DBD: Deep Learning DGA-Based Botnet Detection

Chapter · August 2019

DOI: 10.1007/978-3-030-13057-2_6

5 authors, including:

Vinayakumar Ravi
Prince Mohammad University
294 PUBLICATIONS **6,314** CITATIONS
SEE PROFILE

Soman Kp
Amrita Vishwa Vidyapeetham
780 PUBLICATIONS **11,635** CITATIONS
SEE PROFILE

Prabaharan Poornachandran
Amrita Vishwa Vidyapeetham
109 PUBLICATIONS **3,617** CITATIONS
SEE PROFILE

Mamoun Alazab
Charles Darwin University
333 PUBLICATIONS **11,853** CITATIONS
SEE PROFILE

DBD: Deep Learning DGA-based Botnet Detection

Vinayakumar R, Soman KP, Prabaharan Poornachandran, Mamoun Alazab, and Alireza Jolfaei

Abstract Botnets play an important role in malware distribution and they are widely used for spreading malicious activities in the Internet. The study of the literature shows that a large subset of botnets use DNS poisoning to spread out malicious activities and that there are various methods for their detection using DNS queries. However, since botnets generate domain names quite frequently, the resolution of domain names can be very time consuming. Hence, the detection of botnets can be extremely difficult. This chapter propose a novel deep learning framework to detect malicious domains generated by malicious Domain Generation Algorithms (DGA). The proposed DGA detection method, named, Deep Bot Detect (DBD) is able to evaluate data from large scale networks without reverse engineering or performing Non-Existent Domain (NXDomain) inspection. The framework analyzes domain names and categorizes them using statistical features, which are extracted implicitly through deep learning architectures. The framework is tested and deployed in our lab environment. The experimental results demonstrate the effectiveness of the proposed framework and shows that the proposed method has high accuracy and low false-positive rates. The proposed framework is a simple architecture that contains fewer learnable parameters compared to other character-based, short text classification models. Therefore, the proposed framework is faster to train and is less prone to over-fitting. The framework provides an early detection mechanism for the identification of Domain-Flux botnets propagating in a network and it helps keep the Internet clean from related malicious activities.

Vinayakumar R and Soman KP
Center for Computational Engineering and Networking (CEN), Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India,
e-mail: vinayakumarr77@gmail.com

Prabaharan Poornachandran
Centre for Cyber Security Systems and Networks, Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, India

Mamoun Alazab
Charles Darwin University, Australia
e-mail: mamoun.alazab@cdu.edu.au

Alireza Jolfaei
Federation University Australia
e-mail: a.jolfaei@federation.edu.au

Key words: Botnet, Deep Learning, Domain Name Generation, Malware, Cybercrime, Cyber Security.

1 Introduction

Over the past decade, the number of malicious activities have significantly increased [1–6, 13]. Today, perpetrators use malicious codes to infect a large collection of computers, named, botnet, and use a Command and Control (C&C) server, named, botmaster, to remotely control botnets for running malicious activities, such as Distributed Denial of Service (DDoS) attacks, spam emails, scareware, and spyware. A possible mitigation strategy is to blacklist the botmaster, to whom the botnets try to contact. However, to evade being blacklisted, perpetrators change their domain names frequently through Domain Name System (DNS) poisoning [7]. A commonly used method for C&C location resolution and fail-over resilience is fluxing, which mainly has two types: IP flux and Domain-Flux [8]. In this work, we mainly focus on Domain-Flux services, through which the botmaster changes the domain name that has to be mapped into the IP address of the C&C server frequently. This is done using a Domain Generation Algorithm (DGA), which generates domain names randomly on a large scale for registration. Botnets use various DGAs for domain name generation. For example, Conficker, Torpig, Kraken, and Murofet [9].

A method to combat DGA generated malware is to monitor DNS requests and detect DGA domain names using a machine learning classification algorithm. The past works on DGA detection can be grouped into two categories: retrospective and real-time detection. Retrospective detection cannot be used in real-time, because they model a large volume of predictions over a large volume of domains [10]. These methods cluster domain names using various statistical tests, such as Kullback-Leibler divergence testing [12], and contextual information, such as HTTP headers to further improve the performance. These methods are computationally expensive. On the contrary, real-time detection methods make the classifications using domain names without any additional contextual information. Although such approaches are fast, they cannot perform as accurate as the retrospective methods.

The existing real-time based methods for DGA detection are based on feature engineering and the commonly used features are entropy, string length, vowel to consonant ratio, and n-gram. Features are fed into the machine learning classifier for classification (the commonly used classifier is the random forest). In [10], the authors proposed a retrospective approach for DGA detection based on automatic feature extraction. Recently, the application of deep learning architectures, such as a Long Short Term Memory (LSTM) method, has been considered for DGA analysis [11]. Such approaches outperform the classical machine learning classification in terms of accuracy.

In this chapter, we propose a scalable deep learning DGA-based botnet identification framework, named, DBD, which uses a Convolutional Neural Network with a Long Short Term Memory (CNN-LSTM) pipeline to detect DGA domain

names. Along with DBD, different character-based, short-text classification models are evaluated over a large labeled data set, which consists of a collection of algorithmically generated and benign domains gathered from our lab based network activities. Experimental results confirm the effectiveness of our framework in identifying DGA-based botnets, as well as the compromised systems within the deployed network. DBD utilizes TensorFlow [14], that is, a distributed framework for learning very large scale deep neural networks. Once the framework classifies a particular domain as a DGA domain, the framework assumes that the host is infected and analyses its resolved domain list for finding the corresponding C&C server.

