IEEE TRANSACTIONS ON
INFORMATION FORENSICS AND SECURITY

IEEE Xplore® Digital Library
http://ieeexplore.ieee.org/Xplore

Stealthy Domain Generation Algorithms (DGAs)

Yu Fu, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Benafsh Husain, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, Richard Brooks *Senior Member, IEEE*
Clemson University, Clemson, SC, 29634, USA

Abstract—**Botnets are groups of compromised computers that botmasters (botherders) use to launch attacks over the Internet. To avoid detection, botnets use DNS fast flux to change the mapping between IP addresses and domain names periodically. Domain Generation Algorithms (DGAs) are employed to generate a large number of domain names. Detection techniques have been proposed to identify malicious domain names generated by DGAs. Three metrics, Kullback-Leibler (KL) distance, Edit distance (ED), and Jaccard Index (JI), are used to detect botnet domains with up to 100% detection rate and 2.5% false-positive rate. In this paper, we propose two DGAs that use Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) respectively. Experiment results show that DGA detection metrics (KL, JI and ED) and detection systems (BotDigger and Pleiades) have difficulty detecting domain names generated using the proposed approaches. Game theory is used to optimize strategies for both botmasters and security personnel. Results show that, to optimize DGA detection, security personnel should use the ED detection technique with probability 0.78 and JI detection with probability 0.22, and botmasters should choose the HMM-based DGA with probability 0.67 and PCFG-based DGA with probability 0.33.**

Index Terms—**DGA, Botnet, HMM, PCFG, DGA Detection, Kullback-Leibler distance, Jaccard Index, Edit distance, intrusion detection, BotDigger, Pleiades**

I. Introduction

Botnets have been problematic for over a decade [1]. They are used to launch malicious activities including DDoS (Distributed-Denial-of-Service), spamming, identity theft, unauthorized bitcoin mining and malware distribution. In response to take-down campaigns, botmasters have developed techniques to evade detection [2]. One widely used evasion technique is DNS domain fluxing [3]. Domain names generated with Domain Generation Algorithms (DGAs) are used as the 'rendezvous' points between botmasters and bots. In this way, botmasters hide the real location of command and control (C & C) servers by changing the mapping between IP addresses and domain names frequently.

Different approaches have been proposed to detect malicious domain names [4], especially DGA-generated domain names [5], [3], [6]. Yadav et al. [3] used three metrics to detect DGA generated domain names. These metrics had up to 100% detection rate and as low as a 2.5% false-positive rate. Their assumption is that domain names generated by the same DGA are similar because they are generated using the same procedure. The metrics used to differentiate between DGA and human-generated domain names are Kullback-Leibler (KL) distance, Edit distance (ED) and Jaccard Index (JI).

TABLE I: Symbol notations

Symbols	Representations
Θ	set of symbols in a domain name (English alphabet letters and special characters)
$\mathcal{L}$	set of legitimate domain names
$\mathcal{M}$	set of malicious domain names
$D_{\text{SYM}}(P, Q)$	Symmetric Kullback-Leibler (KL) Distance between two distributions P and Q
$D_{\text{ED}}(d, d')$	Edit Distance (ED) between two domain names d and d'
$D_{\text{JI}}(d, d')$	Jaccard Index (JI) distance between two domain names d and d'
$\text{KL}(d, \mathcal{L})$	Average KL distance between domain name d and set $\mathcal{L}$
$\text{ED}(d, \mathcal{L})$	Average ED between domain name d and set $\mathcal{L}$
$\text{JI}(d, \mathcal{L})$	Average JI distance between domain name d and set $\mathcal{L}$

The innovation in this paper is that we introduce new DGAs using hidden Markov models (HMMs) and Probabilistic Context-Free Grammars (PCFGs). These are related methods. Since Yadav's method [3] has the best detection accuracy among published results, we use Yadav's detection techniques to evaluate our DGAs [7]. We benchmark different DGAs using two current DGA detection systems, BotDigger [8] and Pleiades [9]. This analysis show that our DGAs are more difficult to detect than existing ones. We model the competition between security personnel (detection techniques) and botmasters (DGAs) as a Two-Person Zero-Sum (TPZS) game. This provides a methodology for selecting optimal strategies for both sides [10], given different detection techniques and DGAs. The strategies we find should be better at detecting DGA than current approaches.

The motivation of the paper is to propose new DGAs to help researchers identify flaws in existing DGA detection techniques and systems. The new DGAs illustrate flaws in the current lexical features used to detect DGAs. More work is needed in this area. We use game theory to compare the possible detection techniques and suggest the best monitoring approach using current tools. Game theory analysis is needed for creating adaptive defenses that consider the cat and mouse nature of this conflict.

Up to now, botnet detection has been reactive, whereas botnet development has been innovative and agile. Waiting to derive new detection tools in response to criminal innovations keeps security and forensics teams at a constant disadvantage. Our new DGAs can be used to derive new pro-active tools. We use game theory to find the detection approach that should be used given current DGA detection algorithms. Our hope is that these insights can be used by security researchers to create

Yu Fu, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, Richard Brooks are in the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634, USA. (e-mail: fu2, lyu, ohambol, iozceli, jingxus, ksapra, ddu, beasle6, rrb@g.clemson.edu)

Benafsh Husain is in the School of Computing, Clemson University, Clemson, SC, 29634, USA. (e-mail: bhusain@g.clemson.edu)

Digital Object Identifier: 10.1109/TIFS.2017.2668361

better tools for detecting botnet infections.

The structure of this paper is as follows. In Section II, we review related DGA work. Section III provides background on DGA detection metrics [3] and existing DGAs. In Section IV, new DGAs using Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) are proposed. Section V presents experimental results and shows that our DGAs evade Yadav's detection techniques qualitatively and quantitatively. Section VII discusses the experimental results and proposes future work. We present our conclusions in Section VIII.

II. Related Work

To distinguish between benign and DGA-generated domain names, researchers use classification with different features. Raghuram et al. [6] built a probability model of white listed domain names and used it to distinguish between benign and algorithmically generated domain names (AGDs). Schiavoni et al. [11] developed a series of unsupervised classifiers to identify AGDs and their domain generation algorithms (DGAs). Wei-wei et al. [12] detected AGDs by analyzing morphemes in the strings of domain names. The idea was that human-generated domains and AGDs would have different features. Barabosch et al. [13] classified DGAs based on time dependency and causality. They automatically extract DGAs from malware binary. Crawford et al. [14] proposed a DGA called Kwyjibo which generates pronounceable domain names using a Markov process. Gasser [15] proposed a random word generator for pronounceable passwords. They used units that consist of one or two letters and rules defining the unit's usage to create pronounceable syllables and then concatenated generated syllables to form a word. Yadav et al. [3] detected domain fluxes using Kullback-Leibler, edit distance and Jaccard index metrics. Since Yadav's method has the best detection accuracy among published results, we use Yadav's detection techniques to evaluate our DGAs

Besides identifying malicious domain names based on specific string features, group behaviors of botnet DNS requests generated by DGAs can be used for botnet detection. Mowbray et al. [16] discovered DGAs from DNS query data by identifying the unusual distribution of string lengths of domain names. Marchal et al. [17] proposed a new technique that leveraged semantic and natural language processing tools to analyze large volumes of DNS data and identify malicious domain names. Antonakakis et al. [9] proposed a technique to detect DGA use without reverse engineering, where they found that bots from the same botnet (using the same DGA) will have similar Non-Existent Domain (NXDomain) responses. DNSRadar [18] infers unknown malicious domains from the distribution of domain cache-footprints in the network. Wang et al. [19] proposed BotMeter to assess the population distribution of DGA-based botnets by analyzing DNS query patterns. Wang et al. [20] proposed DBod, a DGA-based botnet detection scheme by analyzing the query behavior of botnet DNS traffic. All of these approaches are orthogonal to the work in this paper.

III. Background

Yadav et al. [3] used three metrics to detect DGA generated domain names, and achieved up to 100% detection rate and as low as 2.5% false-positive rate. We analyze their DGA detection algorithms to develop DGAs that evade detection. The notation in this section is listed in Table I.

A. Detection Algorithms

Yadav [3] used Kullback-Leibler Distance, Edit Distance and Jaccard Index metrics to distinguish DGA-generated domain names (from real-life botnets such as Conficker, Torpig, Kraken etc. and a research DGA called Kwyjibo [14]) from legitimate domain names. These detection schemes yield up to 100% true positive rate (TPR) and 2.5% false positive rate (FPR).

1) Kullback-Leibler Distance:
Kullback-Leibler (KL) Distance [21] measures information statistically. It is used in statistical language modeling [22] and information retrieval [23]. Since malicious domains are typically generated using algorithms and legitimate domains are generated manually, the letter distributions measured by KL distance are expected to differ.

For two discrete probability distributions $P = [P_1, P_2, ..., P_N]$ and $Q = [Q_1, Q_2, ..., Q_N]$, where $\sum_{i=1}^{N} P_i = 1$ and $\sum_{i=1}^{N} Q_i = 1$, the KL distance between P and Q is:

$$D_{\text{KL}}(P \mid\mid Q) = \sum_{i=1}^{N} P_i log \frac{P_i}{Q_i} \tag{1}$$

To make KL distance symmetric, we define $D_{\text{SYM}}(P, Q)$, where

$$\begin{aligned} D_{\text{SYM}}(P, Q) &= D_{\text{SYM}}(Q, P) \\ &= \frac{1}{2}(D_{\text{KL}}(P \mid\mid Q) + D_{\text{KL}}(Q \mid\mid P)) \end{aligned} \tag{2}$$

We calculate the probability distribution of a domain name by counting the number of occurrences of each symbol $\theta \in \Theta$ in the domain name, where Θ is the set of symbols in a domain name (English alphabet letters and special characters). For example, the probability distribution of a domain name d is given by $P_d = \{n_\theta/|d| : \forall\theta \in \Theta]\}$, where $|d|$ represents the length of d and n_θ is the number of occurrences of θ in d. For example, given $\Theta = [a, b, c, d, e]$ and $d =$ ece, we obtain the probability distribution of d as P_d = $[0, 0, 1/3, 0, 2/3]$.

