Unsupervised, low latency anomaly detection of algorithmically generated domain names by generative probabilistic modeling

Jayaram Raghuram a,*, David J. Miller a, George Kesidis a,b

a Department of Electrical Engineering, Pennsylvania State University, University Park, PA 16802, USA
b Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802, USA

ABSTRACT

We propose a method for detecting anomalous domain names, with focus on algorithmically generated domain names which are frequently associated with malicious activities such as fast flux service networks, particularly for bot networks (or botnets), malware, and phishing. Our method is based on learning a (null hypothesis) probability model based on a large set of domain names that have been white listed by some reliable authority. Since these names are mostly assigned by humans, they are pronounceable, and tend to have a distribution of characters, words, word lengths, and number of words that are typical of some language (mostly English), and often consist of words drawn from a known lexicon. On the other hand, in the present day scenario, algorithmically generated domain names typically have distributions that are quite different from that of human-created domain names. We propose a fully generative model for the probability distribution of benign (white listed) domain names which can be used in an anomaly detection setting for identifying putative algorithmically generated domain names. Unlike other methods, our approach can make detections without considering any additional (latency producing) information sources, often used to detect fast flux activity. Experiments on a publicly available, large data set of domain names associated with fast flux service networks show encouraging results, relative to several baseline methods, with higher detection rates and low false positive rates.

Introduction

Online bot networks (botnets) are used for spam, phishing, malware delivery, distributed denial of service (DDoS) attacks, as well as unauthorized data exfiltration. Fast-flux service networks (FFSNs) are an evasive type of bot network, employing a large number of compromised IP addresses (machines) as proxy slaves, with client requests to visit the web server first resolved to the proxies and only then forwarded from them to
the real (malicious) server(s), controlled by the bot master. The robustness and longevity of an FFSN is attributable to rapid fluxing of the proxies (on the order of seconds or a few minutes), as well as possibly of the domain names themselves [1]. Recently developed botnets such as Conficker, Kraken, and Torpig use rapid domain name fluxing, wherein the bots DNS-query a series of randomly generated (synchronized by a starting seed) candidate domain names. When a DNS query is successful, the bot has the proper domain name to use in engaging with the bot master in command and control (C&C) communications. The apparent premise is that the large number of domain-name candidates greatly increases the (blacklisting) difficulty for a defense system, whereas the bot master need only remember the names that it (periodically) chooses to be DNS-registered [2,3]. Increasing the frequency with which the master changes the registered domain name will make it more difficult for the bot master to be identified. Apart from FFSNs, algorithmically generated domain names are also used in spam emails to avoid detection based on domain name and signature based blacklists. Direct approaches such as trying to reverse engineer the random domain name generation algorithm used by the bots may be highly time and resource consuming, and may have a low success rate, given that the bots can frequently change the algorithm used [4].

Several different strategies have been proposed to detect FFSNs. One is to build supervised classifiers (based on labeled benign and malicious network examples) which exploit features extracted based on DNS querying that should indicate fast flux of widely distributed, compromised machines; e.g., the number of DNS A-records in a single lookup or in all lookups, the number of unique involved autonomous systems, time-to-live, the domain’s age, and countries of registration [1,2]. Separately, detection algorithms have been proposed to identify fast domain-name fluxing, both by distinguishing computer-generated names from authentic, human-generated ones and from detecting DNS failure signatures, inherent to fast domain flux [3,5].

In Yadav et al. [3], the authors hypothesize that, in algorithmically choosing a long sequence of candidate domain names, bots will tend to use distributions for letters/syllables/n-grams that do not closely match the true distribution (associated with valid domain names). One reason could be that e.g., in choosing names from among the valid words in a dictionary, there is non-negligible probability of choosing an existing (reserved) domain name (or of achieving increased scrutiny by using a name too close to an existing domain name). Moreover, it is simply the case that current, existing FFSNs do not use the most sophisticated mechanisms for stochastically generating their (malicious) domain names. Yadav et al. [3] proposed a trace-based approach, wherein either for an individual IP address or for a connected clique of IP addresses, one measures the empirical distribution of domain names on the n-gram space. One can then use metrics such as the Kullback–Leibler distance, the Jaccard index, and the string edit distance to measure how close the empirical distribution is to a distribution based on a training set of valid domain names, and how close to a distribution based either on known FFSN names or on some assumed model for FFSN domain name generation. In Al-Duwairi and Manimaran [6] and Al-Duwairi et al. [7], the authors propose an interesting approach called “GFlux” for detecting botnet based DDoS and fast flux attacks using the Google search engine. In their approach, first a list of IP addresses associated with a potentially malicious domain name is found, and search queries based on its domain name and IP addresses are then input to Google. A very small number of hits (or search results) indicate that the domain is likely to be associated with malicious activity.

The approach in Yadav et al. [3] is trace-based, requiring the collection of a sufficient number of domain names for each IP address (or connected IP clique) to allow a reasonably accurate empirical estimate of the n-gram (e.g., bigram) distribution. Thus, it is inherently a high-latency method. Moreover, if there is relatively high flux in the IP addresses, it could be that there will be an insufficient number of domain names for each IP address (or IP address clique) to reasonably estimate the n-gram distribution. A disadvantage of the GFlux approach is that it may trigger false positives in the case of newly set-up, but legitimate DNS bindings with statistically normal domain names. In this paper, we propose an anomaly detection approach based on a fully generative probability model for the valid domain name space. The domain name modeling uses techniques from natural language processing and machine learning, and exploits the fact that valid domain names are likely to contain words that are part of a large (common) lexicon. Using such a (null hypothesis) model, estimated based on a large “training set” of valid domain names, one can calculate the likelihood of any individual domain name candidate (obtained from spam email, from a honeypot, or from a suspected web site). If the likelihood is very low, then the domain name is detected as suspicious. The advantage of this approach over Yadav et al. [3] and Yadav and Reddy [5] is that it is a low latency method (uses a pre-trained model of valid domain names) and makes no underlying assumptions about the stochastic model bots use in generating domain names.

