, for the JI measure, the size of the nonmalicious database largely influences the time taken to compute the measure. A good database size results in a higher accuracy, at the cost of increased time taken for analysis. Similarly, edit distance takes longer for large word lengths and the number of test words. However, it is independent of any database, hence the space requirements are smaller.

We briefly describe how we determine the bounds as expressed in Table V for the per-domain group. For the K–L unigram analysis, since we examine every character of every test word, the complexity is bounded by KW. We then compute, for every character in the alphabet A, the divergence values. Therefore, we obtain the complexity as $O(KW + A)$. Bigram distribution-based K–L divergence is calculated similarly, except that the new alphabet size is A^2. While calculating the Jaccard index, note that the number of bigrams obtained is $O(W - 1)$. For each bigram, we examine the queues pointing to words from the nonmalicious database. Thus, for each bigram, we examine $O(WS_{\underline{g}})$ bigrams. Since we do it for K test words, we obtain $O(KW^2S_{\underline{g}})$. For every test word used while obtaining the edit distance, we examine it against the $K-1$ test words. Therefore, the total complexity is simply $O(K^2W^2)$. The expressions for *per-IP* and *per-component* groups are obtained analogously.

It is interesting to note that A is of the size 36 (0–9, *a–z* characters). K used in our analysis varies as 50/100/200/500. However, the average value for K' is higher in comparison. The DNS PTR dataset considered for *per-domain* analysis has approximately 469 000 words used for training purposes. This helps us estimate $S_{\underline{g}}$. For the ISP dataset, $S_{\underline{g}}$ is of the order of 11 522 words. Section III-C.3 gives us an estimate of W for the DNS PTR dataset as 24 characters. Based on our observation with the DNS PTR dataset, the system uses approximately 395 MB of memory for building the database in memory. The space requirements for K–L divergence measures are trivial, while the Edit distance metric inherently implies using no reference. Our analysis also reveals quick computation of each metric value (of the order of seconds) for every group. However, reading the database for JI computation takes a few (3–4) minutes.

VIII. CONCLUSION

In this paper, we propose a methodology for detecting algorithmically generated domain names as used for "domain fluxing" by several recent Botnets. We propose statistical measures such as Kullback–Leibler divergence, Jaccard index, and Levenshtein edit distance for classifying a group of domains as malicious (algorithmically generated) or not. We perform a comprehensive analysis on several datasets including a set of legitimate domain names obtained via a crawl of IPv4 address space as well as DNS traffic from a Tier-1 ISP in Asia. One of our key contributions is the relative performance characterization of each metric in different scenarios. In general, the Jaccard measure performs the best, followed by the Edit distance measure, and finally the K–L divergence. Furthermore, we show how our methodology when applied to the Tier-1 ISP's trace was able to detect Conficker as well as a botnet yet unknown and unclassified, which we call as *Mjuyh*. Analysis with campus DNS trace reveals another new botnet capable of generating domains from dictionary words. In this regard, our methodology can be used as a first alarm to indicate the presence of domain fluxing in a network, and thereafter a network security analyst can perform additional forensics to infer the exact algorithm being used to generate the domain names. As future work, we plan to generalize our metrics to work on n-grams for values of $n > 2$.

REFERENCES

[1] BotLab, "BotLab: A study in spam," 2011 [Online]. Available: http://botlab.org

[2] McAfee, "McAfee Site Advisor," 2011 [Online]. Available: http://www.siteadvisor.com

[3] P. Royal, "On Kraken and Bobax botnets," Damballa, Inc., Atlanta, GA, 2008 [Online]. Available: http://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf

[4] P. Royal, "On Kraken and Bobax botnets," Damballa, Inc., Atlanta, GA, 2008 [Online]. Available: http://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf

[5] PC Tools, "PC Tools experts crack new Kraken," 2008 [Online]. Available: http://www.pctools.com/news/view/id/202/

[6] "Twitter API still attracts hackers," Unmask Parasites blog, 2009 [Online]. Available: http://blog.unmaskparasites.com/2009/12/09/twitter-api-still-attracts-hackers/

[7] WOT, "Web of Trust," 2011 [Online]. Available: http://mywot.com

[8] Microsoft, "Win32/Hamewq," 2009 [Online]. Available: http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32/Hamweq

[9] Yahoo! Research, "Yahoo! Webspam database," [Online]. Available: http://barcelona.research.yahoo.net/webspam/datasets/uk2007/

[10] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *Proc. USENIX Security Symp.*, 2010, p. 18.

[11] A. Bratko, G. V. Cormack, B. Filipic, T. R. Lynam, and B. Zupan, "Spam filtering using statistical data compression models," *J. Mach. Learning Res.*, vol. 7, pp. 2673–2698, 2006.