To see how different a test domain d is from legitimate/malicious domain names, we calculate the average KL distance between d and a set of legitimate domain names ($\mathcal{L}$), and the average KL distance between d and a set of malicious domain names ($\mathcal{M}$). The KL distance between d and the set of legitimate domain names ($\mathcal{L}$) is:

$$\text{KL}(d, \mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\text{SYM}}(P_d, P_l) \tag{3}$$

and the KL distance between d and the set of malicious domain names ($\mathcal{M}$) is:

$$\text{KL}(d, \mathcal{M}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\text{SYM}}(P_d, P_m) \tag{4}$$

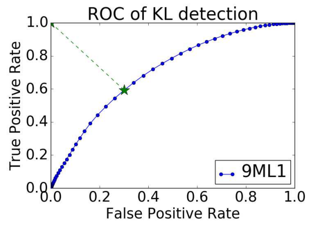


Fig. 1: An example ROC curve

The difference between $\mathrm{KL}(d, \mathcal{L})$ and $\mathrm{KL}(d, \mathcal{M})$ is:

$$\Delta_{\mathrm{KL}} = \mathrm{KL}(d, \mathcal{L}) - \mathrm{KL}(d, \mathcal{M}) \tag{5}$$

To determine if d is legitimate or malicious, we compare Δ_{KL} with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\mathrm{KL}} \geq \alpha, & d \text{ is malicious;} \\ \text{if } \Delta_{\mathrm{KL}} < \alpha, & d \text{ is legitimate.} \end{cases}$$

We draw Receiver Operating Characteristic (ROC) curves [24] and vary the threshold α (from 0 to ∞) to find the optimal threshold. An optimal threshold is the α associated with the point on the ROC curve that is closest to (0,1)[1]. When the optimal threshold is used, it gives the optimal balance between TPR and FPR. An example ROC curve of KL detection is in Figure 1, where the star point is the optimal threshold, and should be used to obtain the optimal detection performance.

2) *Edit Distance:*

Edit Distance (ED or Levenshtein Distance) [25] inspects pairwise string alignment. It has been widely used in spell checking, plagiarism detection [26], and pronunciation measurement [27].

Edit distance gives the minimum number of single-character edits (insertion, deletion, and substitution) to transform one word to another. The computation of edit distance between two domain names d and d' is a dynamic programming problem [28], and can be broken into a collection of subproblems to calculate $lev(i, j)$, where $i = 1, \cdots, |d|$ and $j = 1, \cdots, |d'|$. Assuming $lev(i, 0) = i$ and $lev(0, j) = j$, we start with $i = 1, j = 1$ and alternately increment them by 1 each time until $i = |d|, j = |d'|$. Edit distance between domain names d and d' is defined as $D_{\mathrm{ED}}(d, d') = lev(|d|, |d'|)$. For $i = 1, \cdots, |d|$ and $j = 1, \cdots, |d'|$,

$$lev(i, j) = \min \begin{cases} lev(i-1, j) + 1 \\ lev(i, j-1) + 1 \\ lev(i-1, j-1) + \begin{cases} 2, & \text{if } d(i) \neq d'(j) \\ 0, & \text{if } d(i) = d'(j) \end{cases} \end{cases} \tag{6}$$

where, $d(i)$ is the i^{th} character in d_i and $d'(j)$ is the j^{th} character in d'. Similar to KL distance, the difference between the edit distance from d to legitimate set $\mathcal{L}$ and the distance from d to malicious set $\mathcal{M}$ is given by:

$$\begin{aligned} \Delta_{\mathrm{ED}} &= \mathrm{ED}(d, \mathcal{L}) - \mathrm{ED}(d, \mathcal{M}) \\ &= \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\mathrm{ED}}(d, l) - \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\mathrm{ED}}(d, m) \end{aligned} \tag{7}$$

[1]Note that point (0,1) in ROC curve refers to 0 false positive rate and 100% true positive rate.

We compare Δ_{ED} with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\mathrm{ED}} \geq \alpha, & d \text{ is malicious;} \\ \text{if } \Delta_{\mathrm{ED}} < \alpha, & d \text{ is legitimate.} \end{cases}$$

3) *Jaccard Index:*

Jaccard Index (JI) [29] measures the similarity between two distributions, and is extended to spell checking and set comparison [30].

JI distance between domain name d and d' is defined as:

$$D_{\mathrm{JI}}(d, d') = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{8}$$

where A and B are the sets of unique characters in domain names d and d' respectively. The assumption is that the letter distribution is different in legitimate and DGA-generated domain names. Given a test domain d, we calculate the difference between $\mathrm{JI}(d, \mathcal{L})$ and $\mathrm{JI}(d, \mathcal{M})$ with equation (9).

$$\begin{aligned} \Delta_{\mathrm{JI}} &= \mathrm{JI}(d, \mathcal{L}) - \mathrm{JI}(d, \mathcal{M}) \\ &= \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\mathrm{JI}}(d, l) - \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\mathrm{JI}}(d, m) \end{aligned} \tag{9}$$

and compare it with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\mathrm{JI}} \geq \alpha, & d \text{ is malicious;} \\ \text{if } \Delta_{\mathrm{JI}} < \alpha, & d \text{ is legitimate.} \end{cases}$$

B. *Domain Generation Algorithms (DGAs)*

A DGA can generate a large number of domains used as ‘rendezvous points’ between botmasters and bots. By using a DGA, botmasters don’t need to hard-code the IP addresses of C&C servers. Instead, they put the DGA in the bots and register a few of the domains ahead of the communication. The bot will query possible domain names until one query is resolved to a valid IP address. The advantage of using a DGA is not only obscuring the locations of the C&C nodes, but also the agility of the approach making it impossible for vendors to block all possible domain names while it is inexpensive for the botmasters to register a few domain names.

The DGA generates a list of domain names with pre-shared secrets between botmasters and bots. One secret is the DGA seed [31], which can be a numeric constant, current date/time, or online information. Plohmann et al. [31] analyzed and characterized 43 DGAs based on whether the seed is time independent or deterministic. They observed DGAs using time-independent and deterministic (TID) seeds, time-dependent and deterministic (TDD) seeds, and time-dependent and non-deterministic (TDN) seeds. The other type of secret is salt, which can add complexity to DGAs and complicate the detection [32]. It integrates another random string into the process to defend against dictionary attacks. Salt can be either a constant (new Gameover Zeus or GoZ DGA [33]) or variable string (Tinba DGA [34]).

The fight against DGA-based botnets never stops. According to Damballa [35], C&C servers are becoming more agile by using DGAs as backup strategies. One common approach [36] for the security personnel to take down DGA-based botnets is to sinkhole the DGA-generated domain names to prevent the domain resolution. However, this solution has some

TABLE II: Summary of existing DGA techniques and examples

DGA	DGA techniques	Example domain names
Zeus	MD5 of the year, month, day, and a sequence number between 0 and 999	krhafeobleyhiy-trwuduzlbucutwt, vsmfcabubenvib-wolvgilhirvmz
Conficker	GMT date as the seed of the random number generator	fabdpdri, sfqzqigzs, whakxpvb
Kraken	a random string of 6 to 11 characters	rhxqccwdhwg, huwoyvagozu, gmkxtm
Srizbi	date transformation using XOR operations	wqpyygsq, tqdaourf, aqforugp
Torpig	current date and number 8 as the seed of the random number generator	16ah4a9ax0apra, 12ah4a6abx5apra, 3ah0a16ax0apra
Kwyjibo	Markov process on English syllables	overloadable, refillingman, multible

limitations: (a) it only works for DGAs with known seed/salt; (b) it requires extensive coordination and cooperation of the top-level domains (TLD) that the DGA uses, otherwise the botmasters can easily switch the TLD to evade the sinkholing. At the same time, new DGA-based botnets will emerge, such as Mad Max DGA (2016) [37], Sphinx DGA (2015) [38], GozNym DGA (2015) [39] etc..

Within the scope of this work, to evaluate the performance of our DGAs, we compare them with domain names generated by the best known DGAs, which include DGAs used by 5 live botnets (Zeus, Conficker, Kraken, Srizbi, and Torpig). We also use a research DGA (Kwyjibo) [14], because a) Kwyjibo is one of the sophisticated DGAs, which generates words that are pronounceable yet not in the English dictionary; and b) Yadav [3] evaluated Kwyjibo using three detection metrics (KL, ED and JI). Similar to Yadav's method, we compare our DGAs with Kwyjibo. Each DGA generates one set of malicious domain names. We briefly describe each DGA. All DGA techniques and example DGA-generated domain names are in Table II.

1) Zeus DGA: The Zeus botnet (2007) was used to steal banking information by keystroke logging and screen grabbing. We obtained pseudo code for generating domain names from the 2011 peer-to-peer variant of Zeus [40]. Domain name generation starts by taking the MD5 hash of the year, month, day, and a sequence number between 0 and 999. Then the right-most 5 bits of the MD5 hash is added to the hexadecimal value of letter 'a' to convert numbers to alphabetic characters. All generated alphabetic characters are concatenated to form a domain name.

2) Conficker DGA: The Conficker botnet (2008) exploited vulnerabilities of Windows OS to launch dictionary attacks. A survey [41] in 2012 found that it had infected about 7 to 15 million hosts. A research group from the University of Bonn, Germany reverse engineered DGAs of three Conficker botnet variants [42] and published domain datasets generated by the DGAs [43].