It is worth mentioning that some recent works such as [8–10] have also proposed methods for domain name generation. In Crawford and Aycock [8], a domain name generation tool called Kwyjibo was proposed, which is capable of generating random, yet pronounceable strings that cannot be typically found in the English language. This has applications in areas like random generation of usernames, passwords, and domain name strings which cannot be easily replicated. In Wagner et al. [9], a method called Smart DNS brute-forcer was developed to synthesize new domain names for the purpose of DNS probing. They used a simple generative model for domain names, wherein the empirical distribution of the number of labels, the length of the labels, and the distribution of character n-grams in the labels are calculated on a training data set of domain names. In Marchal et al. [10], the method of Wagner et al. [9] was extended by leveraging semantic analysis of domain names in order to make improved guesses for new and related domain names, which can be useful for DNS probing. However, when considered in the context of the problem of detecting algorithmically generated domain names, we found that the domain name models proposed in these works are quite simplistic and not well suited for this problem. We evaluated the detection performance when the smart DNS brute-forcer method proposed by Wagner et al. [9] is used for modeling valid domain names, and found that our method performs significantly better, as shown in the experimental results section of this paper.
Methodology

In this section, we first describe our method for pre-processing and modeling valid domain names. Next, the method for estimating the model parameters from a data set of valid domain names is described. Finally, our anomaly detection method for detecting suspicious, algorithmically generated domain names (and thus distinguishing from valid domain names) is described.

Modeling of domain names

A domain name is a component of the Uniform Resource Locator (URL) that is used to identify a device or a resource on the Internet. It consists of one or more strings, called domains, delimited by dots. For example, in the URL http://en.wikipedia.org/wiki/Domain_name, the domain name is en.wikipedia.org. The rightmost domain in the domain name is called the top level domain (TLD) (org in this example), and the subsequent domains going from right to left are called second level domain, third level domain, and so on. The component strings of domain names can consist of English letters ‘a’ to ‘z’ (case insensitive), digits ‘0’ to ‘9’, and the character ‘-‘ at some position other than the beginning or the end of the string.

Compound splitting and pre-processing

The component strings in a domain name are usually formed by concatenating valid English words, proper nouns, numbers, abbreviated (compressed) words, acronyms, slang words, and even words (phrases) from other languages transliterated into English. A few examples are nytimes, yourfilehost, product-reviews, craigslist, cricinfo, deutschebahn, and hdfc bank. In order to learn meaningful models for domain names, it is useful to perform some pre-processing on the component strings. First, the top level domain and the generic ‘www’ are removed from all the domain names. Then, the ‘*’ and ‘-’ characters are considered as delimiters, and the domain name is split at the position of these characters (i.e., ‘*’ and ‘-’ are replaced with a single space), giving a number of substrings. If there are any numbers in the substrings, the portion to the left and right of the numbers (if any) are separated, and the numbers are discarded. This is done because, under our generative model, numbers (digits) are not likely to be informative about whether the domain names were generated algorithmically. Supposing that we have a large lexicion of words from the English language, we may be able to parse out words from the domain name substrings. For example, usatoday can be parsed into usa today, hdfcbank can be parsed into hdfc bank (although ‘hdfc’ may not be a part of the word list). This problem, known as compound splitting, word segmentation, or word breaking, has been addressed before and some efficient methods have been developed to solve it [11–13]. However, some of these methods can only split a string such that all the words in the split are recognized by the word list. In the case of domain names, this may not be very effective. Thus, we implemented a method which can parse a string based on a large word list and separate out the recognized words, even if there are unrecognized substrings on either (or both) sides of the recognized word strings. In particular, our method may parse a string as: $S_1, W_1, S_2$, where $W_1$ is a valid word, but $S_1$ and $S_2$ are unrecognized substring “phrases”. To illustrate our parsing steps, consider the example domain name www.imovies4you.com. After processing and parsing, the substrings extracted will be ‘i’, ‘movies’, and ‘you’.

Markov modeling of the character sequence

A simple model for the substrings in a domain name is obtained by modeling the joint probability of the characters, assuming the parsed substrings are statistically independent of each other. Suppose a domain name is represented by its component substrings ($w_1, \ldots, w_n$), where the $i$-th substring of length $l_i$ is $w_i = (w_{i1}, \ldots, w_{il_i})$, $i = 1, \ldots, n$. We model its probability as $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i)$. The joint probability of characters in the substring $w_i$ can be generally written as $P(w_i) = P(w_{i1}) \prod_{j=2}^{l_i} P(w_{ij} | w_{i1}, \ldots, w_{ij-1})$, where $w_{ij}$ take values from the set of English letters $A$. If we make a $k$-th order Markov assumption ($k < l$) that $w_{ij}$ is conditionally independent of $w_{i1}, w_{i2}, \ldots, w_{i,k-1}$ given $w_{ij-1}, w_{ij-2}, \ldots, w_{ij-k}$, then the joint probability is given by $P(w_i) = P(w_{i1}) \prod_{j=2}^{l_i} P(w_{ij} | w_{ij-1}, \ldots, w_{ij-k})$. Since the number of probabilities needed to be estimated increases exponentially with $k$, $k$ is chosen to be small, typically in the range $2–5$. Also, we assume that the conditional distribution of characters is stationary, i.e., $P(w_{ij} | w_{ij-1}, \ldots, w_{ij-k})$ does not depend on the position of the character, $j$.