The remainder of the chapter is organized as follows: Section 2 describes the background information on DNS, Keras embedding, and deep learning architecture. Section 3 describes the past literature on DGA detection. Section 4 proposes a scalable deep learning DGA-based botnet identification framework. Section 5 elaborates on the experiments and observations. Finally, Section 6 concludes the chapter.

2 Background and Preliminaries

2.1 Botnet

A botnet is a network of compromised Internet connected computers that are ruled by a bot master remotely through the command and control (C&C) channel. An infected computer in a network is named bot. A bot master uses the bot to host malicious activities. Mostly, the size or structure of botnet can vary. However, they follow the same stages in their lifecycle [15]. To adapt to the new technologies, the botnet evolves with the bot master. The C&C channel provides a communication point between the bot master and the bot to transmit data among them. The C&C communications are as follows:

- A bot master contacts botnets by issuing a command;
- Based on the command, a botnet performs its activities; and
- Botnet forwards the results of performed activities to the bot master.

Botnet can be detected only when the location of the C&C server is identified. In addition, the bot master will not be able to control the botnet without establishing a reliable connection between C&C servers and compromised computers. Primarily, there are three types of C&C architectures based on the way the communication is used, that is, centralized, decentralized, and hybrid. In a centralized botnet architecture, the bot master manages all the active bots in a botnet from a centralized C&C server. In other words, the bot communicates with a C&C server for all operations, such as receiving and answering commands. The centralized botnet architecture uses a hierarchical, star topology. The Internet Relay Chat (IRC) and Hyper Text Transfer Protocol (HTTP) are key protocols in a centralized architecture. This type of architecture is easier to control, since it has only one central

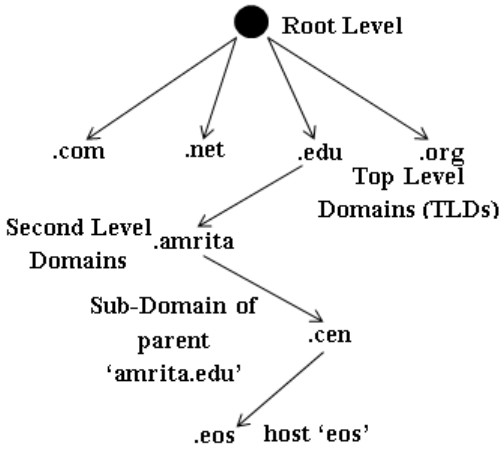


Fig. 1: Hierarchical domain name system

point. However, the design of a centralized architecture is complex. Also, message latency and survivability are short and chances of failure is more compared to other architectures. Decentralized architectures or peer-to-peer architectures contain more than one C&C servers to manage all active bots in a botnet. Each bot acts as a C&C server as well as clients. The detection of the botnet which uses a decentralized architecture is a difficult task, as compared to the centralized architecture. This is due to the use of peer-to-peer protocols. The design of decentralized architecture is complex, while its message latency and survivability is higher and the chances of failure is less compared to other centralized architectures. Hybrid architecture is a combination of centralized and decentralized architecture. The detection and monitoring tasks in a hybrid architecture is more difficult than centralized and decentralized architectures. However, the design of the hybrid architecture is simpler compared to other architectures.

2.2 Domain Name System

Domain Name System (DNS) is an application protocol in the Internet infrastructure, which has a distributed data base that cross-references domain names to their corresponding Internet Protocol (IP) addresses and vice versa. DNS maintains a hierarchy to control its distributed data base, and it is comprised of a root level, top level domains, second level domains, sub-domains, and hosts. Domain names consist of a Top Level Domain (TLD) and a Second Level Domain (SLD), which are separated by a dot point. For example, in YouTube.com, com is a TLD and YouTube is a SLD. The DNS hierarchy is divided into zones which are controlled by a zone

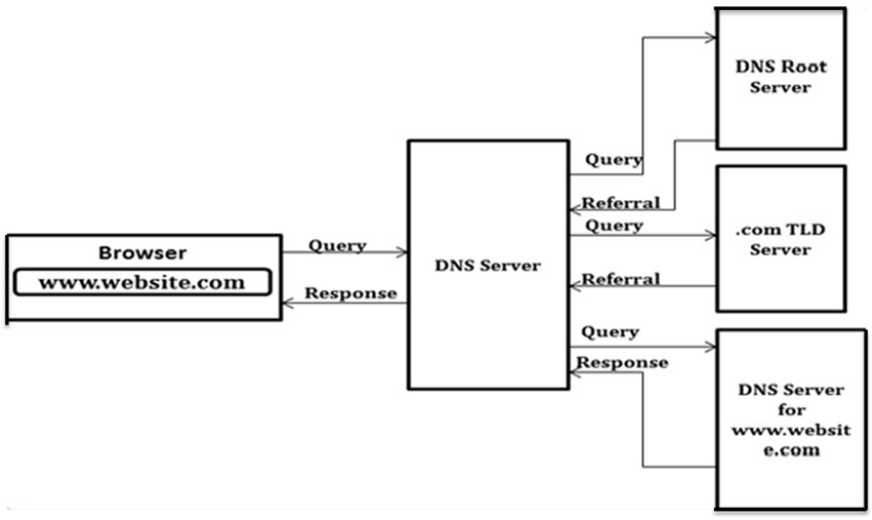


Fig. 2: Recursive DNS query

manager. Each node in the DNS hierarchy has a label, which includes information related with the domain name.