3) Kraken DGA: Used for spamming, the Kraken botnet (2008) [44] used algorithmically generated domain names of Dynamic DNS (DDNS) providers to locate and communicate with the botnet C & C servers. To initiate communication, the malware generates a string of six to eleven characters, used as a subdomain of one of the DDNS domains ($yi.org$, $dyndns.org$, $afraid.org$, or $dynserv.com$) to form a candidate C & C domain name (e.g., $bdubefoeug.yi.org$). We did not implement the Kraken DGA, but obtained a list of 1000 bot domains [44].

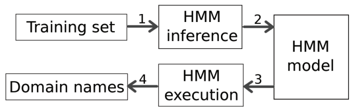


Fig. 2: Flowchart of HMM-based DGA

4) Srizbi DGA: The Srizbi botnet (2008) infected around 450,000 machines to send spam [45]. Given a date, the day, month, and year are split, transformed and concatenated using XOR operations. Four domain names are generated for each date throughout the year. We generated 10000 domain names using the Srizbi DGA [46].

5) Torpig DGA: Torpig (2008) stole about 500,000 online bank accounts and credit/debit cards. We used Torpig pseudo code from [47] to generate domain names. The DGA was seeded using the current date and a constant number (8). The algorithm computed the domain name either weekly using the current week and year, or daily using the current day. We generated malicious domain names using the daily domain name generation approach.

6) Kwyjibo DGA: The kwyjibo DGA [14] is a random word generator that guarantees the word to be short and pronounceable. It breaks dictionary words into syllables using a hyphenation algorithm [48]. It then generates random domain names based on a Markov process by making each syllable a state, and limits the length of output domain names to between two and four syllables. We use this technique [48] to obtain syllables from dictionary words for input to our DGAs. But unlike Kwyjibo, our DGAs learn from the statistical patterns from legitimate domain names and generate domain names with alphabets, numbers and '-'.

IV. Proposed Domain Generation Algorithms

A. Hidden Markov Model (HMM)-based DGA

Our first DGA uses HMMs inferred from legitimate domain names to generate new domain names, which should have the same statistical properties as legitimate domains. The zero-knowledge HMM inference algorithm from [49], [50], [51], [52] is used to derive the HMMs. The flowchart of the HMM-based DGA is shown in Figure 2.

The training sets used to infer HMMs are in Table III. HMMs are generated using two parameters, L and dictionary:

1) L is the maximum number of history symbols used to calculate state transition probabilities in a HMM. Larger L produces an HMM with less conditional entropy between states. Take 'abc' as an example input string. If $L = 1$, we calculate the transition probability $P(a|a)$,

TABLE III: HMM dataset details

Name	Dictionary	L	Contain letters	Contain numbers	Pronou-nceable	Example Output
DNL1	English dictionary	1	yes	no	maybe	stgaarls, lohirak
DNL2		2				leaptoin, coinicaf
DNL3		3				ersonsto, bolue
DNL4		4				lyes, eftanne
9ML1	9 million IPv4 domains	1	yes	yes	less likely	pore9nn10, 15fapn-43f
500KL1	Randomly selected 500k from 9ML1	1	yes	yes	less likely	b10-82025, 1-14455a
500KL2		2	yes	yes	less likely	t5154x150, oa-1840-15
500KL3		3	yes	yes	less likely	whp01075, ota-150-pc

$P(a|b)$, $P(a|c)$, *etc.*. If $L = 2$, we calculate the transition probability $P(a|aa)$, $P(a|ab)$, $P(a|ac)$, $P(a|ba)$, $P(a|bb)$, $P(a|bc)$, $P(a|ca)$, $P(a|cb)$, $P(a|cc)$, *etc.*. For our DGA, we consider transition probabilities for symbol histories of up to four letters ($L = 4$).

2) Dictionary is the training set used to build the HMM, which is a word list containing legitimate domain names. To compare the performance of different dictionaries, we build 8 HMMs using different dictionaries and different values of L, and discuss their performance in Section V. In our experiment, three dictionaries are used:
   - English word list [53];
   - 9 million domain names from IPv4 space (Appendix A);
   - 500,000 entries randomly selected from the 9 million IPv4 subdomains. The complexity of inferring the HMM is $O(k^{L+1}) + O(N)$ [54], [49], where k is the number of symbols, L is the maximum substring length considered, and N is the size of the input symbol sequence. As the dictionary size increases, HMM generation becomes intractable. So we use the subset of all IPv4 domains to build the HMM.

Given the created HMM, we generate domain names by randomly choosing a start state in the model. Since each transition in the HMM is associated with a symbol, a transition is taken from the current state and the corresponding symbol is selected. If there are more than one possible transitions out of the current state, the transition is chosen randomly. In our experiment, each domain name is restricted to 3 to 10 letters[2] to be consistent with the length of real IPv4 domain names.

B. Probabilistic Context-Free Grammar (PCFG)-based DGA

The second DGA proposed in this work uses PCFG to generate domain names. A context-free grammar (CFG) [55] is a set of recursive production rules used to generate or recognize string patterns. It is represented as a tuple $G = (N, \sum, R, S)$ in Chomsky normal form where:

[2]An illustrative example can be found https://clemson.box.com/s/uhr6p0kemq2d4u83gtufcn55pgl92pyp.

Terminals	a, c
Non-terminals	S, B
Productions	$S \to BB$ $B \to aBc\,(p = 0.25)$ $B \to \varnothing\,(p = 0.75)$

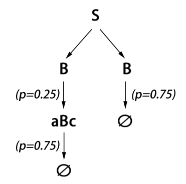


Fig. 3: An example PCFG grammar and parse tree

TABLE IV: PCFG source lists

Source List	Source	Contain letters	Contain numbers	Pronoun-ceable	Example Output
pcfg_dict	English syllables	yes	no	yes	guisedswerv, daveFlorid
pcfg_dict_num	English syllables +number	yes	yes	yes	89chute, chooser3119
pcfg_ipv4	ipv4 syllables	yes	no	maybe	zenaidae, toowoom
pcfg_ipv4_num	ipv4 syllables +number	yes	yes	maybe	hrab-324, 1245ickb

1) N is a set of nonterminal symbols, which are the placeholders for terminal symbols;
2) $\sum$ is a set of terminal symbols, which are characters from the alphabets that the strings are made of;
3) R is a set of production rules in the form of $A \to \alpha$, where $A \in N$ and $\alpha \in (\sum \cup N)$, which describes the recursive pattern of the grammar;
4) S is the start symbol, and $S \in N$, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

A Probabilistic Context-Free Grammar (PCFG), denoted by (G, θ), uses a probability vector θ to assign a probability to each production rule in R [56]. PCFGs are a natural stochastic model for recursive processes [57], [50]. A PCFG can also be described as a random branching process [58], [59], and its asymptotic behavior can be characterized by its branching rate. The branching rate refers to the geometric growth rate of the sentence derivation. Since domain names must have finite length, the branching rate has to be less than 1 [58], [59].

The grammar we use is defined in Figure 3. This is a simple grammar, derived from parentheses matching, which is given as an example process. We use it as a simple example, because it cannot be recognized using state machines. The grammar can be visualized with a parse tree, which is a finite tree regulated by grammar rules. An example parse tree is illustrated in Figure 3, in which terminal a and c are two syllables randomly picked from two mutually exclusive lists of syllables A and C. A and C are random disjoint subsets of a source list (Table IV[3]), where $A \cup C = Source\ list$ and $A \cap C = \varnothing$. For example, if a = 'cr' and b = 'out', the output of the example parse tree in Figure 3 would be '$crout$'. Since different source lists produce different domain names, we use 4 different source lists in our experiment:

[3]Pronounceable refers to if the generated domain is easy to be pronounced by native English speakers.

- pcfg_dict: A syllable list generated from an English dictionary [53] using hyphenation [48],
- pcfg_dict_num: The syllable list in pcfg_dict plus a number list (the number list contains number from 0 to 2000, where each repeated 20 times),
- pcfg_ipv4: A syllable list parsed from IPv4 domain names (see Appendix A), and
- pcfg_ipv4_num: The syllable list in pcfg_ipv4 plus a non-alphabetic list (see Appendix B).

V. Experiment Results and Analysis

We proposed 2 DGAs (HMM-based and PCFG-based) in this work. To figure out the parameters in HMM/PCFG-based DGAs that can generate 'best' domain names ('best' refers to domain names that have the best anti-detection performance), we have 2 tests:

- Test 1: Compare 8 HMMs generated with different training sets (Table II) and find the HMM whose domain names are most difficult to detect;
- Test 2: Compare 4 PCFGs generated with different training sets (Table IV) and find the PCFG whose domain names are most difficult to detect;

With the best HMM-based (from Test 1) and the best PCFG-based DGAs (from Test 2), we compare them with DGAs used by real-world botnets (Zeus, Conficker, Kraken, Srizbi, and Torpig) and one research DGA (Kwyjibo) as:

- Test 3: Compare the best HMM-based DGA, the best PCFG-based DGA, Kwyjibo with 5 real-life DGAs.

A. Experiment setup

For each DGA to be tested, we generate sets of domain names (test dataset $\mathcal{T}$) using this DGA. Then we compare $\mathcal{T}$ with set of legitimate domain names (legitimate dataset $\mathcal{L}$) collected using the method in Appendix A and sets of malicious domain names (malicious dataset $\mathcal{M}$) generated by the same DGA. The goal of DGA is to generate domain names that are not distinguishable from $\mathcal{L}$ and $\mathcal{M}$. The reason why we compare the DGA against itself ($\mathcal{T}$ and $\mathcal{M}$ are generated by the same DGA) is that it corresponds to the practice of security companies, where they compare each test DGA against their botnet DGA database. If there is similarity between the test DGA and existing DGAs, an alarm will be raised. So the domain names generated by the ideal DGA will not have similar character attributes, which are not the case with existing botnet DGAs.