Given a training set of strings, one can estimate the conditional probabilities using the maximum likelihood (ML) or maximum a posteriori (MAP) estimation methods. However, even for moderately large $|A|$ and small $k$, using these methods directly can result in noisy or even undefined estimates for some character tuples. This problem has been well studied in the natural language processing literature, and addressed using what are called smoothing or interpolation methods [14,15]. In this paper, we focus on a method called Jelinek–Mercer smoothing [16], in which higher order conditional probability models are interpolated (smoothed) using lower order models.

In this method, the interpolated $k$-th order conditional probability model is a convex combination of the $k$-th order maximum likelihood estimated conditional probability model and the interpolated $(k – 1)$-th order conditional probability model.

The interpolated conditional probability models for lower orders are defined in the same way, recursively. For example, the conditional probability model for $k = 3$ is given by

$$P_{3l}(w_{ij} | w_{ij-1}, w_{ij-2}, w_{ij-3}) = \lambda_3 P_{ML}(w_{ij} | w_{ij-1}, w_{ij-2}, w_{ij-3}) + (1 - \lambda_3) P_{3l}(w_{ij} | w_{ij-1}, w_{ij-2}),$$

where,

$$P_{3l}(w_{ij} | w_{ij-1}, w_{ij-2}) = \lambda_2 P_{ML}(w_{ij} | w_{ij-1}, w_{ij-2}) + (1 - \lambda_2) P_{3l}(w_{ij} | w_{ij-1}),$$

$$P_{3l}(w_{ij} | w_{ij-1}) = \lambda_1 P_{ML}(w_{ij} | w_{ij-1}) + (1 - \lambda_1) P_{3l}(w_{ij})$$

and $P_{ML}$ refers to the maximum likelihood estimates. The hyperparameters $\lambda_1, \lambda_2, \lambda_3 \in [0, 1]$ control the contribution of the models of different orders. The method for setting these hyperparameters is discussed in a later section. The motivation
behind this method is that when there is insufficient data to estimate a probability in the higher order models, the lower order models can provide useful information and also avoid zero or undefined probabilities. It can be shown that the maximum likelihood estimates are given by the normalized empirical frequency counts over the training set of “known normal” (white listed) domain names, i.e.,

$$P_{ML}(w_{ij}|w_{ij-1}, \ldots, w_{ij-k}) = \frac{N(w_{ij}, w_{ij-1}, \ldots, w_{ij-k})}{\sum_{w_{ij}\in\mathcal{D}} N(w_{ij}, w_{ij-1}, w_{ij-2}, \ldots, w_{ij-k})},$$

(2)

where $N(w_{ij})$ denotes the frequency count on a training set. If this probability model is learned based on a large training set of valid domain names, the character tuples that occur frequently in the training set will tend to have high probabilities, and the character tuples that occur less frequently will have low probabilities. A domain name generated randomly based on some algorithm is likely to have character sequences which have low probability under the valid domain name model, i.e., they are likely to be anomalies or outliers relative to the valid domain name model. This is discussed further in the section Anomaly detection approach.

Parametric modeling of the number of substrings and the substring lengths

In addition to modeling the character sequences in the substrings of a domain name, one would expect that it is useful to model other characteristics of a domain name such as the number of substrings it possesses (after pre-processing and parsing), the total length (number of characters) in the domain name, and the lengths of the component substrings, because these features are likely to have different probability distributions on a set of valid domain names than on a set of algorithmically generated domain names. In order to substantiate this claim, we calculated the empirical probability distributions of these features on a data set of valid domain names and on a data set of domain names associated with fast flux or attack activity (these data sets which are used in our experiments will be described in a later section). The empirical probability mass functions (PMFs) of the number of substrings, the total length of the domain name, the length of the second substring, and the length of the third substring estimated from each of the data sets are compared in Fig. 1(a-d), which reveal substantial differences. Accordingly, we now represent a domain name as $(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n)$, where $n$ is the number of substrings, $l = l_1 + \ldots + l_n$ is the total length of the domain name, $l_i$, $i = 1, \ldots, n$ are the substring lengths, and $w_i$, $i = 1, \ldots, n$ are the substrings. The joint probability of the domain name (assuming substring independence) can then be expressed as

$$P(N = n, L = l, l_1, \ldots, l_n, w_1, \ldots, w_n) = P(N = n)P(L = l|N = n)P(L_1 = l_1, \ldots, L_{n-1} = l_{n-1})$$

(3)

where the uppercase and lowercase notations are used to denote random variables and their corresponding values. To simplify notation, we will drop the use of the uppercase, and assume that the symbols identify the probability distributions.

That is, $P(n)$ is the probability of a domain name having $n$ substrings, $P(l|n)$ is the probability that the length of the domain name is $l$ given that it has $n$ substrings, $P(l_1, \ldots, l_{n-1}|l, n)$ is the joint probability of the substring lengths given the length of the domain name and the number of substrings. Since these probability distributions are unknown, a commonly used approach is to model them with suitable parametric distributions and estimate the parameters of the distributions from a training data set. We next describe our choices for these.

Since the number of substrings in domain names does not usually take a large value (In Fig. 1(a), the domain names with more than 5 substrings have a negligible probability mass), we decided to model $P(n)$ directly with the empirical PMF, with a smoothing factor added to avoid zero probabilities outside the support of the training set.