A hierarchy of DNS is named domain namespace, which is similar to an eDirectory with an inverted tree structure. Organizations can use their own domain namespace to create private networks. These private networks are not visible on the Internet. A simple DNS hierarchy is shown in Fig. 1. A domain name composed of one or more sections or sub domains is named a label. A path from a sub-domain to a root is named a fully qualified domain name. A DNS hierarchy can have a maximum number of 127 levels. These labels are separated by dots. Each label can contain 0 to 63 characters. A label of length 0 is assigned to the root zone. The full domain name length can contain 253 characters. A domain name redirects the end-user request to a particular source where the information exists in the DNS name space. All the information of domain names is stored in the DNS server in the form of a record, named, resource records.

Recursive and non-recursive (or iterative) are two types of DNS servers. Non-Recursive DNS servers act as the Start of Authority (SoA); they answer queries inside the governed domains without querying other DNS servers even if the Non-Recursive DNS server cannot provide the requested answer. Recursive DNS servers, an example of which is shown in Fig. 2, respond to queries of all types of domains, by querying other servers and passing the response back to the client. Recursive DNS servers usually suffer from DDoS attacks, DNS cache poisoning, unauthorized use of resources, and root name server performance degradation. Furthermore, query to DNS servers are usually not encrypted [16]; thus, it is possible to detect the websites that are being browsed.

2.3 Domain-Flux and Domain Generation Algorithms

Domain fluxing is a technique in which fully qualified domain names linked with the IP address of C&C servers are frequently changed to keep the botnets in operation. To carry out this operation, the bot master makes use of a Domain Generation Algorithm (DGA) to generate domain names on a large scale, which can bypass blacklisting and heuristics methods for detecting DGA domain names. Fig. 3 shows the flow diagram of domain-flux attacks. The bot uses two domains, abc.com and def.com. The domain abc.com is not registered and receives an NXDomain response from the DNS server. def.com is registered and hence it is able to contact the C&C server. In Domain-Flux attacks, botnets contact their botmaster with randomly generated domain names using a hit and trial method. In most cases, this would generate several Non-Existent (NX) response queries for domains which have no IP addresses (NXDomains). Monitoring DNS traffic is an important task and it helps to detect the botnet using a DNS analysis. This method is more efficient than the static and binary analysis of malware. This is mainly because these methods require much less time to reverse engineer the binaries to assign a signature [37].

Analysis of DNS queries helps to detect DGA generated domains and it helps to trace the botnet and block the communication point between the botmaster and C&C server in a timely manner. Detection of DGA generated domain names through analysis of DNS data has various advantages. For example, the DNS protocol includes only a small amount of traffic in the entire network, which makes it appropriate for analysis even in large scale networks. In addition, the DNS traffic is normally cached which reduces the traffic. Some of the domain features, such as the Autonomous System (AS) number and the domain owner, can be added to DNS traffic features to further increase the detection rate of DGA domain names. Furthermore, the analysis of DNS queries helps to identify attacks in early stages or even before they occur.

A DGA generates a large number of domain names using a random seed, which can be a date, a number, or any random characters. To construct domain names, the DGA generator uses a combination of bitshift, xor, divide, multiply, and modulo operations to generate a sequence of characters, which follow a certain distribution, such as a normal or a uniform distribution model. DGA generators normally use different seeds quit frequently. For instance, the seeds may change within a period of one day. This makes the blacklisting strategies inefficient since DGAs keep creating differing domain names. However, DGA generated domain names contain unique statistical properties, which are different from legitimate domain names. As explained in Section 4, our proposed detection method would utilize these properties to detect DGA domain names.

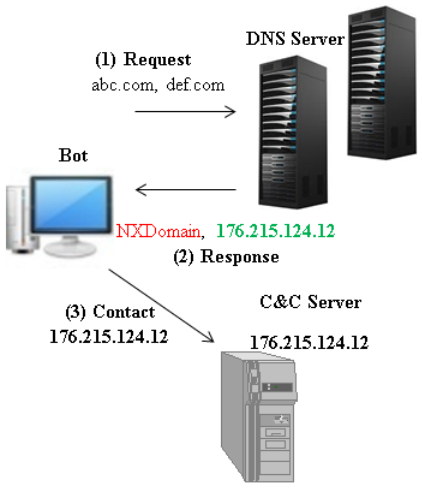


Fig. 3: Diagram of domain-flux attacks

2.4 *Keras embedding and Convolutional Neural Network with Long Short-Term Memory*

Keras embedding converts positive integers (indexes) into continuous or dense vector representations of fixed sizes. These indexes direct to a particular character in a vocabulary. Dense vector representation is a character embedding matrix which is calculated using

$$(nb\_char, vocab\_size) \times (vocab\_size, embed\_dim) = (nb\_char, embed\_dim), \quad (1)$$

where nb_char denotes the number of unique characters, $vocab\_size$ denotes the size of vocabulary, and $embed\_dim$ denotes the size of the vector space in which characters will be embedded. The resultant matrix of the embedding layer are of the shape $(vocab\_size, embed\_dim)$ which are passed as input to the deep learning architectures. The weights in embedding layer are initialized using a Gaussian distribution. The Keras embedding layer collaboratively works with the other deep learning layers to optimize the weights during the back-propagation.