Each test dataset contains 500 domain names. We use ROC curves to illustrate the performance of DGAs. For each threshold in ROC curves, we repeat the experiment 1000 times using different test datasets and calculate the average TPR and FPR. Each ROC curve is obtained with 100 thresholds evenly distributed from 0 to ∞ (maximum distance we observe from the data).

B. Experiment results

1) Test 1 - Compare different HMM-based DGAs:

Figure 4 shows the ROC curves of HMM-based DGAs under three detection techniques introduced in Section III-A. Note that botmaster wants to maximize the distance from the ROC curve and point $(0, 1)$; or equivalently, to make the ROC curve as close to $y = x$ as possible. This means the detection cannot give a reasonable TPR while maintaining a close to 0 FPR. So the optimal DGA, or the DGA of best anti-detection performance is the one whose ROC curve is closest to $y = x$.

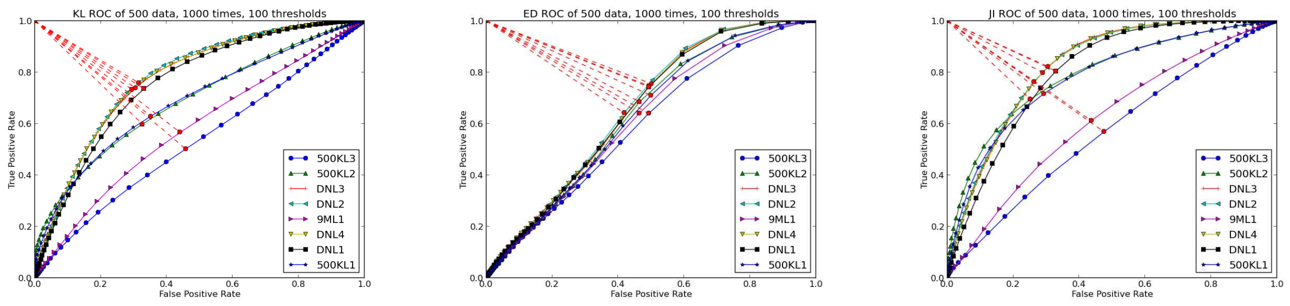
- KL (Figure 4(a)): Based on the distance from the ROC curve to point $(0, 1)$, HMM-based DGAs in descending order of performance are 500KL3 $>$ 9ML1 $>$ 500KL1 $=$ 500KL2 $>$ DNL1 $=$ DNL2 $=$ DNL3 $=$ DNL4 (using 95% confidence intervals). There are two reasons why 500KL3 outperforms the other HMM-based DGAs: first, the training set used to infer the HMM consists of real IPv4 domains rather than English words; and second, larger L produces an HMM with less conditional entropy between states [60], which better represents the statistical patterns in legitimate domain names. We believe the HMM inferred using all 9 million subdomains with $L = 3$ will exhibit even better anti-detection performance. However, due to the intractable computational complexity, we only provide the results of $L = 1$ (9ML1).
- ED (Figure 4(b)): All ROC curves of ED detection are close to each other and to diagonal $y = x$. So in general, ED is not an effective method for detecting domains generated with HMM-based DGAs.
- JI (Figure 4(c)): When JI detection is applied, the performance is 500KL3 $>$ 9ML1 $>$ 500KL1 $=$ 500KL2 $=$ DNL1 $=$ DNL2 $=$ DNL3 $=$ DNL4 (using 95% confidence intervals). This is similar to the results with KL detection.

Based on the performance, 500kL3 is the best HMM-based DGA because its ROC curve is always closest to y=x under all three detection techniques.

2) Test 2 - Compare different PCFG-based DGAs:

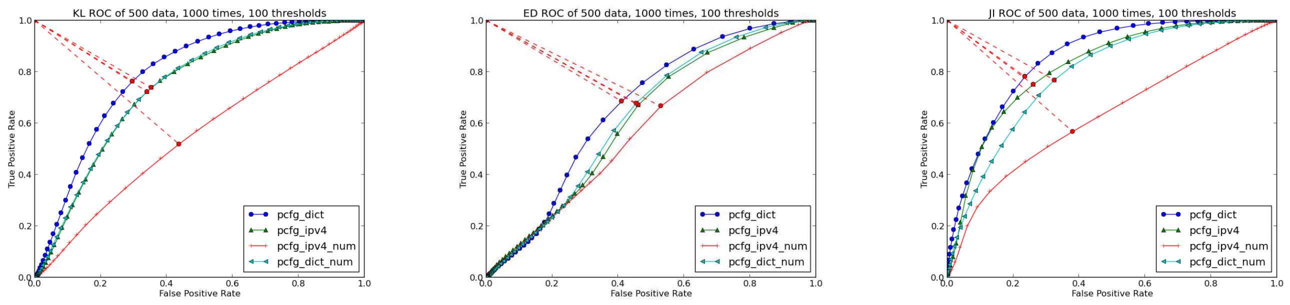
Figure 5 shows the ROC curves of 4 different PCFG-based DGAs under three detection routines:

- KL (Figure 5(a)): The performance is pcfg_ipv4_num $>$ pcfg_dict_num $=$ pcfg_ipv4 $>$ pcfg_dict (using 95% confidence intervals). The reasons why pcfg_ipv4_num outperforms the rest are: (1) the PCFG is trained using real IPv4 domains, and thus generates domains more similar to legitimate ones; and (2) by including numbers, diversity of generated domain is increased. We interpret the advantages of pcfg_ipv4_num over pcfg_dict_num as patterns in IPv4 are harder to detect than patterns in English dictionary words. That is also to say, the distance between IPv4 domain names and malicious domain names is generally smaller than that between English dictionary words and malicious domain names.
- ED (Figure 5(b)): Four PCFG-based DGAs show similar performance under ED detection, since all ROC curves are close to $y = x$. It indicates ED is not a reliable measurement of differences between legitimate domains and PCFG-generated domains. This is also consistent with the results of HMM-based DGAs under ED detection.
- JI (Figure 5(c)): The performance is pcfg_ipv4_num $>$



(a) (b) (c)

Fig. 4: ROC curves of HMM-based DGAs under (a) KL detection, (b) ED detection, and (c) JI detection



(a) (b) (c)

Fig. 5: ROC curves of PCFG-based DGAs under (a) KL detection, (b) ED detection, and (c) JI detection

pcfg_dict_num > pcfg_ipv4 > pcfg_dict. It is similar to the performance with KL detection.

Based on the performance, pcfg_ipv4_num is the best PCFG-based DGA because its ROC curve is closest to y=x under all three detection techniques.

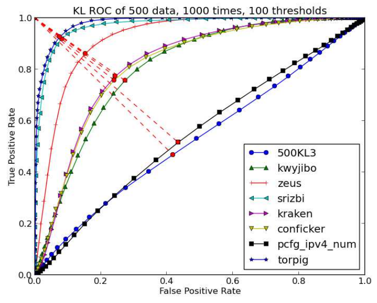
3) Test 3 - Compare HMM-based, PCFG-based with Kwyjibo and 5 real-life DGAs:
We compared 500KL3 (the best HMM-based DGA from Test 1), pcfg_ipv4_num (the best PCFG-based DGA from Test 2) with Kwyjibo and 5 real-life DGAs. The results are in Figure 6.

- KL (Figure 6 (a)): The performance is: 500KL3 (HMM-based DGA) > pcfg_ipv4_num (PCFG-based DGA) >> Kwyjibo = Conficker = Kraken >> Zeus >> Srizbi = Torpig (using 95% confidence intervals). The proposed DGAs (500KL3 and pcfg_ipv4_num) are significantly better than the other DGAs, and their ROC curves are very close to $y = x$. Therefore, both proposed DGAs can effectively evade KL detection.
- ED (Figure 6 (b)): The performance is: pcfg_ipv4_num (PCFG-based DGA) > 500KL3 (HMM-based DGA) >> Kraken = Conficker > Kwyjibo >> Srizbi >> Zeus = Torpig (using 95% confidence intervals). Although the advantage of our DGAs are not so obvious as when KL detection is applied, they still outperform all the other DGAs. The ROC curves of our DGAs are very close to $y = x$, which indicates our approaches can effectively evade ED detection. Notice that ED is not effective in detecting HMM-based DGAs with different parameters (Figure 4(b)). It is the same case with detecting PCFG-based DGAs (Figure 5(b)). But ED is effective in differentiating HMM-based (hmm_500KL3), PCFG-based (pcfg_ipv4_num) and existing botnet DGAs. We suspect that some botnet DGAs use similar domain names (for example, Torpig uses '0ah2a2abx3apra' and '4ah0a5ax19apra'), which are close to each other in terms of ED distance. However, our DGAs don't suffer the limitations.
- JI (Figure 6 (c)): The performance is: 500KL3 (HMM-based DGA) > pcfg_ipv4_num (PCFG-based DGA) >> Conficker = Kraken = Kwyjibo >> Zeus = Srizbi = Torpig (using 95% confidence intervals). The advantages of HMM-based and PCFG-based DGAs over the others are obvious.

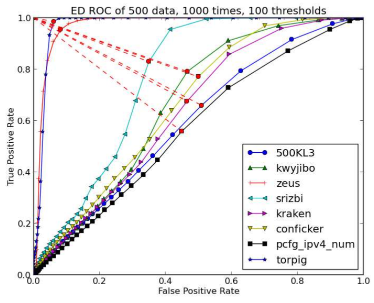
C. Result Analysis

1) Qualitative Analysis:
Based on the experiment results, it is safe to say that both HMM-based and PCFG-based DGAs thwart all three detection routines. They outperform Kwyjibo and 5 real-life DGAs. In KL/ED/JI detection, the advantages of our DGAs over Kwyjibo and 5 real-life DGAs are obvious.