That is,

$$P(n) = \frac{N(n) + \epsilon}{\sum_{n=1}^{N_{max}} N(n) + \epsilon}, \quad n = 1, 2, \ldots,$$

(4)

where $\epsilon$ is a smoothing hyperparameter and $N_{max}$ is the maximum number of substrings over the domain names in the training set. The method for setting $\epsilon$ is discussed in a future section. Next, we discuss our choice of model for $P(l|n)$. Given the number of substrings, we assume that the individual substring lengths are statistically independent and that the length of substring $i$ follows a Poisson distribution with parameter $\mu_i$, i.e.,

$$P(l_i|n; \mu_i) = \frac{e^{-\mu_i} \mu_i^{l_i}}{(l_i - 1)!}, \quad l_i = 1, 2, \ldots,$$

where the domain of the distribution starts from 1 because the length of a substring has to be at least 1 character. Given the number of substrings $N = n$, it can be shown that the total length $L = \sum_{i=1}^{n} L_i$ also has a Poisson distribution with a shifted domain and parameter $\mu = \sum_{i=1}^{n} \mu_i$, given by

$$P(L|n; \mu) = \frac{e^{-\mu} \mu^{L-n}}{(L-n)!}, \quad l = n, n+1, \ldots,$$

(5)

Another property of independent Poisson distributed random variables is that, given their sum $L = l$, the joint distribution of the random variables $L_i$, $i = 1, \ldots, n-1$ is a multinomial distribution ($l_i$ is deterministic given $l$ and $l_n = \sum_{i=1}^{n-1} l_i = l - 1$). In this case, it follows that

$$P(l_1, \ldots, l_{n-1}|l, n; \mu) = \frac{(l-n)!}{(l_1-1)! \cdots (l_{n-1}-1)!} \prod_{i=1}^{n-1} \left( \frac{\mu_i}{\mu} \right)^{l_i-1}, \quad l_i = 1, 2, \ldots,$$

(6)

where $\mu = (\mu_1, \ldots, \mu_n)$. The joint distribution of characters in a substring, given their lengths is chosen as the interpolated model $P_{\text{int}}(w_j|l_j) = \prod_{i=1}^{l_j} P_{\text{ML}}(w_j | w_{j-1}, \ldots, w_{j-k}, l_j)$, which was discussed earlier. An alternate, more sophisticated model for the substrings which makes use of word lists is discussed in the next section.

From the discussion so far, we have a fully generative model, consistent with the following stochastic domain name generation steps:

1. Select the number of substrings $n$ by sampling from the distribution $P(n)$.
2. Select the total length of the domain name $l$ by sampling from the Poisson distribution $P(l|n; \mu)$. 

J. Raghuram et al.
3. Select the individual substring lengths $l_i$, $i = 1, \ldots, n$, by sampling from the multinomial distribution $P(l_1, \ldots, l_n) / l, n; q$.
4. Independently, for each substring of length $l_i$, generate the character sequence $w_i$ according to the model $P_{\text{int}}(w_i | l, E_i = 1)$.

Modeling recognized word occurrences in domain names

So far, the model presented for substrings in a domain name considered the joint distribution of its characters, making some conditional independence assumptions. Although such a model captures dependencies between sequences of characters, it does not take into account the possibility that one or more substrings (obtained from the parsing step) could be part of a lexicon or vocabulary, as is often the case with domain names. As we discussed earlier, domain names are usually created by humans by concatenating words from their vocabulary, which also include proper nouns abbreviations, acronyms, slang words, etc. Using a suitably collected eclectic word list that is representative of words usually found in valid domain names, it is possible to develop a more sophisticated model for the substrings in valid domain names. Also, algorithmically generated domain names which are usually part of some malicious activity such as FFSNs are unlikely to contain substrings which are part of a word list [3]. Hence, it should be useful to learn a model of valid domain names which combines both the joint probability of the character sequences, and the probability of occurrence of recognized words from a word list.

Consider a word list $\mathcal{V} = \{v_1, \ldots, v_M\}$ with $M$ words and with maximum word length $l_{\text{max}}$. Let $\forall_i$ be the set of words of length $l_i$ such that $\bigcup_{i=1}^{j} \forall_i = \mathcal{V}$. Let $q_\bullet(v)$ be a PMF on the words of length $l_i$ from the word list, such that $\sum_{v \in \forall_i} q_\bullet(v) = 1$. Let $I(c)$ be the binary indicator function, which takes a value 1 (0) if the condition $c$ is true (false). Also, let $E_i$ be the binary random variable which takes a value 1 (0) if a substring of length $l_i$ belongs to (does not belong to) the word list. We propose to model a substring $w$ of length $l_i$, given that it belongs to the word list, via the following mixture model:

$$P_{\text{int}}(w | l, E_i = 1) = \pi q_\bullet(w) + (1 - \pi) P_{\text{int}}(w | l, E_i = 1) = \pi q_\bullet(w) + (1 - \pi) \frac{P_{\text{int}}(w | l, I(v \in \forall_i))}{\sum_{v \in \forall_i} P_{\text{int}}(v | l, I(v \in \forall_i))}$$

$$= \pi q_\bullet(w) + (1 - \pi) \frac{P_{\text{int}}(w | l, I(v \in \forall_i))}{\sum_{v \in \forall_i} P_{\text{int}}(v | l)}, \forall w \in \mathcal{A}^l$$

Fig. 1 Plots of empirical PMF of the number of substrings, total length, length of the second substring, and length of the third substring estimated on a data set of normal domain names and on a data set of attack domain names.
where $\pi$ is the prior probability that a word is selected from the word list according to the PMF $q_v(w)$, rather than $P_{\text{str}}(w|l, E_t = 1)$. The PMF $P_{\text{str}}(w|l, E_t = 1)$ is the joint probability of the characters in the substring with the interpolated model, conditioned on the event that the substring is in the word list, and the final simplified expression in (7) is obtained by applying Bayes rule. For substrings of length $l$ which are not part of the word list, we use the joint probability of the characters in the substring with the interpolated model, conditioned on the event that the substring is not in the word list, given by

$$P_{\text{str}}(w|l, E_t = 0) = \frac{P_{\text{str}}(w|l)I(w \notin V_l)}{\sum_{v \in V_l} P_{\text{str}}(w|l)I(w \notin V_l)} = \frac{P_{\text{str}}(w|l)I(w \notin V_l)}{1 - \sum_{v \in V_l} P_{\text{str}}(w|l)}, \quad \forall w \in \mathcal{A}^l$$  \hspace{1cm} (8)

Also, let $\gamma \in [0, 1]$ be the prior probability of selecting a substring from the word list.