Given a matrix representation M of Keras embedding, the CNN captures the features from the character representation. CNN is composed of a 1D convolution, a pooling, and a fully connected layer. The 1D convolution contains a filter that slides over a sequence and detects features from different locations. To reduce the feature dimension, maxpooling 1D is applied. These new features are fed into LSTM. This procedure is mathematically represented as

$$H^{CNN} = CNN_{char}(M), \quad (2)$$

$$h_t = LSTM(e_t, h_{t-1}). \tag{3}$$

where $e = H^{CNN}$, each row is a time step for LSTM, h_t is a hidden layer representation in time step t. LSTM works on each row of H^{CNN} with hidden vectors from previous time-step to produce embedding for the subsequent time-steps. Finally, the successive feature representation of LSTM is passed into the fully connected layer, which contains a non-linear activation function that gives 0 or 1. The output 0 indicates a legitimate domain and the output 1 indicates a DGA generated domain name.

3 Related Works

Pleiades designed the first method for detecting DGA-based domains, which does not require the reverse engineering of the bot malware [10] . In [18], Woodbridge et al. applied a Long Short-Term Memory (LSTM) method for detection and classification of DGAs. Woodbridge et al.'s LSTM network is composed of an embedding layer, an LSTM layer and a logistic regression. Woodbridge et al. compared their detection method with a random forest classifier and a feature-less hidden markov model [10]. In all experiments, the LSTM method outperformed in comparison to the hand-crafted features and featureless methods. In [19], the authors discussed various issues, including the accuracy and the false alarm rate of the existing DGA detection methods, and proposed a new way to label the DNS traffic data. They used this labeled data to evaluate the performance of featureless methods CNN and LSTM with Keras embedding and hand crafted features with conventional machine learning classifiers. In [20], the authors studied the efficiency of RNN on a large number of malware samples with a total of 61 malware families.

In [21], the authors discussed the performance of deep learning architectures for DGA detection and categorization on a combination of data set, which was collected from publically available data sets and real time OSINT DGA feeds. They showed detailed experimental analysis of each of the deep learning architectures. Their experiments showed that deep learning architectures perform better compared to random forest classifier. In [22], the authors discussed methods of collecting DNS logs inside Ethernet LAN. They evaluated the performance of recurrent structures using two different data sets. One data was collected from public sources and the other was collected from the real-time DNS traffic. This was done to effectively find out a robust machine learning model, which can be deployed in real time to monitor the DNS traffic. In [23], the authors proposed a scalable framework which provides a method to collect DNS traffic data at the ISP level.

In [24], the authors proposed a unique deep learning architecture, which can detect both DGA and malicious Uniform Resource Locator (URL). The unique architecture proposed in [24] has been evaluated using DNS, URL and Email data analysis for various types of testing data sets [25]. The results are compared with

Table 1: A short review on deep learning based DGA domain detection

Reference	Method	Data source	Text representation
Woodbridge et al. [18]	LSTM	*legitimate:* One million domains names from Alexa *DGA domains:* OSINT DGA feed from Bambenek Consulting	Keras embedding
Yu et al. [19]	CNN and LSTM	*legitimate:* One million domains names from Alexa and Farsight Security DNS data stream *DGA domains:* DGArchive and Farsight Security DNS stream	Keras embedding
Lison et al. [20]	Architectures related to recurrent structure	*legitimate:* One million domains names from Alexa, Statvoo, and Cisco *DGA domains:* DGArchive and Bambenek	Keras embedding and one-hot encoding
Vinayakumar et al. [21]	CNN and recurrent structures	*legitimate:* One million domains names from Alexa and OpenDNS *DGA domains:* OSINT DGA feeds	Keras embedding
Vinayakumar et al. [22]	RNN and LSTM	*legitimate:* One million domains names from Alexa, OpenDNS, and Ethernet LAN DNS stream *DGA domains:* OSINT DGA feeds and Ethernet LAN DNS stream	Keras Embedding
Vinayakumar et al. [23]	CNN and recurrent structures	*legitimate:* One million domains names from Alexa and real time DNS stream from the Internet Service Provider (ISP) level *DGA domains:* real time DNS stream from the ISP level	Keras embedding
Yu et al. [31]	CNN, RNN, LSTM, CNN and RNN, CNN and LSTM, Bidirectional RNN, Bidirectional GRU	*legitimate:* One million domains names from Alexa *DGA domains:* Bambenek	Keras embedding and One-hot encoding
Mohan et al. [24]	CNN and CNN-LSTM	*legitimate:* One million domains names from Alexa and OpenDNS *DGA domains:* OSINT DGA feeds	Keras embedding
Vinayakumar et al. [25]	CNN-LSTM, CNN and CNN-RNN	*legitimate:* One million domains names from Alexa and OpenDNS *DGA domains:* publically available DGA algorithms, 360-DGA, OSINT DGA feeds, DGArchieve and Ethernet LAN DNS stream	Keras embedding
Mac et al [30]	LSTM, CNN and LSTM, Bidirectional LSTM	*legitimate:* One million domains names from Alexa *DGA domains:* Bambenek	Keras embedding and One-hot encoding
Curtin et al. [27]	RNN, LSTM, and generalized likelihood ratio test	*legitimate:* One million domains names from Alexa and OpenDNS *DGA domains:* simulated domain names from publically available DGA algorithms, DGArchive, Andrey Abakumov's DGA repository on Github, Johannes Bader's DGA implementations, and Sinkholed domains collected from public WHOIS registration	One-hot encoding and TF-IDF
Tran et al. [28]	Cost-sensitive LSTM	*legitimate:* One million domains names from Alexa *DGA domains:* OSINT DGA feed from Bambenek Consulting	Keras embedding

other deep learning architectures and logistic regression with bigram text representation. [27] proposed an RNN based approach for DGA domain detection. The approach uses a smashword score which measures the closeness in respect to English words. [28] discussed the importance of cost-sensitive approach along with LSTM for DGA botnet detection.