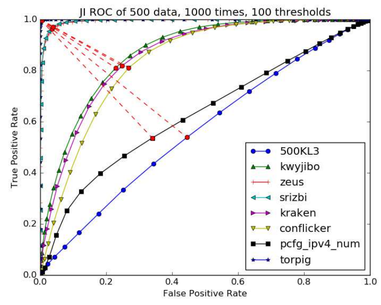
2) Quantitative Analysis:
Further analysis was carried out by modeling the competition between DGAs and detection techniques as a Two-Person Zero-Sum (TPZS) game. Although both detection techniques and DGAs in reality are not limited to those in discussion, our analysis provides a methodology for selecting the optimal



(a)



(b)



(c)

Fig. 6: ROC curve comparison of HMM-based, PCFG-based, Kwyjibo and 5 real-life DGAs under (a) KL detection (b) ED detection (c) JI detection

TABLE V: The payoff matrix of player II using HMM-based DGA

		player I: security personnel (minimizer)			
		KL	ED	JI	min
player II: botmaster (maximizer)	500KL3	0.68	0.61	0.64	0.61
	500KL2	0.52	0.56	0.40	0.40
	DNL3	0.40	0.56	0.35	0.35
	DNL2	0.40	0.55	0.35	0.35
	9ML1	0.62	0.59	0.58	0.58
	DNL4	0.40	0.56	0.35	0.35
	DNL1	0.43	0.56	0.38	0.38
	500KL1	0.51	0.58	0.41	0.41
	max	0.68	0.61	0.64	

TABLE VI: The payoff matrix of player II using PCFG-based DGA

		player I: security personnel (minimizer)			
		KL	ED	JI	min
player II: botmaster (maximizer)	pcfg_dict	0.38	0.52	0.32	0.32
	pcfg_ipv4	0.44	0.57	0.36	0.36
	pcfg_ipv4_num	0.65	0.63	0.58	0.58
	pcfg_dict_num	0.44	0.56	0.40	0.40
	max	0.65	0.63	0.58	

strategies, given different detection techniques and DGAs. This is an example of botnet counter-counter-measurements [61].

Botmasters would like the ROC curve to be as close to $y = x$ as possible, while security personnel want to minimize the distance in the ROC curve. This allows us to model the problem as a Two-Person Zero-Sum (TPZS) game [62] with the distance as the payoff function. Payoff function takes the distance in ROC curve from one point to the best detection point (0,1), which measures how well detectors can achieve given a threshold. The security personnel tries to minimize the payoff function, while botmaster tries to maximize it. If they end up with the same payoff function (at the saddle point), one strategy that both players tend to follow for their best interests will be obtained. Otherwise, it will result in a mixed strategy.

Table V gives the distances of different HMM-based DGA/detection pairs. Notice that maximin $=$ minimax $= 0.61$. So the game has a saddle point and both players should stick to the strategy corresponding to the saddle point, i.e. 500KL3 for botmaster and ED-based detection for security personnel.

Table VI gives the payoff matrix of player II (botmaster) using PCFG-based DGAs, where minimax $=$ maximin $=$ 0.58. Given the saddle point, the optimal strategies are pcfg_ipv4_num for botmaster and KL-based detection for security personnel.

We also consider the scenario where the botmaster is presented with all 8 DGAs, 500KL3 (the best HMM-based DGA), pcfg_ipv4_num (the best PCFG-based DGA), Kwyjibo and 5 real-life DGAs. The corresponding payoff matrix is shown in Table VII, where minimax $\neq$ maximin. We can solve it with an algebraic method [62].

TABLE VII: The payoff matrix of player II using HMM-based, PCFG-based, Kwyjibo and 5 real-life DGAs

		player I: security personnel (minimizer)			
		KL	ED	JI	min
player II: botmaster (maximizer)	hmm_500KL3	0.68	0.61	0.64	0.61
	kwyjibo	0.37	0.51	0.30	0.30
	zeus	0.21	0.09	0.05	0.05
	srizbi	0.12	0.39	0.05	0.05
	kraken	0.33	0.57	0.31	0.31
	conficker	0.34	0.55	0.33	0.33
	pcfg_ipv4_num	0.65	0.63	0.57	0.57
	torpig	0.11	0.06	0.006	0.006
	max	0.68	0.63	0.64	

	ED	JI
hmm_500KL3	0.61	0.64
pcfg_ipv4_num	0.63	0.57

TABLE VIII: Reduced 2×2 matrix for HMM results

a) First we observe that the column of KL is dominated by ED and JI. Because for each DGA, we always have D_{KL} $> min(D_{ED}, D_{JI})$. Since a security professional has no incentive to choose KL detection at all, we can delete the column of KL.
b) Similarly for the rows, we observe that the rows of hmm_500KL3 and pcfg_ipv4_num dominate the other DGAs. Since a botmaster has no incentive to choose DGAs except hmm_500KL3 and pcfg_ipv4_num, we can only keep the rows that are dominated.
c) The original matrix is reduced to a 2$\times$2 matrix (Table VIII).

The optimal strategy for the botmaster is to choose hmm_500KL3 with probability p, where

$$p = \frac{0.57 - 0.63}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.67 \qquad (10)$$

The optimal strategy for the security personnel is to choose ED detection with probability q, where

$$q = \frac{0.57 - 0.64}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.78 \qquad (11)$$

And the expected value of the game yields

$$V = \frac{0.61 \times 0.57 - 0.63 \times 0.64}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.62 \qquad (12)$$

where V is the expected distance between the ROC curve and point (0,1), when both players stick to their optimal strategy.

VI. Impacts on existing DGA-detecting systems

In this section, we evaluated the DGAs discussed earlier (our DGAs and botnet DGAs) against two DGA-detecting systems, BotDigger [8] and Pleiades [9]. BotDigger [8] detects botnet DGAs by analyzing DNS traffic in terms of quantity, temporal and linguistic features. Pleiades [9] uses a combination of clustering and classification algorithms to identify DGA-generated domain names by analyzing DNS traffic. Both claim that they can achieve up to 100% TPR and 0 FPR for detecting botnets using DGA-generated domain names. Similar to the two detection systems, BotMeter [19] assesses the population distribution of DGA-based botnets by analyzing DNS query patterns.

A. BotDigger

BotDigger [8], proposed by Zhang et al. in 2016, detects botnet DGAs. Botnets typically generate multiple domain names, but only pay to register a few. This means that DGA traffic is correlated with many Non-Existent Domains (NX-Domains) returns. BotDigger uses NXDomain correlation together with features including quantity, temporal and linguistic evidence to detect an individual bot by monitoring traffic in local network. Their experiment shows that BotDigger can detect all Kraken bots and 99.8% of Conficker bots. Since our work focuses on linguistic features, we can test the DGAs using the linguistic features processing of BotDigger. The other features are outside the scope of this work. The BotDigger code is available online (https://github.com/hanzhang0116/BotDigger) and we re-designed the experiments to test our DGAs using this code.

We use the 11 domain linguistic features from BotDigger to detect DGAs: (1) length of dictionary words in a domain name, (2) percentage of dictionary words in a domain name, (3) length of the longest meaningful substring (LMS) in a domain name, (4) percent of the length of the LMS in a domain name, (5) entropy in a domain name, (6) normalized entropy in a domain name, (7) number of distinct digital characters in a domain name, (8) percent of distinct digital characters in a domain name, (9) number of distinct characters in a domain name, (10) percent of distinct characters in a domain name, and (11) length of a domain name.

To calculate the dissimilarity of a domain linguistic feature between two domains (denoted as D_1 and D_2), we denote the features of D_1 as f_{1j} and the features of D_2 as f_{2j} where $1 \le j \le 11$. Based on the paper [8], the dissimilarity of feature j is calculated as:

$$S_j(D_1, D_2) = \begin{cases} 0, & \text{if } f_{1j} = 0 \text{ and } f_{2j} = 0. \\ \frac{|f_{1j} - f_{2j}|}{max(f_{1j}, f_{2j})}, & \text{otherwise.} \end{cases} \qquad (13)$$

The overall dissimilarity of all features between D_1 and D_2 is calculated as:

$$S_{All}(D_1, D_2) = \sqrt{\frac{\sum_{j=1}^{11} S_j(D_1, D_2)^2}{11}} \qquad (14)$$

After all 11 features are extracted from all domain names, hierarchical clustering algorithm is used to classify the domain names. In one experiment and for each DGA, we mix 100 DGA-generated domain names with 100 legitimate domain names (subdomain names randomly chosen from the same domain name), and classify them with the hierarchical clustering algorithm. We start with taking each domain name as a cluster and combine the two clusters with the least dissimilarity until there are two clusters left. One cluster is labeled as 'malicious' and the other one is 'legitimate'. Then a set of TPR and FPR can be calculated based on the result. We repeat the experiments 1000 times and calculate the mean and standard deviation (Sd) of all TPRs and FPRs, and the distance between (mean_FPR, mean_TPR) and (0,1). The result is in Table IX and Figure 7[4].

TABLE IX: The mean and standard deviation of the TPR and FPR, and the distance between (mean_FPR, mean_TPR) and (0,1) under BotDigger detection

DGA	Mean of TPR	Mean of FPR	Sd of TPR	Sd of FPR	distance to (0,1)
pcfg_ipv4_num	0.89	0.96	0.25	0.13	0.89
hmm_500KL3	0.92	0.97	0.50	0.16	0.92
kraken	0.56	0.94	0.49	0.15	0.57
zeus	0.56	0.92	0.50	0.18	0.57
conficker	0.45	0.93	0.50	0.16	0.46
torpig	0.75	0.91	0.43	0.19	0.75
srizbi	0.48	0.92	0.50	0.18	0.49
kwyjibo	0.42	0.90	0.49	0.20	0.44

From the distance in Table IX and Figure 7, we can see that the performance of DGAs in evading BotDigger is hmm_500KL3 $>$ pcfg_ipv4_num $>$ torpig $>$ kraken $=$ zeus $>$ srizbi $>$ conficker $=$ kwyjibo (using 95% confidence intervals). The results show that our DGAs can evade the cutting-edge DGA-detecting systems. We suspect the reasons why BotDigger is not effective are because (1) linguistically, our DGAs are more advanced in mimicking legitimate domain names than other botnet DGAs, and (2) for botDigger, other features like quantity and temporal evidence are important to detect DGAs, even though they are outside the scope of this paper. The DGAs we provide should be more difficult for BotDigger to detect than other DGAs. We suspect that BotDigger performance in detecting our DGAs would be improved if the other features were available for evaluation.