[12] T. Cover and J. Thomas, *Elements of Information Theory*. Hoboken, NJ: Wiley, 2006.
[13] H. Crawford and J. Aycock, "Kwyjibo: Automatic domain name generation," in *Software Practice and Experience*. Hoboken, NJ: Wiley, 2008.
[14] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, "Measurement and classification of humans and bots in internet chat," in *Proc. 17th USENIX Security*, 2008, pp. 155–169.
[15] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. 17th USENIX Security*, 2008, pp. 139–154.
[16] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," presented at the 15th Annu. NDSS, Feb. 2008.
[17] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proc. 1st USENIX LEET*, Apr. 2008, Article no. 9.
[18] S. Savage, J. Ma, L. K. Saul, and G. Voelker, "Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs," in *Proc. ACM KDD*, Jul. 2009, pp. 1245–1254.
[19] R. Tibshirani, J. Friedman, and T. Hastie, "glmnet: Lasso and elastic-net regularized generalized linear models," Tech. rep., 2009.
[20] J. P. John, A. MoshChuck, S. D. Gribble, and A. Krishnamurthy, "Studying spamming botnets using botlab," in *Proc. NSDI*, 2009, pp. 291–306.
[21] M. Konte, N. Feamster, and J. Jung, "Dynamics of online scam hosting infrastructure," in *Proc. Passive Active Meas. Conf.*, 2009, pp. 219–228.
[22] C. D. Manning, P. Raghavan, and H. Schutze, *An Information to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[23] D. K. McGrath and M. Gupta, "Behind phishing: An examination of phisher modi operandi," in *Proc. USENIX LEET*, Apr. 2008, Article no. 4.
[24] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi, "Fluxor: Detecting and monitoring fast-flux service networks," in *Proc. Detection Intrusions Malware, Vulnerability Assess.*, 2008, pp. 186–206.
[25] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Proc. ACSAC*, Dec. 2009, pp. 311–320.
[26] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems," in *Proc. IEEE ICDM*, 2006, pp. 488–498.
[27] P. Porras, H. Saidi, and V. Yegneswaran, "Conficker C P2P protocol and implementation," SRI International, Menlo Park, CA, Tech. Rep., Sep. 2009.
[28] P. Porras, H. Saidi, and V. Yegneswaran, "Conficker C analysis," Tech. rep., Apr. 2009 [Online]. Available: http://mtc.sri.com/Conficker/addendumC
[29] P. Porras, H. Saidi, and V. Yegneswaran, "An analysis of Conficker's logic and rendezvous points," Tech. rep., Mar. 2009.
[30] J. Stewart, "Inside the storm: Protocols and encryption of the storm botnet," presented at the Black Hat Tech. Security Conf., 2008.
[31] 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 *Proc. ACM CCS*, Nov. 2009, pp. 635–647.
[32] H. L. V. Trees, *Detection, Estimation and Modulation Theory*. New York: Wiley, 2001.
[33] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, "Unconstrained endpoint profiling: Googling the internet," in *Proc. ACM SIGCOMM*, Aug. 2008, pp. 279–290.
[34] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: Signatures and characteristics," *Comput. Commun. Rev.*, vol. 38, no. 4, pp. 171–182, 2008.
[35] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, "Botgraph: Large scale spamming botnet detection," in *Proc. USENIX NSDI*, 2009, pp. 321–334.

Sandeep Yadav (S'11) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Guwahati, India, in 2007, and is currently pursuing the Ph.D. degree in computer engineering at Texas A&M University, College Station.

He has worked as an Intern with Hewlett-Packard Cloud and Security Lab, Princeton, NJ, where he focused on detecting malicious domains through graph inference techniques. His internship with Ericsson, San Jose, CA, in 2008, explored evaluating SMP kernel parameters for core network routers. Additionally, as a Research Associate with the Politecnico di Milano, Milan, Italy, in 2006, he worked on designing and developing a mobile-device-based Web services application server. His research interests include network security, with a particular focus on anomaly detection through scalable network traffic analysis.

Ashwath Kumar Krishna Reddy received the B.Tech. degree in electrical and electronics engineering from the National Institute of Technology, Surathkal, India, in 2007, and the M.S. degree in computer engineering from Texas A&M University, College Station, in 2010.

He is currently working as a Software Engineer with Microsoft Corporation, Redmond, WA. He was an Embedded Systems Engineer with Ittiam Systems, Bangalore, India, from 2007 to 2008. His research interests are in network security and embedded systems.

A. L. Narasimha Reddy (M'85–SM'98–F'10) received the B.Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, in 1985, and the M.S. and Ph.D. degrees in computer engineering from the University of Illinois at Urbana–Champaign in 1987 and 1990, respectively.

He is currently the J. W. Runyon Professor with the Department of Electrical and Computer Engineering at Texas A&M University, College Station. From 1990 to 1995, he was a Research Staff Member with the IBM Almaden Research Center, San Jose, CA. He holds five patents. His research interests are in computer networks, multimedia systems, storage systems, and computer architecture.

Prof. Narasimha Reddy is a a member of the Association for Computing Machinery (ACM). He was awarded a technical accomplishment award while with IBM. He has received an NSF Career Award in 1996. His honors include an outstanding professor award from the IEEE Student Branch at Texas A&M during 1997–1998, an outstanding faculty award from the Department of Electrical and Computer Engineering during 2003–2004, a Distinguished Achievement award for teaching from the former students association of Texas A&M University, and a citation "for one of the most influential papers from the 1st ACM Multimedia conference."

Supranamaya Ranjan (S'00–M'06) received the Master's and Ph.D. degrees in electrical and computer engineering from Rice University, Houston, TX, in 2002 and 2005, respectively.

He is a Research Scientist with MyLikes Corporation, San Francisco, CA. He has published more than 25 ACM and IEEE papers and coauthored 13 patent applications (three awarded) with USPTO in the broad area of network security. His research interests are at the cross section of computer networking and data mining. More specifically, his interests are in designing novel solutions to detect all things malicious in the Internet, including distributed denial-of-service attacks, spam, Botnets, DNS, and BGP anomalies, by using techniques from advanced statistics, graph theory, and data mining.

Dr. Ranjan has served on the Technical Program Committee for IEEE INFOCOM 2007, 2010, and 2011 and IEEE ICDCS 2008.