In [29], the authors applied various ImageNet models for DGA analysis. They described how to transform the pre-trained weights of the ImageNet database and make the database usable for DGA data. In [30], Mac et al. studied various conventional machine learning classifiers, deep learning architectures, and the combination of deep learning architectures and conventional machine learning for DGA analysis. In a hybrid network, deep learning architecture obtained optimal features and the conventional machine learning was used for classification. In [31], Yu et al. discussed the efficiency of various character-based models for DGA analysis. The models are Endgame [18], Invincea [32], CMU [33], MIT [35], NYU [36].

To model short character strings (URLs, file paths, or registry keys), Saxe and Berlin proposed a three-layer CNN network with Keras embedding [32]. Saxe and Berlin method contains 1024 neurons and it uses a *ReLU* activation function. To speed up the model training and prevent from over fitting, Vosoughi et al. used batch normalization and regularization techniques. In [33], Dhingra et al. proposed a representation and classification method for Tweeter posts. Dhingra et al. used a bidirectional Gated Recurrent Unit (GRU) to learn feature representation of Twitter data. The tweets are tokenized into a stream of characters and each character is represented using one hot character encoding. These one-hot representations are mapped into a character space and are passed into the bidirectional GRU. It contains both forward and backward GRUs, which facilitate the learning process of the sequence of characters in the domain name. A $softmax$ non-linear activation function is used for tweet classification. In [36], Zhang et al. proposed the NYU model for tweet classification. They combined a stack of CNN layers with a LSTM layer with word-based text representation. Since stacked CNN layers would impose over-fitting, Zhang et al. suggested the use of a minimum number of CNN layers, that is, one. They evaluated the efficacy of deep learning models using the conventional text representation methods, such as bag-of-words, ngrams with Term Frequency-Inverse Document Frequency (TF-IDF). In [35] Vosoughi et al. proposed a novel tweet representation based on a Convolutional Neural Network (CNN), that is, commonly used in the field of image processing [34], LSTM and hybris of CNN and LSTM.

Details of various character-based, short-text classification algorithms are given in Table 2. All algorithms use a character-level embedding as their first layer to transform the domain names into numeric vectors, and this is followed by a deep learning architecture for optimal feature extraction and classification. Details of deep learning based DGA detection algorithms are given in Table 1.

Table 2: Well-known short text classification methods

Name	**Architecture**	**Task**
CMU [33]	Bi-directional gated recurrent unit layer	Social media text classification
MIT [35]	Hybrid of stacked CNN layers and LSTM layer	Social media text classification
NYU [36]	Stacked CNN layers	Text classification

Table 3: Detailed configuration of DBD

Layer (type)	**Output Shape**	**Param #**
Embedding	(None, 91, 128)	6,528
Conv1D	(None, 8,764)	41,024
MaxPooling1	(None, 2,164)	0
LSTM	(None, 70)	37,800
Dense	(None, 1)	71
Activation	(None, 1)	0
Total params: 85,423 Trainable params: 85,423 Non-trainable params: 0		

4 Proposed DGA Detection: Deep Bot Detect

Since DNS traffic is not encrypted, an analysis of DNS traffic would reveal botnets that utilize DGAs. Generally, the presence of DGAs can be detected through

- Analysis of Non-Existent (NX) domain DNS response;
- Analysis of statistical features of DNS queries;
- Analysis of statistical features of DNS responses; and
- Time based analysis of DNS traffic.

The proposed DGA detection method, named, Deep Bot Detect (DBD), is based on an analysis of statistical features of DNS queries. This includes a preprocessing phase, an optimal features extraction phase, and a classification phase. In the preprocessing phase, the feature vectors are extracted from domain names using text representation methods, and then, the optimal features are computed from the numeric vectors using deep learning architectures. Finally, a classification is done using a fully connected layer with a non-linear activation function.

The proposed method converts domain names into lower-case, character strings, because domain names are case-insensitive and differentiating between capital and small letters may cause regularization issues; otherwise, the method has to be trained for more numbers of epochs to learn about differing patterns of the domain names.

The detailed configuration of DBD is listed in Table 3, which looks similar to that of [24] and [25]. The proposed method has less number of trainable parameters

Table 4: Detailed configuration of DNN

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	128,128
batch_normalization_1 (Batch)	(None, 128)	512
activation_1 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 96)	12,384
batch_normalization_2 (Batch)	(None, 96)	384
activation_2 (Activation)	(None, 96)	0
dropout_2 (Dropout)	(None, 96)	0
dense_3 (Dense)	(None, 64)	6,208
batch_normalization_3 (Batch)	(None, 64)	256
activation_3 (Activation)	(None, 64)	0
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2,080
batch_normalization_4 (Batch)	(None, 32)	128
activation_4 (Activation)	(None, 32)	0
dropout_4 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 16)	528
batch_normalization_5 (Batch)	(None, 16)	64
activation_5 (Activation)	(None, 16)	0
dropout_5 (Dropout)	(None, 16)	0
dense_6 (Dense)	(None, 1)	17
Total params: 150,689 Trainable params: 150,017 Non-trainable params: 672		

in comparison with other character based short text classification models. Thus, the DBD framework is faster to train and less prone to overfitting.