[4]Note that we might use the complements of the decisions since these choices provide better results than the original choices. When ROC curves are consistently below the diagonal, it is common practice to use the complement.

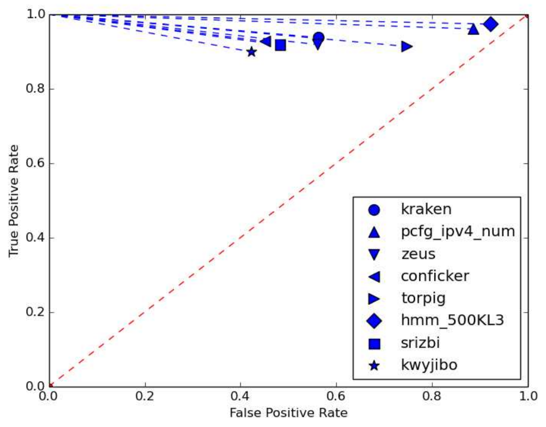


Fig. 7: ROC curves of all DGAs under BotDigger detection system

Possibly, BotDigger could be augmented to use more advanced lexical features.

B. Pleiades

Pleiades [9], proposed by Antonakakis et al. in 2012, also uses NXDomains of DNS traffic to differentiate DGA-generated domain names and legitimate domain names, and achieves 99% TPR and 0.1% FPR in detecting Conficker DGA. Since Pleiades is commercialized from Damballa, we re-write the code based on the paper [9].

We use 19 features from Pleiades to detect DGAs: (1)-(12) The median, mean and standard deviation of the distribution of n-gram frequency from a list of domain names, where n = 1,2,3,4; (13)-(14) The mean and standard deviation of the Shannon entropy of a list of domain names; (15)-(18) The mean, median, standard deviation and variance of the length of a list of domain names; and (19) The number of distinct characters appeared in a list of domain names.

In one experiment and for each DGA, we have 100 groups of domain names, half of which are DGA-generated domain names and the other half are legitimate domain names (subdomain names randomly chosen from the same domain name). Each group contains 10 domain names. We apply k-means clustering ($k = 2$) to the 100 groups and get 2 clusters. One cluster is labelled as 'malicious' and the other one is 'legitimate'. As with BotDigger, we repeat the experiments 1000 times and calculate the mean and standard deviation of all TPRs and FPRs, and the distance between (mean_FPR, mean_TPR) and (0,1). The result is in Table X and Figure 8.

From the distance in Table X and Figure 8, we can see that the performance of DGAs in evading Pleiades is pcfg_ipv4_num $>$ kraken $=$ hmm_500KL3 $=$ kwyjibo $=$ conficker $=$ torpig $=$ srizbi $>$ zeus (using 95% confidence intervals). From the TPR and FPR, Pleiades detection of all DGAs achieve less than 29.7% TPR and almost 0 FPR. This shows that the linguistic features alone are not effective for detecting DGA use. Note that the result is the best scenario case for Pleiades, because we put domain names generated

TABLE X: The mean and standard deviation of the TPR and FPR, and the distance between (mean_FPR, mean_TPR) and (0,1) under Pleiades detection

DGA	Mean of TPR	Mean of FPR	Sd of TPR	Sd of FPR	distance to (0,1)
pcfg_ipv4_num	0.114	0.014	0.32	0.12	0.89
hmm_500KL3	0.192	0	0.39	0	0.81
kraken	0.185	0	0.39	0	0.82
zeus	0.297	0	0.46	0	0.70
conficker	0.216	0	0.41	0	0.78
torpig	0.218	0	0.41	0	0.78
srizbi	0.227	0	0.42	0	0.77
kwyjibo	0.197	0	0.40	0	0.80

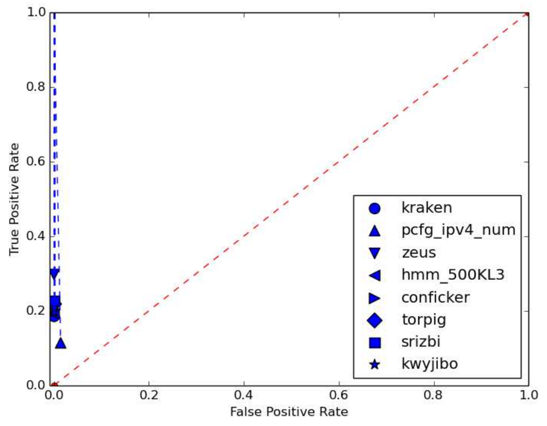


Fig. 8: ROC curves of all DGAs under Pleiades detection system

by the same DGA into the same group before the clustering. In real-life DGA detection, this is the best case assumption. We suspect why Pleiades is not effective is that it assumes (1) domain names generated by the same DGA have similar statistical characteristics, and (2) if domain names are identified as coming from a new DGA, they are either similar to an existing DGA or can be generated using an existing DGA. Both assumptions are not true for our HMM/PCFG-based DGAs. However, Pleiades might be able to identify our DGAs by analyzing the overlapping sets of queried domain names, but it requires ISP-level cooperation and extensive computation. The use of passive DNS for data collection could make this possible. Pleiades' success seems to depend on leveraging NXDomain features. We believe both Pleiades and BotDigger could benefit from integrating more sophisticated linguistic primitives. Future work should test BotDigger/Pleiades against a research botnet that uses our DGA domain name. This could more fully explore the interactions between NXDomain and linguistic features in DGA detection.

VII. Discussion & Future work

In this section, we discuss anomalies observed in Section V.

For botnet DGAs, properties other than detection evasion must be considered. One is the collision of generated domain names with existing (registered) benign domain names.

TABLE XI: Collision percentages between DGAs and legitimate domain names

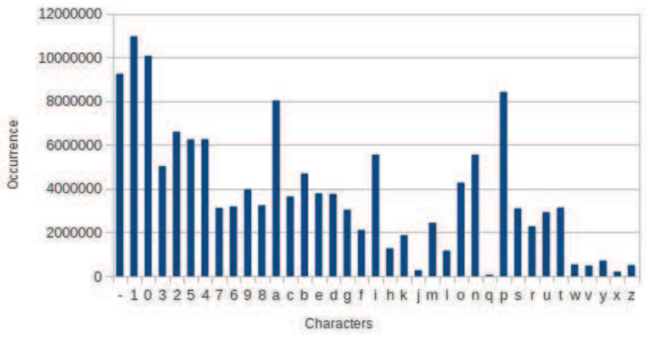
DGA	hmm _500KL3	pcfg _ipv4_num	kwyjibo	conficker
Collision Percentages	3.9	5.2	7.7	0
DGA	zeus	srizbi	kraken	torpig
Collision Percentages	0	0.1	0.5	0

Because botmasters prefer benign-looking, but not registered domain names for C&C servers. We investigate the collision rate between all DGA-generated domains (our best HMM-based DGA, PCFG-based DGA, Kwyjibo and 5 real-life botnet DGAs) and legitimate domain names. Figure XI shows the percentage of domain name collisions between DGAs and benign domain names. We see that our DGAs have at most a 5% collision rate. The research DGA Kwyjibo has an 8% collision rate. The real-life DGAs have almost no collisions (less than 1%), which suggests that they sacrifice their ability to evade detection for fewer collisions. Or, alternatively, they do not mimic legitimate domain names well enough to get collisions. Also, since DGAs are used to generate domain names as rendezvous points for botmasters and bots, they want to generate the same domain name within a reasonable, finite number of iterations. For HMM-based and PCFG-based DGAs, this is possible. When botmasters and bots agree on the starting state and seed of the random number generator, the path in HMM and PCFG is guaranteed to be the same, so the generated domain names are the same. These DGAs could be deployed in a manner that reduces the number of NXDomain responses.

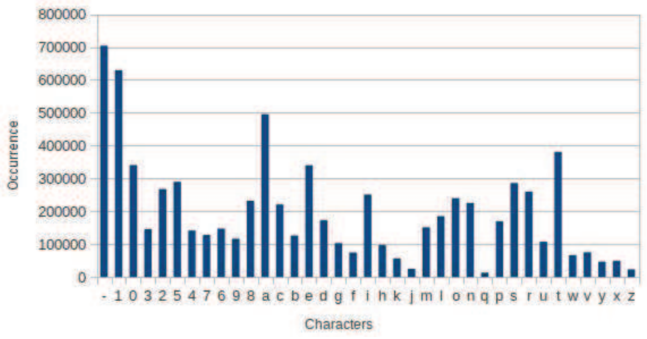
An HMM is a probabilistic regular grammar, and a PCFG is a context-free grammar. According to the Chomsky Hierarchy [55], unlike regular grammars, context-free grammars can generate recursive patterns. So in theory, the PCFG-based DGA will produce patterns that cannot be expressed by regular expressions. These patterns should be harder to detect than HMM-based approaches. For the three approaches used:

- KL distance looks at letter probability distributions. These patterns exist in both HMMs and PCFGs. Whereas the context-free patterns could include more complex internal (e.g., recursive) patterns.
- Edit distance looks at changes of individual letters between domain names. Since regular grammars construct names letter by letter, it is reasonable for this approach to detect these patterns. But context-free grammars can embed sequences of letters between two letters, so one would expect ED to be less effective.
- JI, like KL, is based on the probability distributions of letters. Both regular and context-free grammars should produce characteristic letter distributions. Once again, the context-free grammars could have more complex internal structures.

