Dictionary Extraction and Detection of Algorithmically Generated Domain Names in Passive DNS Traffic

Mayana Pereira[1], Shaun Coleman[2], Bin Yu[1], Martine DeCock[2], and Anderson Nascimento[2]

[1] Infoblox Inc.
Santa Clara, California
email: mpereira@infoblox.com,biny@infoblox.com
[2] Institute of Technology, University of Washington Tacoma
Tacoma, Washington
e-mail: spcole@uw.edu, mdecock@uw.edu, andclay@uw.edu

Abstract. Automatic detection of algorithmically generated domains (AGDs) is a crucial element for fighting Botnets. Modern AGD detection systems have benefited from the combination of powerful advanced machine learning algorithms and linguistic distinctions between legitimate domains and malicious AGDs. However, a more evolved class of AGDs misleads the aforementioned detection systems by generating domains based on wordlists (also called dictionaries). The resulting domains, Dictionary-AGDs, are seemingly benign to both human analysis and most of AGD detection methods that receive as input solely the domain itself. In this paper, we design and implement a method called WordGraph for extracting dictionaries used by the Domain Generation Algorithms (DGAs) based solely on DNS traffic. Our result immediately gives us an efficient mechanism for detecting this elusive, new type of DGA, without any need for reverse engineering to extract dictionaries. Our experimental results on data from known Dictionary-AGDs show that our method can extract dictionary information that is embedded in the malware code even when the number of DGA domains is much smaller than that of legitimate domains, or when multiple dictionaries are present in the data. This allows our approach to detect Dictionary-AGDs in real traffic more accurately than state-of-the-art methods based on human defined features or featureless deep learning approaches.

Keywords: malicious domain name, Domain Generation Algorithm, Dictionary-AGD, malware detection, machine learning

1 Introduction

Whenever a client needs to connect to a server over the internet by using web addresses (domains), these are first translated into IP addresses. The Domain Name System (DNS) is responsible for doing this translation. Requests containing web

addresses arrive at DNS servers that reply with corresponding IP addresses, or with an error message in case the domain is not registered – an NX Domain. Malicious software (malware) also uses this mechanism to communicate with their command and control ($C\&C$) center. However, instead of using a single hard-coded domain to communicate with the $C\&C$ (which could be easily blocked), several malware families use a more sophisticated mechanism known as *Domain Generation Algorithms* (DGAs) [15]. DGAs provide a method for controllers of botnets to dynamically produce a large number of random domain names and select a small subset for actual command and control use. This approach makes blacklisting ineffective. Being able to detect algorithmically generated domains automatically becomes, thus, a vital problem.

Traditional DGA algorithms usually start from random seeds and produce domains that are distinctly different from common benign domains. They appear more "random looking", such as, for example, the domain sgxyfixkhuark.co.uk generated by the malware *Cryptolocker*. Traditional DGAs are detected with techniques that leverage the distribution of characters in the domain, either through human engineered lexical features [3, 11, 17] or through training deep neural networks [10, 16, 19, 20, 22, 23].

Lately, a newer generation of DGA algorithms has been observed. This new kind of DGA makes detection by the techniques mentioned above much harder, namely by producing domains that are similar to the ones created by a human. *Dictionary-based DGAs* generate domains by concatenating words from a dictionary. For example, the malware *Suppobox* [7], a known Dictionary-based DGA, produces domains such as: *heavenshake.net, heavenshare.net and leadershare.net* [15].

Due to the challenging nature of the problem of detecting Dictionary-AGDs based solely on the domain name string itself, one often resorts to other information such as the IP address of the source [8], or information about the time when the domain was sent to the DNS server [1]. This kind of information can be expensive to acquire, or due to privacy concerns, it might just not be available. Moreover, detecting an AGD based solely on the domain allows for inline, real-time blocking of such domain at the DNS server level – a highly desirable feature. Another existing approach to detect Dictionary-AGDs is to reverse engineer the malware [4], extracting the list of words in the dictionary and using this list to identify domains that are generated by the malware. This process is labor-intensive and time-consuming, making it unsuitable to detect new Dictionary-based DGA malware as soon as it emerges.

Little or no attention has been given in the literature to the problem that we address in this paper: detecting Dictionary-based DGAs purely based on the domain name string, and without reverse engineering the malware. A notable recent exception is the work by Lison and Mavroeidis [10] who constructed a deep learning-based DGA detection model that can detect Dictionary-AGDs generated from a "familiar" dictionary. Familiar in this context means that a large number of Dictionary-AGDs stemming from the same dictionary are assumed to be available as examples to train the model. Once trained, the model can

detect previously unseen Dictionary-AGDs provided that they originate from a dictionary that has already been seen during training time. In practice, it is natural for hackers to change the dictionary in a Dictionary-based DGA, leaving the problem of detecting Dictionary-AGDs largely unresolved.

Contributions. In this paper, we study the problem of detecting Dictionary-based DGAs. We show that a state-of-the-art DGA classifier based on human engineered lexical features that does well for traditional DGAs performs very poorly when confronted with Dictionary-based DGAs. We also show that deep neural networks, while better at detecting Dictionary-AGDs, struggle to maintain a consistent good performance in face of changes in the dictionary.

We propose the first effective method for *detecting and extracting* the dictionary from Dictionary-based DGAs purely by observing domain name strings in DNS traffic. The intuition behind our approach is that, for known Dictionary-based DGAs, the words from the dictionary are used repeatedly by the DGA in different combinations to generate domains. We leverage these repetitions and combinations within a graph-based approach to isolate Dictionary-based DGA domains in traffic.

The fact that our method is completely agnostic to the dictionary used as generator by the DGA, and in fact learns this dictionary by itself (Section 4), makes it very robust: if in the future the malware starts generating domains with new dictionaries we still detect them, as we show in our experiments (Section 6). Even in a highly imbalanced scenario, where the domain names generated by a specific Dictionary-based DGA algorithm make up only a very small fraction of the traffic, our WordGraph method is successful at isolating these domain names and learning the underlying dictionary.