The corpus contains 39 unique characters: -, ., _, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The maximum length of the domain name is 91. If the length of the domain name is less than 91, it is padded with zeros. In DBD, the vector size of Keras embedding is 128, and each character is mapped into a 1×128 vector. This is different from ASCII encoding, and it helps to learn the similarity among characters by mapping the semantics of similar characters to similar vectors. The keras embedding outputs a matrix representation 39×128, where the number of unique characters is 39 and the dimension of embedding space 128. The Keras embedding jointly works with other layer in the deep learning model and obtains an optimal feature representation that composed of similar characters being embedded closer to each other.

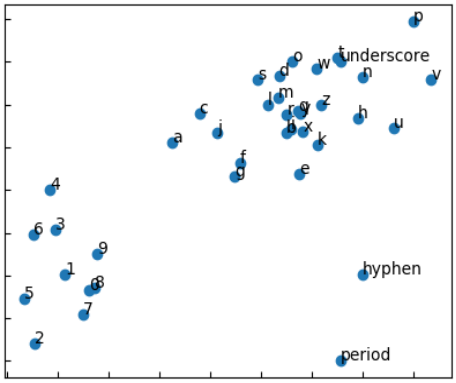


Fig. 4: Keras embedding character vector visualization of DBD model

To visualize, the 128-dimensional representation is fed into a t-SNE [26] that converts it to a two-dimensional representation, as shown in Fig. 4. The numbers, characters and special characters are appeared in their own cluster. This indicates that the model has captured the unique features to differentiate characters, numbers and special characters.

To conduct a fair comparative evaluation strategy, for all models under study, we considered Keras embedding as the domain name representation method and we set the embedding size to 128. To evaluate the effectiveness of Keras embedding, we used a 3-gram text representation method to mapped domain names. We hashed the features of 3 grams into a vector of length $1\times1{,}000$. These vectors are passed into a DNN for optimal feature extraction and classification. The detailed configuration is listed in Table 4. To avoid overfitting and to speed up the training process, dropout and batch normalization is used, respectively, in between the DNN layers. We found both dropout and batch normalization is use full. Following the embedding layer, DBD contains a convolution layer, which contains 64 filters of length 5 with an *ReLU* activation function. The Convolutional layer includes a maxpooling with a pool length of 4 and an LSTM with 70 memory blocks. This type of hybrid CNN-LSTM model learns the important features and forms a fixed length feature vector representation. Finally, the 128 vector representation is passed to the fully connected layer that contains a *sigmoid* non-linear activation function, which outputs 0 for legitimate and 1 for DGA domain names. To minimize the loss during backpropogation, all deep learning architectures use a binary cross entropy loss function, defined as

$$loss(pd,ed) = -\frac{1}{N}\sum_{i=1}^{N}[ed_i \log pd_i + (1-ed_i)\log(1-pd_i)], \tag{4}$$

where pd is a vector of predicted class label in testing data set, ed is a vector of expected class label, and N is the number of samples.

Table 5: Detailed statistics of the data sets

Data set	Legitimate	DGA generated	Total	Observation Dates
Data set 1 (Train)	155,683	115,056	270,739	May 1, 2017 - Oct. 15, 2018
Data set 1 (Test 1)	181,837	37,742	219,579	Jan. 1, 2018 - Mar. 25, 2018
Data set 1 (Test 2)	867,027	39,946	906,973	Apr. 3, 2018 - May 22, 2018
Data set 2 (Test 1)	547,165	27,983	575,148	Feb. 1, 2017 - Nov. 15, 2018

5 Experiments, Results and Observations

Finding a large, ground truth dataset of domain names for DGA analysis is often difficult. Although many published machine learning based DGA detection solutions exist, there are still many ambiguities in regards to the selection of appropriate machine learning algorithms. This is mainly because most benchmark results are from various data sets, which have differing data distributions and are collected from different resources, such as Alexa [38], OpenDNS [39], 360-DGA, OSINT real time DGA feeds [40], DGArchive [41] and publically available DGA algorithms [42]. In addition, some of the data sets and their associated systems are not publically available for research, due to privacy and security concerns. More importantly, the past DGA analytics have not considered time domain information, which is an important factor to identify zero day vulnerabilities. The majority of security companies make their data sets private, since they do not want to disclose their security flaws.

We evaluated the performance of the proposed method using two corpuses: publically available and private corpus. The publically available corpus (Data set 1) is collected from the publically available DGA algorithms including [42], DGArchive [41], and OSINT real time DGA feeds [40]. The main difference between the two test data sets of Data set 1 is that the Test 1 data set is collected from both the publically available DGA algorithms and real-time OSINT DGA feeds and the Test 2 data set is collected from only real-time OSINT DGA feeds. The private corpus (Data set 2) is collected from the internal network [23]. This data set is entirely collected in a different environment in comparison to the Data set 1. The private data set contains less number of DGA domain names, and it is highly imbalanced. These different test data sets show how well the trained model on the previous day data is able to detect DGA domain names, which occur in the future and how well the train model is able to detect DGA domain names which occurs from different sources. The detailed statistics of data set is reported in Table 5. Legitimate domain names are collected from Alexa [38] and OpenDNS [39]. Each of the test data sets are disjoint. The results of the Data set 1 can be reproduced because the data set is publically available and moreover the nature of real time domain names in Data set 2 makes the models to stays safe from an adversary. To defeat the classifier trained on Data set 2, an adversary has to spend more time in reverse engineering in comparison to the Data set 1.