For this model, only step 4 of the domain name generation algorithm can be estimated using a data set of valid domain names. The distribution $P(n)$ is directly calculated using (4). We assume that the conditional probabilities of the character tuples in $P_{\text{str}}(w|l)$ are front-end estimated using (2) on the entire training data set. The parameters of the mixture model are $\theta$ and $\gamma = \{\pi, \{q_v(w), \forall v \in V_l, l = 1, \ldots, l_{\text{max}}\}\}$. The portion of the log-likelihood of the data $\mathcal{X}$ which depends on these parameters is given by

$$L(\theta, \gamma) = \sum_{x \in \mathcal{X}} \sum_{i=1}^n I(w_i \in V_l) \times [\log \gamma + \log(q_v(w_i)) + (1 - \pi)P_{\text{str}}(w_i|l, E_t = 1)]$$

$$+ \sum_{x \in \mathcal{X}} \sum_{i=1}^n (1 - I(w_i \in V_l))[\log(1 - \gamma) + \log P_{\text{str}}(w_i|l, E_t = 0)],$$

where $x$ is used as shorthand for $(n, l, l_1, \ldots, l_c, w_1, \ldots, w_k)$. It can be easily shown that the MLE estimate for $\gamma$ is

$$\gamma = \frac{T \sum_{i=1}^n I(w_i \in V_l)}{\sum_{i=1}^T n_i},$$

which is just the proportion of substrings in the domain name training set which are from the word list.

The MLE solution for the parameters in $\theta$, subject to the appropriate constraints, does not have a closed form solution. However, a widely used method for solving problems of this kind involving mixture models is the Expectation Maximization (EM) algorithm [18,19], which finds a local maximum of the log-likelihood by iteratively maximizing a lower bound, one which is both easier to maximize and which usually has a closed form maximizer. At each iteration, the maximizer of the lower bound necessarily increases the value of the log-likelihood, and the iterations are repeated until a local maximum of the log-likelihood is found. For our problem, the EM algorithm can be summarized as follows:

1. Initialize parameters: We chose the initialization $\pi = 0.5$ and $q_v^{(0)}(w) = \frac{1}{|V_l|}, \forall v \in V_l, l = 1, \ldots, l_{\text{max}}$.
2. Iterate: For $r = 0, 1, 2, \ldots$, until $L(\theta, \gamma)$ converges
   (a) E-Step: For $t = 1, \ldots, T$, and $i \in \{1, \ldots, n_t\}$ such that $w_i \in V_l$, calculate the component posterior

$$P(d_{ij} = 1|w_i, l_i; \theta^{(r)}) = \frac{\pi^{(r)} q_v^{(r)}(w_i) P_{\text{str}}(w_i|l_i, w_j \in V_l)}{\pi^{(r)} q_v^{(r)}(w_i) + (1 - \pi^{(r)}) P_{\text{str}}(w_i|l_i, w_j \notin V_l)},$$  \hspace{1cm} (9)

where the superscript $r$ on the parameters denotes their value at the $r$-th EM iteration.

(b) M-Step: Re-estimate the parameters

$$\pi^{(r+1)} = \frac{\sum_{i=1}^T \sum_{i=1}^{n_t} P(d_{ij} = 1|w_i, l_i; \theta^{(r)}) I(w_i \in V)}{\sum_{i=1}^T \sum_{i=1}^{n_t} I(w_i \in V)},$$

$$q_v^{(r+1)}(w) = \frac{\sum_{i=1}^T \sum_{i=1}^{n_t} P(d_{ij} = 1|w_i, l_i; \theta^{(r)}) I(w_i = v)}{\sum_{i=1}^T \sum_{i=1}^{n_t} P(d_{ij} = 1|w_i, l_i; \theta^{(r)}) I(w_i \in V)}, \quad \forall v \in V_l, \forall l.$$  \hspace{1cm} (11)

We treat the occurrence or non-occurrence of a substring in the word list also as observed data.

Learning the model parameters

In the previous section, we discussed our proposed probability model for domain names. We now discuss how the parameters of this model can be estimated using a data set of valid domain names.

Maximum likelihood and Expectation Maximization

We use the well-known maximum likelihood estimation (MLE) framework [17,18], wherein the parameters of a probability model are found by maximizing the likelihood of a training data set under that model. Consider a training set of valid domain names given by $\mathcal{X} = \{(n, l, l_1, \ldots, l_c, w_1, \ldots, w_{n_t}), l = 1, \ldots, T\}$. It can be shown that the MLE solution for the parameter $\mu_i$ in the Poisson distribution of the length of substring $i$ is given by

$$\mu_i = \sum_{l_l = l_{\text{min}}}^{l_{\text{max}}} (l_i - 1) / \sum_{l_l = l_{\text{min}}}^{l_{\text{max}}} 1.$$
Unsupervised, low latency anomaly detection

Setting the hyperparameters
Recall that the interpolation weights \( \lambda_1, \lambda_2, \ldots \) in (1), and the smoothing factor \( \delta \) in (4) are hyperparameters. They are not estimated using the training data in order to avoid over-fitting, and are usually set using a separate validation data set, if available. Instead, we use 10-fold cross-validation (CV). In our model, the choice of parameters \( \lambda_1, \lambda_2, \ldots \) is independent of the choice of \( \delta \). Each of the \( \lambda_1, \lambda_2, \ldots \) is varied over twenty values in \([0,1]\) and the combination of values which has the largest average log-likelihood on the held out folds is chosen. Similarly \( \delta \) is chosen from a set of twelve values in the interval \([0.001, 100]\).