The remainder of this paper is structured as follows: after presenting an overview of related work in Section 2 and recalling necessary preliminaries in Section 3, we describe our WordGraph method for detection and extraction of DGA dictionaries in Section 4. Next, in Section 5 we provide our experimental methodology. This section contains details about the ground truth and real life traffic data used in our experiments, as well as a more detailed description of the state-of-the-art methods that we compare our WordGraph method with, namely a random forest classifier based on lexical features, and a convolutional neural network based deep learning approach. In Section 6 we present the results of the various methods on ground truth data as well as on real traffic data, showing that, unlike the other approaches, the WordGraph approach has a consistently high true positive rate vs. an extremely low false positive rate. Furthermore, after deploying our solution in a real network, we detected variations of known dictionaries that have never been reported previously in the literature.

2 Related Work

Blacklists were one of the first actions taken by the security community to address the problem of malicious domains. These continuously updated lists serve as databases for known malicious entities. One of the main advantages of black-

lists is that they provide the benefit of lookup efficiency and precision. However, after the deployment of DGAs by recent malware, domain blacklisting became an ineffective technique for disrupting communications between infected machines and C&C centers.

As a consequence, alternative methods for detecting DGA domains have been proposed. In [21], Yadav et al. analyzed the distribution of alphanumeric characters as well as bigrams in domains that map to the same set of IP-addresses. This work is an extension of the analysis made by McGrath and Gupta [13] for differentiating phishing/non-phishing URLs. The approach focuses on classifying groups of URLs as algorithmically-generated or not, solely by making use of the set of alphanumeric characters used. The authors used statistical measures such as Kullback-Leibler divergence, Jaccard index, and Levenshtein edit distance to measure the distance of the probability distributions of the n-grams, in order to make a binary classification (DGA vs. Non-DGA).

In [3], Antonakakis et al. developed a bot detection system called Pleiades which uses a combination of lexical features and host-based features to cluster domains. The main novelty of their work is the use of Non-Existing Domains (NXDomain) queries to detect bots and as training data. Their insight is that most domain queries made from a bot result in non-existent domains. Given this observation they cluster NXDomains that have similar lexical characteristics and are queried by overlapping sets of hosts. In a second stage, the clusters are classified in order to identify their corresponding DGA family.

In order to achieve an overall solution, Schiavoni et al. [17] proposed Phoenix, a mechanism that makes two different classifications: a binary classification that identifies DGA- and non-DGA-generated domains, using a combination of string and IP-based features; and a multi-class classification that characterizes the DGA family, and finds groups of DGA-generated domains that are representative of the respective botnets.

With the intention of building a simple DGA classifier based on domain names only, Mowbray and Hagen [14] proposed a DGA detection classifier based solely on URL length distributions. The approach allows the detection of a DGA at the end of the first time slot during which the first infected machine is used for malicious queries. However, their approach is effective for only a limited set of DGA families.

All methods described above rely on the extraction of predefined, human engineered lexical features from the domain name string. Recently, several works have proposed DGA detection models based on deep learning techniques that learn features automatically, thereby offering the potential to bypass the human effort of feature engineering [10,16,20,22,23]. Deep learning approaches for DGA detection based on convolutional neural networks (CNNs) and long short term memory networks (LSTM) achieve a predictive performance that is at par with or better than the methods based on human defined lexical features, provided that enough training data is available.

The DGA detection methods that we have described so far in this section all use the domain name string itself, sometimes combined with some side in-

formation like IP-based features. All these methods have been proposed and studied in the context of *traditional DGA* detection. Traditional DGAs produce domain names that appear more random looking than usual benign domain names, even to human observers. This substantial difference between the domain name strings created by traditional DGAs vs. those created by humans is the underlying reason for the success of the DGA detection methods described above. A newer generation of DGA algorithms, the so-called Dictionary-based DGAs, attempt to evade the traditional DGA detection methods by producing domain names that look more similar to the ones created by humans. To this end, they concatenate words from a dictionary.

Since catching Dictionary-AGDs based on the domain name string itself is challenging, it is natural to look at side information instead. An interesting approach in this regard is the work of Krishnan et al. [8] who followed the insight of Antonakakis et al. [3] that infected machines tend to generate DNS queries that result in non-existent (NX) responses. Krishnan et al. applied sequential hypothesis tests and focused on NX traffic patterns of individual hosts to identify infected machines. More recently, Abbink and Doerr [1] proposed to detect DGAs based on sudden rises and declines of popularity of domain names in large networks. Neither of these approaches uses information about the domain name string itself, which sets it apart from the work in this paper.

Regarding the development of a classifier that can label a given domain name in real time as benign or malicious, solely based on the domain name string itself, there has been some initial success with deep learning approaches for catching Dictionary-AGDs [10]. As explained in Section 1, and as we also observe in Section 6, this appears to work well for previously seen dictionaries, but doesn't offer any guarantees for consistent predictive performance when the dictionary in the malware is changed, which can be considered as an adversarial attack on the machine learning model. We provide evidence in Section 6 that the WordGraph method proposed in this paper is resilient to such kind of attack, thereby making it the first of its kind.

Finally, we stress that all the existing methods described above are aimed at developing classifiers to distinguish between benign and malicious domain names. The method proposed in this paper goes beyond, by learning the underlying word patterns present in DNS traffic, and extracting the DGA-related words from traffic. This results in the first DGA detection method that automatically extracts malware information from traffic in the form of malware dictionaries. Once these dictionaries are known, it becomes straightforward to construct a domain name classifier based on them, as explained in Section 4.

3 Preliminaries

In this section we present definitions that are used throughout our method description in Section 4. We refer the reader to [6, 9, 12] for more detailed explanations of these concepts. Throughout this paper, a graph $G(V, E)$ (or G for brevity) is defined as a set V of vertices and a set E of edges. In an undirected

graph, an edge is an unordered pair of vertices. If vertex v is one of edge e's endpoints, v is called incident to e. The degree of a vertex is the number of edges incident to it.