To avoid possible biases, the real time DNS traffic was captured from an internal network. Passive sensors were deployed in an internal network. The detailed experimental setup and the collection of DNS traffic data set from the internal network is reported in [23]. The DNS query response messages from the DNS servers are collected by passive sensors. Thus, in this work, the DNS traffic was collected from different DNS servers. The sensors capture the network traffic from DNS servers, extract DNS packets and converts into human readable format. This data will be passed to the DNS log collector. The passive sensors could be installed inside the DNS server itself or mirror only the DNS traffic to a server, which is dedicated for DNS passive sensors. The sensors get the data by port mirroring the traffic from the DNS servers present in the deployed network. The logs received from the internal network are passed to the distributed log parser.

All deep learning architectures are implemented using TensorFlow [14] with Keras [43]. Initially, all the models are trained with Data set 1 (Train). During training, the train data set of Data set 1 is randomly divided into 80% training and 20% validation. The validation data helps to monitor the train accuracy across different epochs. The experiments with all deep learning architectures are run till 100 epochs with learning rate 0.001, *adam* optimizer, batch size of 128. To evaluate the performance, the trained model on Data set 1 is tested with the different test data sets, such as Data set 1 (Test 1), Data set 1 (Test 2), and Data set 2 (Test 1). These data sets are disjointed and they are collected from different sources. Most of the models performed better on Data set 1 (Test 1) compared to the test data taken from Data set 1 (Test 2) and Data set 2 (Test 1). An evaluation is accompanied with the ground-truth test data, which has a set of domain names labeled as DGA generated D or legitimate L. The data in Data set 1 (Test 1), Data set 1 (Test 2), and Data set 2 (Test 1) are collected from various sources and times to examine the detection capability of the trained models.

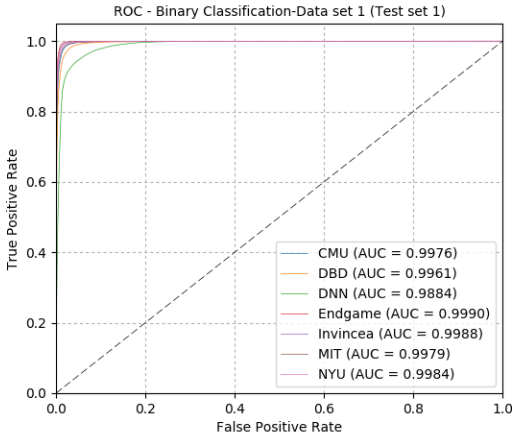
We used $Accuracy \in [0,1]$, $Precision \in [0,1]$, $Recall \in [0,1]$, and $F1-score \in [0,1]$ standard metrics, which are based on True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Detailed results are reported in Table 6. TP represents the number of DGA generated domain name samples correctly identified as DGA, TN represents the number of legitimate domain name samples correctly identified as legitimate, FP represents the number of legitimate domain name samples misclassified as DGA, FN denotes the number of DGA generated domain name samples misclassified as legitimate. The metrics *Accuracy*, *Precision*, *Recall* and $F1-score$ are defined as follows.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{5}$$

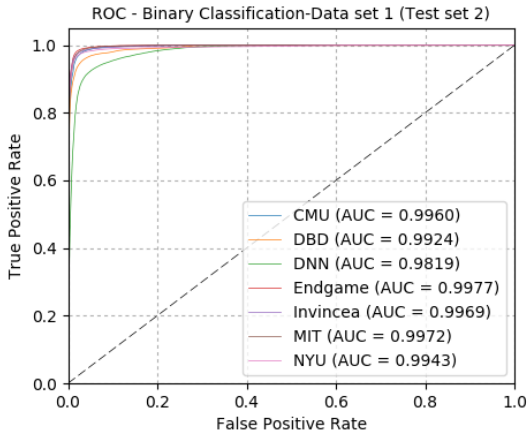
$$Precision = \frac{TP}{TP+FP}, \tag{6}$$

$$Recall = \frac{TP}{TP+FN}, \tag{7}$$

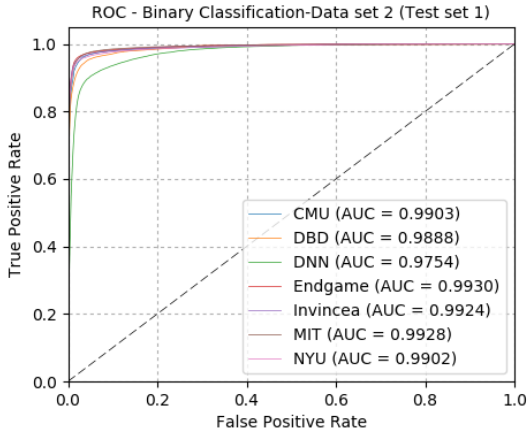
and



(a)

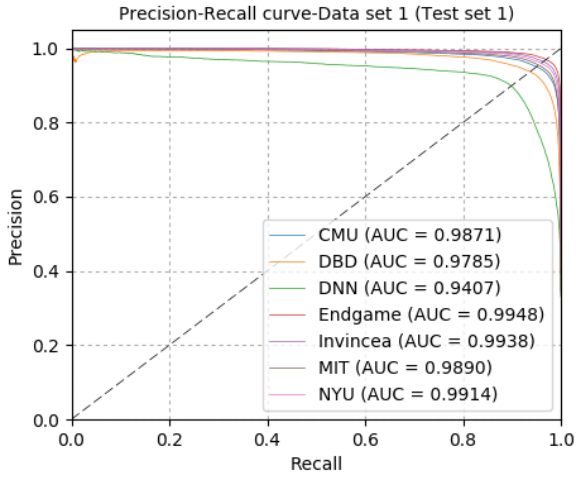


(b)