The existing detection approaches look at probability distributions for letters and letter pairs, which are features suited to regular grammars. It should be possible to design PCFGs



(a) b"



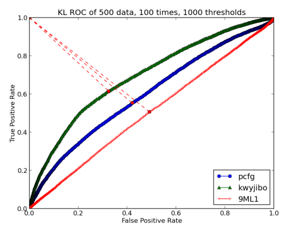
(b) b"

Fig. 9: Statistics of characters in domain names (a) without pre-processing (b) with pre-processing

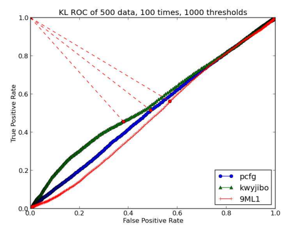
that manipulate these features to avoid detection. Research into more complex PCFGs should be fruitful.

The ROC curves in KL/ED/JI detection give us similar results in detecting HMM-generated and PCFG-generated domain namesin detecting HMM-generated and PCFG-generated domain names. This is because, the parse tree (Figure 3) and dictionaries used for the PCFGs (Table IV) may not be ideal as PCFG inputs. It is difficult to find a dictionary with letters and numbers to represent the characteristic patterns in real IPv4 domain names. More research is needed to find the effective context-free techniques for generating effective DGAs. Questions related to proper dictionary development are also interesting directions for future work.

The collected legitimate domain names from the entire IPv4 space were pre-processed (Appendix A) before inputting them as training sets to infer HMMs. Statistics of characters in domain names with and without pre-processing are in Figure 9. The figure shows significant difference in the symbol distributions. To prove that the performance of our DGAs are not affected by the preprocessing technique, we compare the performance of DGAs generated by two datasets. Figure 10 (a) shows the performance of DGAs generated without pre-processing under KL detection, and Figure 10 (b) shows the performance with pre-processing. We can see that there is a consistent tendency HMM > PCFG > Kwyjibo, and both HMM and PCFG are very close to $y = x$. This shows that HMM and PCFG approaches are resilient. Our pre-processing



(a) b”



(b) b”

Fig. 10: (a) KL detection on training sets without pre-processing (b) KL detection on training sets with pre-processing

has some influence, but it is not essential.

With each new defense (attack), the botnets (network defenders) adapt. The fight against network criminality has become an endless game of cat and mouse. One could suppose that network defenders could use HMMs/PCFGs to detect our new DGAs. The inferred HMMs/PCFGs represent the statistical features of both legitimate and HMM/PCFG-generated domain names, which means that they are identical statistically. If they use the HMMs/PCFGs to distinguish our DGA-generated domain names from legitimate domain names, it will have a high FPR. Network defenders would have to combine these lexical features with other features, like NXDomain returns, for detecting DGA use. However, it is reasonable to use HMMs/PCFGs to detect botnet DGA-generated domain names, which would be a promising future work.

Possible future work includes: (1) considering other text analysis metrics to test DGA performance; (2) tuning parameters in the HMMs and PCFGs to improve their performance; (3) finding additional DGA applications; (4) modelling the economics of botmasters, victims and security personnel, and (5) testing HMMs/PCFGs inferred with passive DNS domain names rather than reverse DNS domain names[5].

VIII. Conclusions

In this paper, we propose two novel DGAs (HMM-based and PCFG-based) and compare them with five real-life botnet DGAs and the Kwyjibo DGA. Three detection schemes (KL, ED, and JI) are used, which have been proven effective [3] in detecting DGA-generated domains. Experimental results show that the proposed DGAs not only successfully thwart these detection techniques, but also outperform the other DGAs in terms of anti-detection performance. The two DGAs also evade deployed botnet detection systems, BotDigger and Pleiades. Game theory analysis is used to find the optimal strategies for security personnel and botmasters. We propose future improvements and research topics which might improve our DGAs as well as DGA detection tools. The use of game theory for designing DGA detection systems seems particularly promising. We hope this paper will help security personnel choose the optimal detection techniques to detect C&C servers and finally take down the botnets.

[5]This could also help determine the difference between domain names found using passive DNS and reverse DNS.

References

[1] C. Li, W. Jiang, and X. Zou, “Botnet: Survey and case study,” in *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*. IEEE, 2009, pp. 1184–1187.

[2] X. Zhong, Y. Fu, L. Yu, R. Brooks, and G. K. Venayagamoorthy, “Stealthy malware traffic not as innocent as it looks,” in *Malicious and Unwanted Software: The Americas (MALWARE), 2015 10th International Conference on*, October 2015.

[3] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, “Detecting algorithmically generated domain-flux attacks with dns traffic analysis,” *Networking, IEEE/ACM Transactions on*, vol. 20, no. 5, pp. 1663–1677, 2012.

[4] M. Xie, H. Yin, and H. Wang, “Thwarting e-mail spam laundering,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 2, p. 13, 2008.

[5] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, “Exposure: a passive dns analysis service to detect and report malicious domains,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 14, 2014.

[6] J. Raghuram, D. J. Miller, and G. Kesidis, “Unsupervised, low latency anomaly detection of algorithmically generated domain names by generative probabilistic modeling,” *Journal of Advanced Research*, 2014.

[7] S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan, “Detecting algorithmically generated malicious domain names,” in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 48–61.

[8] H. Zhang, M. Gharaibeh, S. Thanasoulas, and C. Papadopoulos, “Botdigger: Detecting dga bots in a single network,” 2016.

[9] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon, “From throw-away traffic to bots: Detecting the rise of dga-based malware,” in *USENIX security symposium*, 2012, pp. 491–506.

[10] P. Liu, W. Zang, and M. Yu, “Incentive-based modeling and inference of attacker intent, objectives, and strategies,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, no. 1, pp. 78–118, 2005.

[11] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, “Tracking and characterizing botnets using automatically generated domains,” *arXiv preprint arXiv:1311.5612*, 2013.

[12] Z. Wei-wei and G. J. L. Qian, “Detecting machine generated domain names based on morpheme features,” 2013.

[13] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla, “Automatic extraction of domain name generation algorithms from current malware,” in *Proceedings of the NATO Symposium IST-111 on Information Assurance and Cyber Defense, Koblenz, Germany*, 2012.

[14] H. Crawford and J. Aycock, “Kwyjibo: automatic domain name generation,” *Software: Practice and Experience*, vol. 38, no. 14, pp. 1561–1567, 2008.

[15] M. Gasser, “A random word generator for pronounceable passwords,” DTIC Document, Tech. Rep., 1975.

[16] M. Mowbray and J. Hagen, “Finding domain-generation algorithms by looking at length distribution,” in *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 395–400.

[17] S. Marchal, J. François, R. State, and T. Engel, “Semantic based dns forensics,” in *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*. IEEE, 2012, pp. 91–96.

[18] X. Ma, J. Zhang, J. Tao, J. Li, J. Tian, and X. Guan, “Dnsradar: outsourcing malicious domain detection based on distributed cache-footprints,” *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 11, pp. 1906–1921, 2014.

[19] T. Wang, X. Hu, J. Jang, S. Ji, M. Stoecklin, and T. Taylor, “Botmeter: Charting dga-botnet landscapes in large networks,” in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 334–343.

[20] T.-S. Wang, H.-T. Lin, W.-T. Cheng, and C.-Y. Chen, “Dbod: Clustering and detecting dga-based botnets using dns traffic analysis,” *Computers & Security*, vol. 64, pp. 1–15, 2017.

[21] S. Kullback and R. A. Leibler, “On information and sufficiency,” *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[22] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.

[23] B. Bigi, “Using kullback-leibler distance for text categorization,” in *Proceedings of the 25th European Conference on IR Research*, ser. ECIR’03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 305–319. [Online]. Available: http://dl.acm.org/citation.cfm?id=1757788.1757818

[24] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (roc) curve.” *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[25] V. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[26] Z. Su, B.-R. Ahn, K.-Y. Eom, M.-K. Kang, J.-P. Kim, and M.-K. Kim, “Plagiarism detection using the levenshtein distance and smith-waterman algorithm,” in *Proceedings of the 2008 3rd International Conference on Innovative Computing Information and Control*, ser. ICICIC ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 569–. [Online]. Available: http://dx.doi.org/10.1109/ICICIC.2008.422

[27] W. J. Heeringa, “Measuring dialect pronunciation differences using levenshtein distance,” Ph.D. dissertation, University Library Groningen][Host], 2004.

[28] D. Jurafsky, “Minimum edit distance - definition of minimum edit distance,” 2014, https://web.stanford.edu/class/cs124/lec/med.pdf.

[29] H. Small, “Co-citation in the scientific literature: A new measure of the relationship between two documents,” *Journal of the American Society for information Science*, vol. 24, no. 4, pp. 265–269, 1973.

[30] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. [Online]. Available: http://nlp.stanford.edu/IR-book/information-retrieval-book.html

[31] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, “A comprehensive measurement study of domain generating malware.”

[32] D. Schales, J. Jang, T. Wang, X. Hu, D. Kirat, B. Wuest, and M. P. Stoecklin, “Scalable analytics to detect dns misuse for establishing stealthy communication channels,” *IBM Journal of Research and Development*, vol. 60, no. 4, pp. 3–1, 2016.

[33] A. Kasza, “Using algorithms to brute force algorithms,” 2015, https://blog.opendns.com/2015/02/18/at-high-noon-algorithms-do-battle/.

[34] T. Pereira, “A look inside tinba botnets,” 2015, http://blog.anubisnetworks.com/blog/a-look-inside-tinba-botnets.

[35] Damballa, “Domain generation algorithms (dga) in stealthy malware,” 2012, https://www.damballa.com/domain-generation-algorithms-dga-in-stealthy-malware/.

[36] J. Hagen and S. Luo, “Why domain generating algorithms (dgas)?” 2016, http://blog.trendmicro.com/domain-generating-algorithms-dgas/.