Definition 1. *Path. Let $G = (V, E)$ be a graph. A walk $w = (v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1})$ in G is an alternating sequence of vertices and edges in V and E respectively so that for all $i = 1, \ldots, n$: $\{v_i, v_{i+1}\} = e_i$. A path in G is a walk with no vertex and no edge repeated.*

Definition 2. *Cycle. A closed walk or cycle $w' = (v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1}, e_{n+1}, v_1)$ on a graph $G(V, E)$ is an alternating sequence of vertices and edges in V and E such that $w = (v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1})$ is a walk, and the edge e_{n+1} between v_{n+1} and v_1 does not occur in w.*

Definition 3. *Cycle Basis. A closed walk on a graph $G(V, E)$ is an Eulerian subgraph if it traverses each edge in E exactly once. A cycle space of an undirected graph is the set of its Eulerian subgraphs. A cycle basis is a minimal set of cycles that allows every Eulerian subgraph to be expressed as a symmetric difference of basis cycles.*

Definition 4. *The average shortest-path length (APSL). Let F be the set of all pairs of nodes of a graph G in between which there is a path, then*

$$\mathrm{ASPL}(G) = \frac{1}{|F|} \sum_{(v_i, v_j) \in F} \mathrm{dist}(v_i, v_j) \tag{1}$$

where $\mathrm{dist}(v_i, v_j)$ is the number of edges on a shortest path between v_i and v_j.

Definition 5. *A connected component G' of a graph G is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in G.*

4 WordGraph Method

Let C be a set containing q domain name strings $\{c_1, \ldots, c_q\}$. Each domain name string consists of higher level domains (second-level domain, SLD) and a top-level domain (TLD), separated by a dot. For example, in *wikipedia.org*, the SLD is *wikipedia* and the TLD is *org*. Within C we have domains that are benign and domains that are generated by a Dictionary-based DGA. Our goal is to detect all the Dictionary-AGDs in C and to extract the dictionaries used to produce these domains.

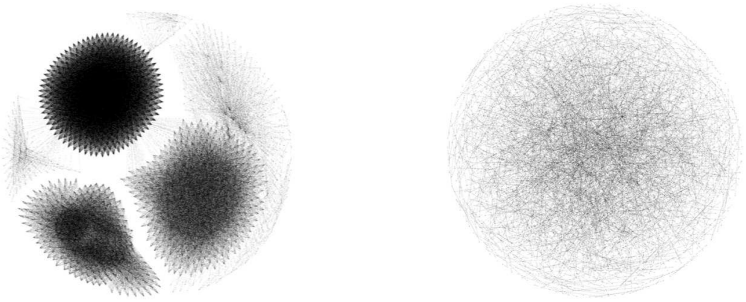
Extracting Words from Domains The word extraction method *learns* words from the set of domain name strings itself. Since Dictionary-based DGAs are known to use words repeatedly, we define a word as a sequence of at least m characters that appears in two or more SLDs within the set C. In the experimental results section, we use $m = 3$. We produce a set D of words as follows:

1. Set $D = \emptyset$
2. For every c_i and c_j in C, $i, j \in \{1, \ldots, q\}, i \neq j$:
   Denote by $l_{i,j}$ the largest common substring in c_i and c_j.
   If $|l_{i,j}| \geq m$, add $l_{i,j}$ to the set D.

It is important to point out that the above word extraction algorithm is applied to the entire set C, including both Dictionary-AGDs and legitimate domain names. The resulting set D will therefore have many elements that are not words from a Dictionary-based DGA. We will eliminate these words in a subsequent phase. To illustrate the word extraction algorithm, consider the following domains:

`facetype.com, facetime.com, bedtime.com,`
`faceboard.com, bedboard.com, bedding.com`

The resulting set of common substrings is $D = \{$face, time, bed, board, facet$\}$.



(a) Suppobox Malware domains (b) Alexa domains

Fig. 1: Differences in structure of (a) Word graph of Suppobox malware domains [7], and (b) Word graph of Alexa (benign) domains [2]. In (a) each dark region consists of words from a different malicious dictionary.

Word Graph Construction We split the set of domains C into partitions $C_1, \ldots, C_r$ such that all the domains within each C_i, $i \in \{1, \ldots, r\}$ have the same top-level domain (TLD). For each C_i we define a graph G_i as follows. The nodes of G_i are the words from the set D that occur in at least one domain name in C_i. Two nodes (words) of G_i are connected if they co-occur in the same domain in C_i, i.e. if there exists at least one domain $c_j \in C_i$ so that these words are both substrings of c_j. The division by TLD is motivated by the fact building separate graphs per TLD prevents noise and limit the graph size. Additionally, based on our observations, Dictionary-based DGAs use a limited number of different TLDs. We can therefore expect the Dictionary-AGDs to be concentrated in a small number of partititions.

In order to detect malware related words in each of the graphs G_i, we exploit the fact that subgraphs with words from malicious dictionaries present a different structure from subgraphs with words from benign domains. To illustrate this, in Figure 1 we visualize the word graph G_d of a set C_d of known Dictionary-AGDs and the word graph G_b of a set C_b of known benign domain names respectively. For more details on how these domain names were obtained, we refer to Section 5.1. The three dark regions in Figure 1(a) each correspond to a different dictionary used by the DGA algorithm. Note that in reality the partitions C_i, $i \in \{1, \ldots, r\}$ contain a mixture of Dictionary-AGDs and benign domain names, meaning that the distinction is not as clear-cut as in Figure 1. Still there are important observations to be made from Figure 1 that explain the rationale of our approach for detecting malware dictionaries in the word graphs G_i.

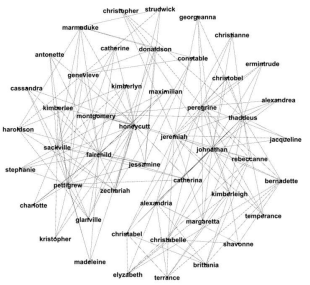
Our first observation is that dictionary words are less likely to have a low degree. Each individual word from a malicious dictionary is used to form a number of different domains, by combining it with other dictionary words. Therefore, from each G_i, we filter out all the nodes (words) with degree less than 3 (a value experimentally determined). With a high probability, a low degree node (word) is related to benign domains. We can also point out that word combinations in Dictionary-AGDs are algorithmically defined. This results in a more uniform graph structure. On the other hand, words from benign domains present less uniform patterns of connectivity in the word graph. To leverage this intuition we extract the connected components of each graph G_i. We expect that dictionaries from DGA algorithms will appear as such connected components.