Anomaly detection approach

Once the parameters of the domain name models are estimated using a data set of valid domain names, the model can be used for detecting anomalous or algorithmically generated domain names. A natural choice for the test statistic for this detection problem is the logarithm of the joint probability of the test domain name under our estimated model of valid domain names. If this value is smaller than a threshold, then we decide that the test domain name is an anomaly. We next consider a number of different test statistics based on progressively more complex models of domain names, consistent with our earlier developments.

First we consider only the interpolated model for the character sequences in the substrings of a domain name. For a domain name represented by the vector \( (n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) \), the test (decision) statistic is given by

\[
T_1^{(c)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) = \sum_{i=1}^n \log P_{\text{err}}(w_i|l_i)
\]

\[
= \sum_{i=1}^n \sum_{j=1}^k \log P_{\text{err}}(w_{ij}|w_{ij-1}, \ldots, w_{ij-k}).
\]

(12)

The domain name is declared anomalous if \( T_1^{(c)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) < \eta \), where \( \eta \) is a suitably chosen threshold. However, in this approach, we are comparing the joint probabilities of domain names with different numbers of substrings and different substring lengths against the same threshold. As the length of a substring increases, the support of its joint probability increases exponentially. Therefore, the joint probability of a character sequence tends to decrease with increasing length. As a result, longer length sequences may be biased to be detected more often as anomalies than shorter length ones. In an attempt to correct this bias, we propose the following modifications of the test statistic (12).

\[
T_2^{(c)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) = \sum_{i=1}^n \log \left( \frac{P_{\text{err}}(w_i|l_i)}{\sqrt{E[P_{\text{err}}(W_i|l_i)]}} \right),
\]

(13)

and

\[
T_3^{(c)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n)
= \sum_{i=1}^n \left( \log P_{\text{err}}(w_i|l_i) - E[\log P_{\text{err}}(W_i|l_i)] \right),
\]

(14)

where the expected values are given by

\[
E[P_{\text{err}}(W_j|l)] = \sum_{w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_k} \prod_{i=1}^{j-1} P_{\text{err}}(w_i|w_{i-1}, \ldots, w_{i-k})^2,
\]

and

\[
E[\log P_{\text{err}}(W_j|l)] = \sum_{w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_k} \prod_{i=1}^{j-1} P_{\text{err}}(w_i|w_{i-1}, \ldots, w_{i-k})
\times \log P_{\text{err}}(w_j|w_{j-1}, \ldots, w_{j-k}).
\]

Since our model assumes the joint distribution of the characters to be a simple Bayesian network, the above summations over the character tuples can be computed efficiently using the Sum-Product algorithm (message passing) [20]. The idea behind dividing by the square root of the expected value in \( T_2^{(c)} \) is that it acts like an \( \ell^2 \) (Euclidean) norm of the vector of joint probabilities over all possible input tuples. In the case of \( T_3^{(c)} \), the idea is that the logarithm of the joint probability of the substrings should have different mean values for different substring lengths, and we subtract off the mean value.

Next, we consider the fully generative model which includes the probability distribution of the number of substrings, the total length of the domain name, and the individual substring lengths. Defining

\[
g(n, l, l_1, \ldots, l_n) = \log P(n) + \log P(l|n; \mu)
\]

\[
+ \log P(l_1, \ldots, l_n|l, n; \mu),
\]

the test statistics for a domain name \( (n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) \) are given by

\[
T_1^{(w)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n) = \sum_{i=1}^n \mathcal{I}(w_i \in \mathcal{V}_b) \log [\gamma P_d(w_i|l_i, E_i)
\]

\[
= \sum_{i=1}^n \mathcal{I}(w_i \notin \mathcal{V}_b) \times \log [(1 - \gamma) P_{\text{err}}(w_i|l_i, E_i = 0)],
\]

(16)

and

\[
T_2^{(w)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n)
= g(n, l, l_1, \ldots, l_n) + T_1^{(w)}(n, l, l_1, \ldots, l_n, w_1, \ldots, w_n).
\]

(17)

Note that in this case it is not clear how to apply bias correction for variable length substrings, since this model considers not only the joint distribution of the characters, but also the probability of occurrence of words from a word list. We consider the methods using test statistics in (12)–(15) as baseline approaches, with the test statistic for our proposed approach given in (16) and (17).

As another baseline method for comparison, we implemented the domain name modeling method of the Smart DNS brute-forcer [9,10], which simply models the label substrings in a domain name with a first order Markov model for the character sequences, as we discussed in the
Introduction section. We used the logarithm of the joint probability under this model as a test statistic for detection.

For all the above variants of the test statistic, the decision rule (normal or anomaly) is based on comparison with a threshold, which can be chosen such that the false positive rate is equal to \( a \). The false positive rate cannot be computed exactly, and hence is approximated using a sampling estimate. Alternatively, one could model the univariate distribution of the test statistic with a suitable parametric density (e.g., Gaussian, Student’s \( t \), Gamma density, etc.), for which it may be possible to compute the false positive rate directly. The detection rate and false positive rate performances of these test statistics are compared in the next section.