[37] J. Edwards, “The mad max dga,” 2016, https://www.arbornetworks.com/blog/asert/mad-max-dga/.

[38] S. Newspaper, “Cracking of sphinx trojan dga opens the door for botnet takedown,” 2016, http://www.securitynewspaper.com/2016/10/24/cracking-sphinx-trojan-dga-opens-door-botnet-takedown/.

[39] P. Paganini, “Cisco talos profiled the goznym botnet after cracking the trojan dga,” 2016, http://securityaffairs.co/wordpress/51744/malware/goznym-botnet-profiling.html.

[40] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, “Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus,” in *MALWARE*. IEEE, 2013, pp. 116–123. [Online]. Available: http://dblp.uni-trier.de/db/conf/malware/malware2013.html#AndriesseRSPB13

[41] S. Shin, G. Gu, N. Reddy, and C. P. Lee, “A large-scale empirical study of conficker,” *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 2, pp. 676–690, 2012.

[42] F. Leder and T. Werner, “Know your enemy: Containing conficker,” *The Honeynet Project, University of Bonn, Germany, Tech. Rep*, 2009.

[43] ——, “Containing conficker,” 2008. [Online]. Available: œhttp://net.cs.uni-bonn.de/wg/cs/applications/containing-conficker

[44] H. Crawford and J. Aycock, “Kwyjibo: Automatic domain name generation,” *Softw. Pract. Exper.*, vol. 38, no. 14, pp. 1561–1567, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1002/spe.v38:14

[45] B. News, “Spam on rise after brief reprieve,” 2008, http://news.bbc.co.uk/2/hi/technology/7749835.stm.

[46] J. Wolf, “Technical details of srizbis domain generation algorithm,” 2008. [Online]. Available: œhttp://www.fireeye.com/blog/technical/botnet-activities-research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html

[47] 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 *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS ’09. New York, NY, USA: ACM, 2009, pp. 635–647. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653738

[48] F. M. Liang, “Word hy-phen-a-tion by com-put-er,” Ph.D. dissertation, Citeseer, 1983.

[49] J. M. Schwier, “Pattern recognition for command and control data systems,” Ph.D. dissertation, Clemson, SC, USA, 2009, aAI3369281.

[50] C. Lu, “Network traffic analysis using stochastic grammars,” Ph.D. dissertation, Clemson, SC, USA, 2012.

[51] C. Lu, J. M. Schwier, R. Craven, L. Yu, R. R. Brooks, and C. Griffin, “A normalized statistical metric space for hidden markov models.” *IEEE T. Cybernetics*, vol. 43, no. 3, pp. 806–819, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/tcyb/tcyb43.html#LuSCYBG13

[52] L. Yu, J. M. Schwier, R. M. Craven, R. R. Brooks, and C. Griffin, “Inferring statistically significant hidden markov models,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 7, pp. 1548–1558, 2013.

[53] A. Beale, “12 dicts introduction - the 3esl list,” 2003, http://wordlist.aspell.net/12dicts-readme/.

[54] C. R. Shalizi and J. P. Crutchfield, “Computational mechanics: Pattern and prediction, structure and simplicity,” *Journal of statistical physics*, vol. 104, no. 3-4, pp. 817–879, 2001.

[55] N. Chomsky, “Three models for the description of language,” *Information Theory, IRE Transactions on*, vol. 2, no. 3, pp. 113–124, 1956.

[56] T. L. Booth and R. A. Thompson, “Applying probability measures to abstract languages,” *Computers, IEEE Transactions on*, vol. 100, no. 5, pp. 442–450, 1973.

[57] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[58] T. E. Harris, *The theory of branching processes*. Courier Dover Publications, 2002.

[59] Z. Chi, “Statistical properties of probabilistic context-free grammars,” *Computational Linguistics*, vol. 25, no. 1, pp. 131–160, 1999.

[60] J. M. Schwier, R. R. Brooks, C. Griffin, and S. T. S. Bukkapatnam, “Zero knowledge hidden markov model inference.” *Pattern Recognition Letters*, vol. 30, no. 14, pp. 1273–1280, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/prl/prl30.html#SchwierBGB09

[61] Y. Fu, B. Husain, and R. R. Brooks, “Analysis of botnet counter-counter-measures,” in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*. ACM, 2015, p. 9.

[62] J. Von Neumann and O. Morgenstern, “Theory of games and economic behavior,” *Bull. Amer. Math. Soc*, vol. 51, no. 7, pp. 498–504, 1945.

[63] R. team, “Welcome to project sonar!” 2013, https://community.rapid7.com/community/infosec/sonar/blog/2013/09/26/welcome-to-project-sonar.

Appendix A
How to obtain IPV4 domain names

To obtain domain names for the IPv4 space, we used the Sonar project [63], which provides a database of IPv4 reverse DNS records (updated at 2014-03-28). We downloaded the domain records of the entire IPv4 space as legitimate domain names. Since Botmasters register DGA-generated domain names to communicate with bots, DGAs should not generate domains that conflict with existing records. So we deleted the following patterns in IPv4 domain names to avoid registered domain names:

1) IP addresses (XXX-XXX-XXX-XXX where XXX is a number in the range 0-255)
2) Repeated patterns with a large number of records (ip-X, iptv-X, dsl-X, ti-X, host-X, hostX ull-X, ntl-X, link-X, OL-X, ss-X, dhcp-X, dhcpX, sl-X, i-d-X, user-X, dyn-X, static-X,adsl-X, ppp-X, softbank-X, Dinamico-X, -ipbfp-X, AWBLnew-X, ipX, 0xX, client-X), where X is a number or a IP address

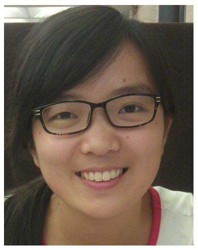
Also, botmasters use fast-flux to change the mapping between domain names and IP addresses to evade detection. They often register free domain names from dynamic DNS websites. Most of those websites don’t accept domain with special characters other than letters, numbers and hyphen, so we deleted domain names with those patterns.

Appendix B
How to obtain a *number+hyphen* list to generate PCFG training sets

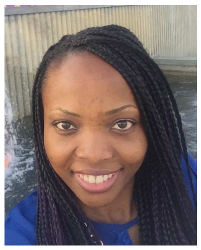
PCFG requires training sets containing information from IPv4 domain names. A hyphenation algorithm [48] was used to extract syllables from IPv4 domain names, but it lacked numbers and hyphens. In order to have a reasonable *number + hyphen* list, we went through the following steps:

- Step1: Parse all IPv4 domains by separating alphabetic and numerical characters, and delete repeated entries in each list. The aphabet list is called List 1, and the numerical list is called List 2. For example, we separated $789abc - 123def$ as 789, abc, 123, and def.
- Step 2: Regenerate a *number + hyphen* list (called List 3) by adding an optional hyphen to each entry from List 2 in Step 1.

To generate domain names for PCFG training sets, we calculated the probability of there being a number in an IPv4 domain name is 0.567. Then we generated domains by choosing entries from List 1 and List 3 in a way that guarantees the probability of a number being in the generated domain name is also 0.567.


Yu Fu is a 4th year PhD student in the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. She received BS in Electrical Engineering in East China University of Science and Technology (ECUST), Shanghai, China. Her research interests include network traffic analysis, protocol mimicry and botnet markets.

Lu Yu received the BS in information engineering and MS degree in control theory from Xi'an Jiaotong University, Xi'an, Shaanxi, China, and the PhD degree in electrical engineering from Clemson University, Clemson, South Carolina. Her research interests include Markov process and game theory.


Oluwakemi Hambolu received the MS degree in computer engineering from Clemson University , and is currently working towards the PhD degree in the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. Her research interest include privacy preserving statistics.

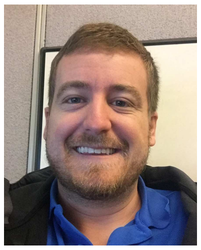
Ilker Ozcelik received the MS degree in electrical engineering from the Syracuse University in 2010, and is currently working toward the PhD degree in electrical engineering in the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. His research interests include network traffic analysis, network security and game theory.

Benafsh Husain is 4th year PhD candidate in School of Computing at Clemson under Dr. Joshua Levine. He received her B.Sc in Biomedical Engineering from Bombay University in 2008 and M.Sc in Electrical Engineering from California Polytechnic, SLO in 2010. She is currently working towards her dissertation on Performance Visualization techniques. Research interests include Scientific and Information Visualization, Image Processing, Networking and Network Security.

Jingxuan Sun received the BS in Electrical Engineering from Dalian University of Technology, Dalian, China. She is working towards Master degree in Computer Engineering from Clemson University. Her research interests include authentication and authorization in high performance file system.

Karan Sapra is 4th year PhD candidate in Electrical and Computer Engineering at Clemson under Dr. Melissa Smith. He received B.Sc in Computer engineering with Mathematical Science minor from Clemson University in 2011. He is currently working towards his dissertation on Performance modeling in High Performance Architecture. Research interests include P2P, Cloud, Networking and Network Security, and Sensors.

Dan Du received the BEE degree in electronic information engineering from Central China Normal University and received the MSPH degree in Physics from Huazhong University Of Science and Technology. She is currently a graduate student in ECE department, Clemson University. Her research interests include network security and data leak prevention.



Christopher Tate Beasley received his BA degree in electrical engineering from Western Carolina University, Cullowhee, NC and a Masters Degree in computer engineering from Clemson University, Clemson, SC. He is currently a full time employee of the Intel corporation.

Dr. Richard R. Brooks (M'97-SM'04) received the BA degree in mathematical sciences from The Johns Hopkins University, Baltimore, MD, and the PhD degree in computer science from Louisiana State University, Baton Rouge. He is currently an associate professor with the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. His research interests include adaptive distributed systems and game theory. He is a senior member of the IEEE.