Feature Vector Construction for Connected Components Let $G_i^{(1)}, \ldots, G_i^{(n)}$ be the connected components of word graph G_i. For each connected component $G_i^{(j)}$, $j \in \{1, \ldots, n\}$, we measure the following structural features (see Figure 2):

1. D_{mean}: Average vertex degree of $G_i^{(j)}$;
2. D_{max}: Maximum vertex degree of $G_i^{(j)}$;
3. C: Cardinality of cycle basis set of $G_i^{(j)}$;
4. C_V: $C/|V|$, where V is the set of vertices of $G_i^{(j)}$;
5. ASPL: Average shortest-path length of $G_i^{(j)}$.

Note that all steps above are done in a fully unsupervised fashion, i.e. without knowledge which domains in C are generated by a Dictionary-based DGA and which ones are not. We apply these preprocessing steps to the training data as well as to batches of new domain names observed during deployment.

Graph based Dictionary Finding Given a set of domains C_{Train}, we apply all the previous steps to C_{Train} and obtain all the connected components of all the graphs G_i derived from C_{Train}. We manually label every connected component in every graph G_i as DGA/non-DGA (indicated as True/False in Figure 2). Next we train a decision tree over the training dataset of labeled feature vectors.



ID	D_{mean}	D_{max}	C	C_V	ASPL	Label
ID_1	7.16	16.0	63	2.62	1.84	True
ID_2	6.91	16.0	60	2.50	1.86	True
...	...	...	...	...	...	
ID_N	3.54	80.7	20	1.7	3.78	False

Fig. 2: In order to classify word graph components as DGA/non-DGA, each graph component is represented as a vector of structural features. Each description vector is part of a dataset that describes the overall word graph.

The decision tree model is later used for classifying new vectors (connected components) without human intervention, even if these connected components stem from word graphs that originate from a completely different dictionary. Each connected component that is classified as DGA by the decision tree is subsequently converted into a dictionary in a straightforward manner, i.e. by treating each node of the connected component as a word in the dictionary. An overview of the WordGraph dictionary finding phase is presented in Figure 3.

Classification of Domain Names From the previous steps, we obtain a set of *detected dictionaries*, each one associated with a TLD. We flag a domain as malicious if it has at least two words from a same discovered dictionary, and it has the same TLD as the dictionary.

5 Experimental Methodology

We follow a similar approach as [10] and create an experimental setting with labeled *ground truth data* obtained from the DGArchive [15], which is a web database for DGA domains from various families, and from the Alexa top 1 million domains (a snapshot from 2016) [2]. The main goal of our experiments with labeled data is to compare our methodology with *state-of-the-art* techniques for DGA classification: classifiers based on human engineered lexical features, and classifiers based on deep neural networks. Moreover, we want to observe how robust these methods are to changes in the dictionary used for generating malicious domains. This is a question that has not been explored in the literature, to the best of our knowledge. Our method is based solely on structural features representing *how* the words from the dictionary are put together but not on the specific words themselves. Therefore, we expect our method to be robust against changes in the dictionary.

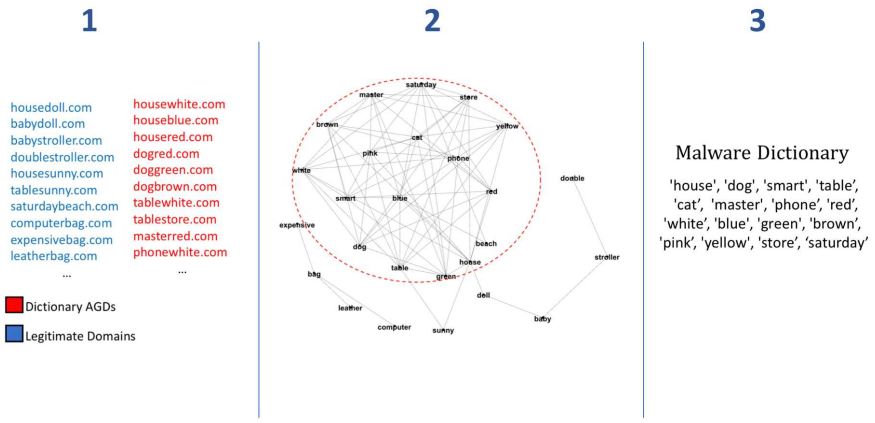


Fig. 3: An overview of the proposed WordGraph method. In **1** a dataset containing malicious and benign domains is analyzed, and frequent words are *learned* from the dataset. In **2** a word graph is built, and the structure of each graph component is analyzed to detect the malware dictionary **3**.

In more details, we analyze the performance of our WordGraph method and compare with classification models based on Random Forests and Deep Learning methods in three different settings:

- Training and testing datasets containing Dictionary-AGDs generated from the *same dictionary*. This experimental setting has been used in previous works [10, 20, 23].
- Testing datasets containing domains formed from *dictionaries that are distinct from the dictionaries used to generate the domains in the training dataset*. We want to evaluate the robustness of all models in a scenario where a botnet administrator wants to mislead a trained detection model by updating the dictionary in the malware code. This question has not been previously addressed in the literature.
- Small number of training samples. How well does each method perform when only a very small number of training samples is available?

We also evaluate the performance of the proposed WordGraph method when facing *real traffic data*. Many of the previous works are evaluated only in scenarios of synthetically created datasets. We show that the WordGraph model achieves similar performance when used for detecting Dictionary-AGDs in real DNS traffic, and moreover, it is able to detect new varieties of dictionaries and malware seeds due to its nature of pattern discovery.

5.1 Datasets

The evaluation of the proposed approach is conducted on datasets with ground truth labels and on real traffic unlabeled data.

	Train				Test-Familiar				Test-Unfamiliar			
Dataset	Alexa	WL1	WL2	WL3	Alexa	WL1	WL2	WL3	Alexa	WL1	WL2	WL3
Round 1	50K	20,768	20,768	0	30K	12K	12K	0	30K	0	0	32,768
Round 2	50K	0	20,768	20,768	30K	0	12K	12K	30K	32,768	0	0
Round 3	50K	20,768	0	20,768	30K	12K	0	12K	30K	0	32,768	0

Table 1: Description of the datasets used in two experiments: when the train and test data are both composed of AGDs generated from the same dictionaries (Test-Familiar), and when the train and test data are composed of AGDs generated from different dictionaries (Test-Unfamiliar).