**Results and discussion**

We obtained a data set of valid (benign) domain names and a data set of attack domain names associated with fast flux activity from http://pcsei.twbbs.org/datasets/-1-fast-flux-attaack-datasets. They collected a list of benign domain names from sources such as well-known top websites listed by Alexa (http://www.alexa.com/topsites), and lists of popular blogs. They collected the fast flux data sets from sources such as ATLAS (http://atlas.arbor.net/summary/fastflux), domain name system blacklists (http://www.dnsbl.info/), and FluXOR [2]. The data set of benign domains has 90,588 names and the fast flux attack data set has 25,210 names. We held out 5000 randomly selected benign domain names as part of the test set for calculating the false positive rates. The entire set of attack domain names is used in the test set for calculating the detection rates. We collected a large list of words from internet sources such as the Wiktionary frequency lists (http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists), a text corpus from project Gutenberg (http://norvig.com/big.txt), a list of common male and female first and last names (http://www.census.gov/genealogy/www/data/1990surnames/names_files.html), and a list of common technical terms (http://www.techterms.com/list/a). The word list collected from these sources is used by the method which models word occurrences.

Receiver Operation Characteristic (ROC) curves are plotted for all the test statistics discussed in the previous section. The ROC curve is plotted by varying a threshold on the test

![ROC curves](image_url)

**Fig. 2** ROC curves for the test statistics based on the joint distribution of character sequences in the substrings parsed out of the domain names.
statistic, and for each threshold value calculating the detection rate and false positive rate on the test data set. In our problem, the detection rate is the fraction of attack domain names that are correctly detected as attack, and the false positive rate is the fraction of benign domain names that are incorrectly detected as attack. Recall that the decision rule is to declare a domain name as attack if its test statistic is smaller than a threshold, and declare it as benign otherwise. The area under the ROC curve (AUC) is frequently used as a figure of merit, with larger areas corresponding to better performance (with a maximum value of 1).

Performance using only character modeling

We made a third order \( (k = 3) \) Markov dependency assumption on the joint distribution of characters for all the methods developed in this paper. First, we evaluated the performance of the baseline test statistics \( T_{1}^{(c)} \), \( T_{2}^{(c)} \), and \( T_{3}^{(c)} \) (defined in (12)–(14)), which are based only on character modeling of the sub-strings representing the domain names. The corresponding ROC curves and their AUC values are shown in Fig. 2(a–c). The test statistic \( T_{1}^{(c)} \), which is simply the logarithm of the joint probability, has a relatively good detection performance. Among the modified test statistics, which attempt to handle the problem of comparing variable length domain names, \( T_{2}^{(c)} \) gives a small improvement in the AUC, but \( T_{3}^{(c)} \) performs poorly compared to the other two.

We also evaluated the effect of parsing the domain names as a pre-processing step. Instead of learning the Markov character transition probabilities from the parsed domain names (where the substrings are assumed to be independently generated), we just treated the domain names as a single character sequence. For this experiment we used the test statistic \( T_{2}^{(c)} \), and the ROC curve is shown in Fig. 2(d). Although the performance without parsing using the character based model does not change much compared to the performance with parsing applied, we will see that the use of word modeling from a word list (which is used to model strings once they are parsed) gives significant improvement.

Value of modeling the number of substrings and substring lengths

Next, we evaluated the method which models the number of substrings, the total length, and the length of the individual

![ROC curves for the test statistics based on the distribution of the number of substrings, the total length, the length of the individual substrings, and the joint distribution of characters.](image-url)
substrings, in addition to modeling the characters in the substrings. For this model, the ROC curves corresponding to the test statistics $T_i^{(c)}$, $i = 1, 2, 3$, (defined in (15)) are shown in Fig. 3(a–c). We observe that there is a small decrease in the AUC value in this case. Based on the clear difference between the empirical distributions of these features in Fig. 1, one would expect that modeling these feature distributions should increase the chance of detecting algorithmically generated domain names. Presumably, on this data set, just modeling the joint distribution of the characters in the domain names with the interpolated model captures the distribution of normal domain names well. Another reason could be that the single parameter Poisson distribution does not offer enough flexibility for modeling the length of the substrings well. Evaluating this model on other data domains of fast flux activity may give us a better understanding of this phenomenon. Next, we discuss the detection performance of the baseline domain name modeling method of Wagner et al. [9]. The ROC curve for this method, shown in Fig. 3(d) has significantly lower detection performance compared to the other methods developed in this paper. This is not surprising since this domain name model considers only first order character dependencies, does not use any smoothing method, or model the occurrence of recognized words from a vocabulary as we do. Note that the method of [3] also uses only character bigram probabilities in calculating metrics for anomaly detection.

**Value of modeling word occurrences from a word list**

Finally, we evaluated our most sophisticated proposed method, which also models the probability of occurrence of words from the word list we collected. The ROC curves for the test statistics $T_i^{(w)}$ and $T_i^{(w)}$ (defined in (16) and (17)) are shown in Fig. 4(a and b). We observe that this method has the best AUC performance, as compared to the methods which use only character modeling for the substrings in the domain name. On this data set, a high detection rate of about 0.9 can be achieved with a false positive rate of less than 0.1. The improvement in performance can be explained by the fact that valid domain names are usually embedded with recognizable words from a vocabulary. Also, domain names associated with fast flux activity do not *usually* contain meaningful words or phrases, since fast fluxing activity typically requires a large number of frequently generated domain names that do not already exist in the DNS. Thus, using deterministic patterns from a finite vocabulary would decrease the number of possible unique domain names (making domain name fast fluxing less effective). However, in our experiments we have observed that in some cases domain names associated with attack or malicious activity also contain some valid words embedded in the middle of randomly generated character sequences. On the other hand, we also observed that some valid domain name strings do not have much informative content. For example, they could be short acronyms, abbreviations, or slang words which may get detected as anomalies under the valid domain name model. To give some examples for both these scenarios, Table 1 shows a portion of valid and attack test set domain names ranked in order of increasing $p$-values (which are approximately calculated by sampling). Note that under a good model for valid domain names, anomalous domain names should have small $p$-values (close to 0).