Ground Truth Data The ground truth data contains 80,000 benign domain names randomly selected from the Alexa top 1M domains [2]. In our experiments, we 50,000 out of the 80,000 Alexa domain names for training, while the rest is reserved for testing. In addition, we use $3 \times 32,768$ AGDs obtained from the Dictionary-based DGA Suppobox [15], corresponding to three different dictionaries or wordlists, referred to as WL1, WL2, and WL3. How we split this malware data into portions for training and testing varies with the experiment.

Tables 1 provides an overview of the setup of two experiments involving the ground truth data.

- Test-Familiar: The test data consists of Dictionary-AGDs generated with the same dictionaries as the AGDs in the training data;
- Test-Unfamiliar: The test data consists of Dictionary-AGDs that were generated with a dictionary that was not known or available during training time. This experimental setting is intended to show that the model can be trained on a specific family and detect a distinct family, *unfamiliar* to the model.

Both experiments each consist of three rounds, corresponding to which wordlist is left out when training. For instance, as can be observed in Table 1, in round 1, Dictionary-AGDs generated with wordlist 3 do not appear in the training data.

	Train				Test-Imbalanced			
Dataset	Alexa	WL1	WL2	WL3	Alexa	WL1	WL2	WL3
Round 1	50,000	169	169	0	10,000	0	0	169
Round 2	50,000	0	169	169	10,000	169	0	0
Round 3	50,000	169	0	169	10,000	0	169	0

Table 2: Description of imbalanced datasets used for testing and training. The imbalance present in this data is very common in real traffic, where only a very small fraction of the data corresponds to malicious activity.

In an additional experiment, we measure the performance of all DGA domain detection methods in a scenario where very few samples of AGDs are available for training (see Table 2). The 507 AGDs involved in this experiment were selected from DGArchive; they were valid for one day only (Dec 1, 2016).

Real Traffic Data The data used in our real traffic experiments consists of a real time stream of passive analysis of DNS traffic, as in [5]. The traffic stems from approximately 10 billion DNS queries per day collected from multiple ISPs (Internet Service Providers), schools and businesses distributed all over the world. We collected 8 days of traffic from December 2016 to perform our experiments, from Dec 8 to Dec 15 (see Table 3). From the data, we keep only A and AAAA type DNS queries (i.e. IPv4 and IPv6 address records), and exclude all domains that receive less than 5 queries in a day.

	All Domains			Known AGDs (DGArchive)	
Dataset	Domains	Resolved	NX	Resolved	NX
Day 01	4,886,247	4,433,248	454,003	47	593
Day 02	4,922,618	4,532,932	390,735	67	673
Day 03	4,906,309	4,477,049	430,239	62	608
Day 04	4,350,224	3,981,514	369,673	87	662
Day 05	5,898,723	5,380,945	518,886	82	665
Day 06	5,425,651	4,963,786	463,584	73	680
Day 07	5,631,353	5,098,121	534,572	83	591
Day 08	5,254,954	4,747,867	508,319	95	635

Table 3: DNS traffic data description. We collected 8 days of real traffic data to measure the performance of our proposed WordGraph model. Days 1, 2 and 3 are used for model training, and days 4 through 8 are used for model testing.

The data stream consists of legitimate domains and malicious domains. All domains from this stream that are known to be Dictionary-based DGAs according to DGArchive [15] are marked as such. Although the number of unique Dictionary-based AGDs found in the real traffic data by cross-checking it against DGArchive is small, the total number of queries for such domains is tens of thousands per day. Furthermore, as will become clear in Section 6.2, the real traffic data contains more AGDs than those known in DGArchive.

5.2 Classification Models: Random Forest and Deep Learning

As stated in Section 1, the existing state-of-the-art approaches for classifying domain names as benign of malicious are either based on training a machine learning model with human defined lexical features that can be extracted from the domain name string, or on training a deep neural network that learns the features by itself. To show that the method that we propose in this paper outperforms the state-of-the-art, we include an experimental comparison with each kind of the existing approaches. For the approach based on human defined features, we train random forests (RFs) based on lexical features, extracted from each domain name string (see e.g. [24, 25]). Within supervised learning, tree ensemble

methods – such as random forests – are among the most common algorithms of choice for data scientists because of their general applicability and their state-of-the-art performance. Regarding the deep learning approach, a recent study [23] found no significant difference in predictive accuracy between recently proposed convolutional [16] and recurrent neural networks [10, 20] for the task of DGA detection, while the recurrent neural networks have a substantially higher training time. In our comparative overview we therefore use a convolutional neural network (CNN) architecture as in [16].

Data Preprocessing The strings that we give as input to all classifiers consist of a second level domain (SLD) and a top level domain (TLD), separated by a dot, as in e.g. *wikipedia.org*. As input to the CNN approach, we set the maximum length at 75 characters. The SLD label and the TLD label can in theory each be up to 63 characters long each. In practice they are typically shorter. If needed, we truncate domain names by removing characters from the end of the SLD until the desired length of 75 characters is reached. For domains whose length is less than 75, for the CNN approach, we pad with zeros on the left because the implementation of the deep neural network expects a fixed length input. For the RF and WordGraph approaches we do not do any padding. We convert each domain name string to lower case, since domain names are case insensitive.

Random Forest (RF) In each experiment, we train a random forest (RF) on the following 11 commonly used features, extracted from each domain name string: ent (normalized entropy of characters); nl2 (median of 2-gram); nl3 (median of 3-gram); naz (symbol character ratio); hex (hex character ratio); vwl (vowel character ratio); len (domain label length); gni (gini index of characters); cer (classification error of characters); tld (top level domain hash); dgt (first character digit). Each trained random forest consists of 100 trees. We refer to [22] for a detailed description of each of these features.

Deep Learning (CNN) In addition, in each experiment, following [16], we train a convolutional neural network that takes the raw domain name string as input. The neural network consists of an embedding layer, followed by a convolutional layer, two dense hidden layers, and an output layer. The role of the embedding layer is to learn to represent each character that can occur in a domain name by a 128-dimensional numerical vector, different from the original ASCII encoding. The embedding maps semantically similar characters to similar vectors, where the notion of similarity is implicitly derived (learned) based on the classification task at hand. The embedding layer is followed by a convolutional layer with 1024 filters, namely 256 filters for each of the sizes 2, 3, 4, and 5. During training of the network, each filter automatically learns which pattern it should look for. In this way, each of the filters learns to detect the soft presence of an interesting soft n-gram (with $n = 2, 3, 4, 5$). For each filter, the outcome of the detection phase is aggregated over the entire domain name string with the help of a pooling step. That means that the trained network is detecting the presence or absence of patterns in the domain names, without retaining any information on where

exactly in the domain name string these patterns occur. The output of the convolutional layer is consumed by two dense hidden layers, each with 1024 nodes, before reaching a single node output layer with sigmoid activation. In all experiments, we trained the deep neural networks for 20 epochs, with batch size 100 and learning rate 0.0001.

6 Results

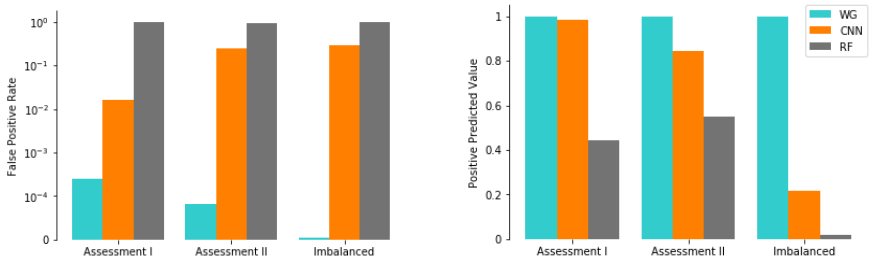
We report the results of all methods in terms of precision (positive predictive value, PPV), recall (true positive rate, TPR) and false positive rate (FPR). As usual, PPV = TP/(TP+FP), TPR = TP/(TP+FN) and FPR = FP/(FP+TN) where TP, FP, TN, and FN are the number of true positives, false positives, true negatives, and false negatives respectively. Blocking legitimate traffic is highly undesirable, therefore a low false positive rate is very important in deployed DGA detection systems. For parameter tuning purposes, in each experiment, we systematically split 10% of the training data off as validation data for the RF and CNN methods.

6.1 Experimental Results: Ground Truth Data

Figure 4 presents an overview of the results achieved by all models in the three different experimental settings with ground truth data described in Section 5.1. A first important result is that, across the board, the WordGraph method achieves a perfect TPR of 1, meaning that all Dictionary-AGDs are detected. To allow for a fair comparison, we selected classification thresholds for which the RF and CNN methods also achieve a TPR of 1. It is common for such a high TPR to be accompanied by a rise in FPR and a drop in PPV. As can be seen in Figure 4, this is most noticeable for the RF method, and, to a somewhat lesser extent for the CNN method. The WordGraph method on the other hand is barely impacted at all: it substantially outperforms the CNN and RF methods in all experiments.

Test-Familiar Experiment Detailed results for all methods in the "Test=Familiar" experiment are presented in Table 4. In this experiment, the train and test data contain AGDs generated from the same set of dictionaries. This experimental setup gives an advantage to classification models such as CNNs and Random Forests, since it allows the classification model to 'learn' characteristics of the words from the dictionaries.

A first observation from Table 4 is that the RF method does not do well at all. This is as expected: the lexical features extracted from the domain name strings to train the RFs have been designed to detect traditional DGAs, and Dictionary-based DGAs have been introduced with the exact purpose of evading such detection mechanisms. This is also apparent from the density plots of the features in Figure 5: the feature value distributions for the AGDs from WL1, WL2, and WL3 are very similar to those of the Alexa domain names, which explains the poor performance of the RF models that are based on these features.



(a) FPR Average (TPR=1) (b) PPV Average (TPR=1)

Fig. 4: Overview of FPR (lower is better) and PPV (higher is better) for all methods across the experimental setups on the ground truth data, for a fixed TPR=1. The WordGraph (WG) approach consistently achieves a very low FPR, of the order of magnitude of 10^{-4}, two order of magnitudes lower than the best FPR achieved by the CNN model.

The WordGraph method on the other hand works extremely well. It detects *all* AGDs in the test data in all rounds, while only misclassifying a very small number of benign domain names as malicious (namely 8/30,000 in round 1; 4/30,000 in round 2; and 1/30,000 in round 3). Finally, it is worth to call out that the CNN models have a very good performance as well. This is likely due to the fact that, as explained in Section 5.2 the CNN neural networks learn to detect the presence of interesting soft n-grams (with $n = 2, 3, 4, 5$), so, in a sense, they can memorize the dictionaries. It is interesting to point out that the CNN method performs consistently well throughout the rounds, i.e. given the fact that the dictionary was seen before by the model, there is no dictionary that is easier to 'learn'.

	Round 1		Round 2		Round 3	
Method	FPR	PPV	FPR	PPV	FPR	PPV
WordGraph	$2.67 \cdot 10^{-4}$	0.999	$1.33 \cdot 10^{-4}$	0.999	$3.33 \cdot 10^{-5}$	0.999
CNN	0.018	0.981	0.015	0.982	0.014	0.983
RF	1.0	0.444	1.0	0.444	1.0	0.444

Table 4: Results of random forest (RF), deep learning (CNN), and our proposed WordGraph approach (WG) on balanced ground truth data, for a fixed TPR=1. The AGDs in the training and test data are generated from the same dictionaries (**"Test-Familiar"**).