### Table 1

| Domain Name | $p$-Value under null model | Valid or attack |
|-------------|---------------------------|----------------|
| nkotb       | 0.090852                  | Valid          |
| kdo od govern | 0.090903                | Attack         |
| sua od years    | 0.090997                 | Attack         |
| epupz        | 0.091044                  | Valid          |
| assetos      | 0.092950                  | Valid          |
| ngo duck half  | 0.094218                 | Attack         |
| eqo od federal | 0.094246                | Attack         |
| loser boi music blog spot | 0.094316    | Valid          |
| cool veg if exot    | 0.094363                | Attack         |
| images wun bit ip  | 0.094422                | Attack         |
| circle mat i me pav  | 0.094657              | Attack         |
| baux per ten forum | 0.094719               | Valid          |
| kreuz        | 0.110932                  | Valid          |

![ROC curves](image-url)  
**Fig. 4** ROC curves for the test statistics based on the modeling of substrings with word occurrences from a word list.
Conclusions

We proposed a method for generatively modeling the valid domain name space using natural language processing techniques, which can be used in an anomaly detection setting to detect suspicious looking (or algorithmically generated) domain names. The detection performance of our method on a real data set of malicious domain names associated with fast-flux activity is encouraging. We wish to emphasize that this detection of domain names associated with fast flux activity is based solely on modeling a representation of the domain names, and does not use any other background information like DNS lookups, or packet trace collection and analysis, which may be expensive and which can induce delay in the decision making. At the same time, there are limits to the detection performance achievable using only the domain name character strings. As discussed in the Results section, some valid domain names may just be short strings like acronyms or abbreviations (for example www.cbs.com, www.cnn.com), which do not have much information. On the other hand, some of the attack, fast flux, and blacklisted domain names used in our experiments have valid words concatenated with random-looking sequences, presumably to maximize their degree of confounding. Given these challenges, a detector based solely on domain names may be most effectively used as part of a larger detector/classifier system which uses additional discriminating features. Such a system could also be extended to an active learning framework which automatically identifies the best new samples to label by feasibly involving a human operator in the loop.

Conflict of interest

The authors have declared no conflict of interest.

References

[1] Holz T, Gorecki C, Rieck K, Freiling FC. Measuring and detecting fast-flux service networks. In: Proceedings of the network & distributed system security symposium; 2008.

[2] Passerini E, Paleari R, Martignoni L, Bruschi D. Fluxor: detecting and monitoring fast-flux service networks. In: Detection of intrusions and malware, and vulnerability assessment. Springer; 2008. p. 186–206.

[3] Yadav S, Reddy AKK, Reddy A, Ranjan S. Detecting algorithmically generated malicious domain names. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM; 2010. p. 48–61.

[4] Stone-Gross B, Cova M, Cavallaro L, Gilbert B, Szydlowski M, Kemmerer R, et al. Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on computer and communications security. ACM; 2009. p. 635–47.

[5] Yadav S, Reddy AN. Winning with DNS failures: strategies for faster botnet detection. In: Security and privacy in communication networks. Springer; 2012. p. 446–59.

[6] Al-Duwairi B, Manimaran G. Just-google: a search engine-based defense against botnet-based DDoS attacks. In: IEEE International conference on communications (ICC); 2009. p. 1–5.

[7] Al-Duwairi B, Al-Qudahy Z, Govindarasu M. A novel scheme for mitigating botnet-based DDoS attacks. J Networks 2013;8(2):297–306.

[8] Crawford H, Aycock J. Kwyjibo: automatic domain name generation. Softw Pract Exp 2008;38(14):1561–7.

[9] Wagner C, Francois J, Engel T, Wagener G, Dulaunoy A. SDBF: Smart DNS Brute Forcer. In: IEEE network operations and management symposium (NOMS); 2012. p. 1001–7.

[10] Marchal S, Francois J, Wagner C, Engel T. Semantic exploration of DNS. In: Networking. Berlin, Heidelberg: Springer; 2012. p. 370–84.

[11] Wang K, Thrasher C, Hsu BIP. Web scale NLP: a case study on URL word breaking. In: Proceedings of the 20th international conference on World Wide Web. ACM; 2011. p. 357–66.

[12] Koehn P, Knight K. Empirical methods for compound splitting. In: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics, vol. 1. Association for Computational Linguistics; 2003. p. 187–93.

[13] Khaitan S, Das A, Gain S, Sampath A. Data-driven compound splitting method for English compounds in domain names. In: Proceedings of the 18th ACM conference on information and knowledge management. ACM; 2009. p. 207–14.

[14] Chen SF, Goodman J. An empirical study of smoothing techniques for language modeling. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics. Association for Computational Linguistics; 1996. p. 310–18.

[15] Witten IH, Bell TC. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. IEEE Trans Inform Theory 1991;37(4):1085–94.

[16] Jelinek F. Interpolated estimation of Markov source parameters from sparse data. Pattern Recognit 1980:381–97.

[17] Poor HV. An introduction to signal detection and estimation. New York: Springer-Verlag; 1994. p. 173–5.

[18] Dempster A, Laird N, Rubin D. Maximum likelihood from sparse data via the EM algorithm. J Roy Stat Soc Ser B (Methodological) 1977:1–38.

[19] Yuille A, Stolorz P, Utans J. Statistical physics, mixtures of distributions, and the EM algorithm. Neural Comput 1994;6(2):334–40.

[20] Bishop CM. Pattern recognition and machine learning. New York: Springer; 2006. p. 394–8.