Test-Unfamiliar Experiment In the "Test-Unfamiliar" experiment, the trained models are tested on AGDs generated from dictionaries that were not seen dur-

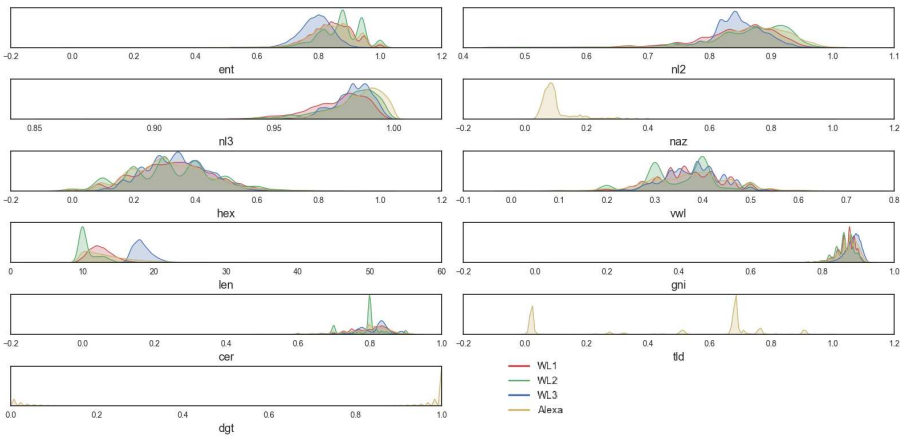


Fig. 5: Kernel density plot of the features extracted for the RF experiments where the x-axis represents the feature value and the area under the curve represents the probability of having a feature in that range of x values.

ing training. As Table 5 shows, the WordGraph method has no problem at all detecting these new AGDs across all rounds, while, as expected, the RF method continues to struggle. Interestingly enough, unlike in the previous experimental setting, the performance of the CNN model is no longer consistently good across all rounds. In round 1, the CNN model performs significantly worse when compared to rounds 2 and 3, having a FPR higher by one order of magnitude. In round 1 the model is trained on WL1 and WL2 DGA domains, while the testing data contains WL3 AGDs. A possible explanation for the poor performance of the CNN method in round 1 is that, as observed in Figure 5 for the 'len' feature, the AGDs in WL3 tend to be longer than those in WL1 and WL2, leading the CNN to misclassify the AGDs from WL3 as malicious because it has never seen such long malicious domain names during training.

	Round 1		Round 2		Round 3	
Method	FPR	PPV	FPR	PPV	FPR	PPV
WordGraph	0.0	1.0	$6.67 \cdot 10^{-5}$	0.999	$1.33 \cdot 10^{-4}$	0.999
CNN	0.589	0.650	0.084	0.928	0.050	0.956
RF	1.0	0.522	1.0	0.522	0.723	0.602

Table 5: Results of all methods on balanced ground truth data, for a fixed TPR=1. The AGDs in the test data are generated from dictionaries that were not available during training time (**"Test-Unfamiliar"**).

Results on Imbalanced Ground Truth Data The results for the models trained and tested on the imbalanced ground truth data are presented in Table 6. The trend we observed before persists: the WordGraph approach perfectly detects all Dictionary-AGDs in all rounds without misclassifying even a single benign domain name, the CNN method sometimes does well and sometimes not, and the RF method does not do well at all.

	Round 1		Round 2		Round 3	
Method	FPR	PPV	FPR	PPV	FPR	PPV
WordGraph	0.0	1.0	0.0	1.0	0.0	1.0
CNN	0.718	0.230	0.127	0.117	0.039	0.300
RF	1.0	0.017	1.0	0.017	1.0	0.017

Table 6: Results of all methods on imbalanced ground truth data, for a fixed TPR=1, the three rounds of experiment are listed in Table 2 (**"Test-Imbalanced"**).

Overall, the WordGraph method clearly outperforms the CNN and RF methods. It is able to 'find' dictionaries in data through graph analysis, even when only a small sample is available. The imbalanced experiment from Table 6 illustrates a scenario where only one infected machine is present in a network. After one day of Dictionary-AGD DNS requests, the WordGraph approach is able to extract the malicious dictionary from traffic. Out of 10,169 domains in each test dataset, only 169 domains are malicious, and the WordGraph method is able to identify 100% of the words from the malicious dictionary, with FPR=0.0 in every round of this experiment.

6.2 Experimental Results: Real DNS traffic

To study the performance of the WordGraph method in a realistic scenario, we evaluate it on the real traffic data from Table 3. To further reduce the dataset size, we used solely NX Traffic since most of AGD queries result in Non-Existent Domain (NXDomain) responses. This approach has been consistently used in the literature [3, 8].

The first three days in the real traffic dataset were used for training the graph component classifier as described in Section 4. The Dictionary-based AGDs in the dataset were labeled using DGA archive [15] as a source of ground truth labels. The number of Dictionary-based AGDs for each day is described in 3. We then evaluated the results of our method on the remaining five days of traffic.

Overall, we identified 81 dictionaries in five days of traffic. Fifteen of these discovered dictionaries are present in DGArchive. We manually verified the remaining dictionaries and confirmed they were all malicious. Since DGArchive has the complete dictionary for the 15 cases (by reverse engineering the malware) we managed to verify that our method recovered the complete dictionary

in this situation. We identified several dictionaries related to malware download hosts, such as *apartonly.gq*, *oftenthere.ga* and *quitethough.cf*. We also discovered variations of the malware family Suppobox, where the generated domains in this variation present TLD "*.ru*".

Once we obtained the dictionaries, we also flagged the domains that were generated by these words in the resolved DNS traffic, giving information on active C&C (Command and Control) centers.

7 Discussion and Limitations

Computational Complexity The most expensive part of our algorithm is the graph building with complexity $\mathcal{O}(n^2)$, where n is the number of words extracted from domains. We apply our algorithm only to NXDomains (since DGAs are mostly non-resolved) on a daily basis. The datasets have about 5 millions domains, where about 500,000 are NXdomains. The size of the graphs are about 60,000 nodes, which corresponds to the number of extracted words from traffic. The entire algorithm runs in about 30 minutes, considering word extraction and graph analysis phases. All experiments were run on a machine with an 2.3 GHz Intel Core i5 processor and a maximum of 16GB of allowed memory consumption.

Limitations (1) Our WordGraph method is very successful in extracting dictionaries from AGDs. This is the case for all variations of the Suppobox family, and other unidentified families that we were able to identify in real traffic. The method leverages the fact that such families uses limited dictionaries and the reuse of dictionary words is frequent. We suggest as future work the investigation of Dictionary-based AGDs that utilizes a very large dictionary and very low word reuse rate. (2) Additionally, there are malware families, such as Matsnu, that use a DGA as a secondary resource for C&C communication, with hardcoded domains being the primary method. Such a malware family typically not only generates a very small number of DGA domains daily [18], but those domains are only queried in the case that all domains from the hardcoded list receive an NX response. Matsnu malware for instance generates only four DGA domains per day. In our real traffic dataset, we did not encounter any occurrences of Matsnu AGDs. Preliminary results on ground truth data indicate that the WordGraph method would need more than one month of observation to be able to recover the dictionary. Using one month of Matsnu AGDs from DGAarchive, we were able to extract from 590 domains a partial dictionary of 82 words, leading to the detection of 273 domains. (3) Once the dictionaries are extracted, we detect malicious domains by checking if they have two or more words from the extracted dictionaries. False positive rates were at most 10^{-4} in all the performed experiments (with real traffic and synthetic data). An adversary could, in principle, try to increase the false positive rate by using dictionaries with words commonly used in legitimate domains. However, this approach has a drawback for the adversary since several of the generated domains will already be registered and thus useless for the bot-master.

8 Conclusion

We proposed a novel WordGraph method for detection of Dictionary-based DGAs in DNS traffic. The WordGraph method consists of two main phases: (1) malicious dictionary extraction from traffic observations and (2) detection of Dictionary-AGDs present in traffic. We evaluated WordGraph on ground truth data consisting of Dictionary-AGDs from DGArchive and benign domains from Alexa. Our experiments show that WordGraph consistently outperforms random forests based on human defined lexical features as well as deep learning models that take the raw domain name string as input and learn the features themselves. In particular, unlike these existing state-of-the-art methods, WordGraph detects (nearly) all Dictionary-AGDs even when the dictionary used to generate them is changed. Furthermore, when we analyzed 5 days of real traffic from multiple ISPs with WordGraph, we were able to detect the presence of Dictionary-AGDs generated by known as well as by previously unknown malware, and we discovered domains related to C&C proxies that received thousands of requests. Due to its nature of discovering, through a graph perspective, malicious patterns of words in traffic, the WordGraph method guarantees a very low false positive rate, presenting itself as a DGA detection system with practical relevance.

References

1. Jasper Abbink and Christian Doerr. Popularity-based detection of domain generation algorithms. In *Proc. of the 12th International Conference on Availability, Reliability and Security*, page 79. ACM, 2017.
2. ALEXA. Top sites on the web. `http://alexa.com/topsites`, 2017.
3. Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of DGA-based malware. In *21st USENIX Security Symposium*, pages 24–24, 2012.
4. Thomas Barabosch, Andre Wichmann, Felix Leder, and Elmar Gerhards-Padilla. Automatic extraction of domain name generation algorithms from current malware. In *Proc. of NATO Symposium IST-111 on Information Assurance and Cyber Defense*, 2012.
5. Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive DNS analysis. In *Ndss*, 2011.
6. Reinhard Diestel. *Graph Theory*, volume 137 of *Graduate Texts in Mathematics*. Springer, 2005.
7. Jason Geffner. End-to-end analysis of a domain generating algorithm malware family. *Black Hat USA*, 2013, 2013.
8. Srinivas Krishnan, Teryl Taylor, Fabian Monrose, and John McHugh. Crossing the threshold: Detecting network malfeasance via sequential hypothesis testing. In *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2013.
9. Pedro G Lind, Marta C Gonzalez, and Hans J Herrmann. Cycles and clustering in bipartite networks. *Physical review E*, 72(5):056127, 2005.
10. Pierre Lison and Vasileios Mavroeidis. Automatic detection of malware-generated domains with recurrent neural models. *arXiv:1709.07102*, 2017.

11. Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1245–1254, 2009.
12. Guoyong Mao and Ning Zhang. Analysis of average shortest-path length of scale-free network. *Journal of Applied Mathematics*, http://dx.doi.org/10.1155/2013/865643, 2013.
13. D. Kevin McGrath and Minaxi Gupta. Behind phishing: An examination of phisher modi operandi. *LEET*, 8:4, 2008.
14. Miranda Mowbray and Josiah Hagen. Finding domain-generation algorithms by looking at length distribution. In *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops*, pages 395–400, 2014.
15. Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In *25th USENIX Security Symposium*, pages 263–278, 2016.
16. Joshua Saxe and Konstantin Berlin. eXpose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv:1702.08568*, 2017.
17. Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: DGA-based botnet tracking and intelligence. In *Proc. of DIMVA 2014, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 192–211, 2014.
18. Stanislav Skuratovich. Matsnu technical report. `http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/`, 2015.
19. Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen. A LSTM based framework for handling multiclass imbalance in DGA botnet detection. *Neurocomputing*, 275:2401–2413, 2018.
20. Jonathan Woodbridge, Hyrum S Anderson, Anjum Ahuja, and Daniel Grant. Predicting domain generation algorithms with long short-term memory networks. *arXiv:1611.00791*, 2016.
21. Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement*, pages 48–61, 2010.
22. Bin Yu, Daniel Gray, Jie Pan, Martine De Cock, and Anderson Nascimento. Inline DGA detection with deep networks. In *Data Mining for Cyber Security, Proc. of International Conference on Data Mining (ICDM2017) Workshops*, pages 683–692, 2017.
23. Bin Yu, Jie Pan, Jiaming Hu, Anderson Nascimento, and Martine De Cock. Character level based detection of DGA domain names. In *Proc. of IJCNN at WCCI2018 (2018 IEEE World Congress on Computational Intelligence)*, 2018.
24. Bin Yu, Les Smith, and Mark Threefoot. Semi-supervised time series modeling for real-time flux domain detection on passive DNS traffic. In *Proc. of the 10th International Conference on Machine Learning and Data Mining*, pages 258–271, 2014.
25. Bin Yu, Les Smith, Mark Threefoot, and Femi Olumofin. Behavior analysis based DNS tunneling detection with big data technologies. In *Proc. of the International Conference on Internet of Things and Big Data*, pages 284–290, 2016.