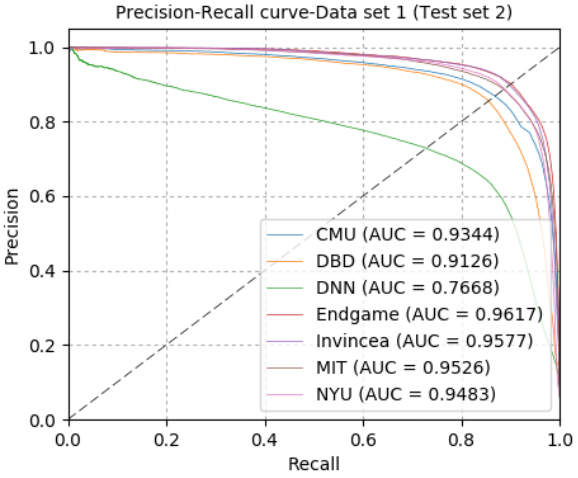


(c)

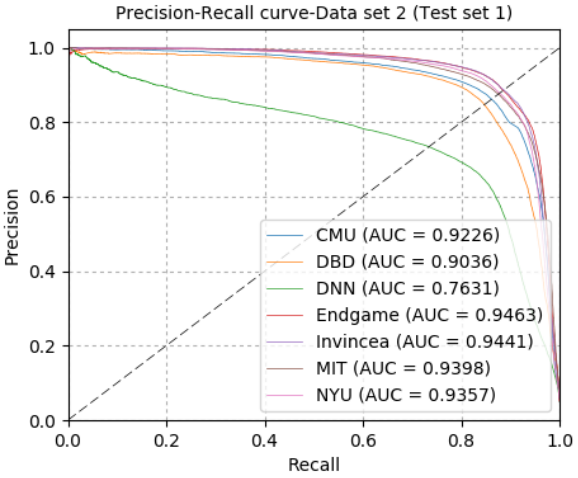
Fig. 5: ROC curve for (a) Data set 1 (Test 1), (b) Data set 1 (Test 2), and (c) Data set 2 (Test 1)



(a)



(b)



(c)

Fig. 6: Precision-Recall curve for (a) Data set 1 (Test 1), (b) Data set 1 (Test 2), and (c) Data set 2 (Test 1)

Table 6: Detailed test results

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Data set 1 (Test 1)				
Endgame [18]	99.0	96.3	97.9	97.1
NYU [36]	98.7	95.3	96.9	96.1
MIT [35]	98.5	94.8	96.4	95.6
CMU [33]	98.2	92.5	97.6	94.9
Invincea [32]	98.8	96.4	96.5	96.5
3-gram with 5-layer DNN	96.5	89.9	89.9	89.9
DBD	97.8	91.5	95.9	93.7
Data set 1 (Test 2)				
Endgame [18]	98.8	79.8	96.1	87.2
NYU [36]	98.7	79.2	94.6	86.2
MIT [35]	98.6	77.2	95.5	85.4
CMU [33]	97.8	67.5	96.7	79.5
Invincea [32]	98.9	82.9	94.2	88.2
3-gram with 5-layer DNN	96.6	57.0	90.7	70.0
DBD	97.6	65.9	93.9	77.5
Data set 2 (Test 1)				
Endgame [18]	98.6	81.1	93.3	86.8
NYU [36]	98.5	80.5	92.3	86.0
MIT [35]	98.4	78.3	93.1	85.1
CMU [33]	97.7	69.2	94.3	79.8
Invincea [32]	98.8	84.3	91.8	87.9
3-gram with 5-layer DNN	96.4	58.8	88.6	70.7
DBD	97.5	67.8	92.1	78.1

$$F1-score = 2 \times \left(\frac{\mathrm{Pr}ecision \times \mathrm{Re}call}{\mathrm{Pr}ecision + \mathrm{Re}call} \right). \tag{8}$$

Two most commonly used diagnostic tools to identify the interpretation of the binary classifiers are Receiver Operating Characteristic (ROC) and the precision-recall curves. Generally, the ROC curves are used when the samples of each class are balanced; otherwise, precision-recall curves are used.

To generate precision-recall curves, we estimated the trade-off between the $Precision \in [0,1]$ and $Recall \in [0,1]$ across varying threshold in the range of $[0,1]$. Most commonly, the area under the curve (AUC) is used to compare the ROC and precision-recall curves. AUC, as the name indicates, just the area under the ROC

Table 7: Results on test data sets. TPR and FPR are w.r.t. a threshold 0.5

Model	TPR (%)	FPR	AUC (%)
Data set 1 (Test 1)			
Endgame [18]	72.046	0.246	99.76
NYU [36]	81.353	0.348	99.61
MIT [35]	83.404	0.339	98.84
CMU [33]	70	0.078	99.9
Invincea [32]	75.82	0.276	99.88
3-gram with 5-layer DNN	84.849	0.209	99.79
DBD	86.871	0.383	99.84
Data set 1 (Test 2)			
Endgame [18]	96.079	0.542	99.6
NYU [36]	96.842	0.568	99.24
MIT [35]	96.962	0.574	98.19
CMU [33]	93.597	0.346	99.77
Invincea [32]	96.374	0.543	99.69
3-gram with 5-layer DNN	93.418	0.487	99.72
DBD	96.755	0.573	99.43
Data set 2 (Test 1)			
Endgame [18]	93.947	0.511	99.03
NYU [36]	95.910	0.555	98.88
MIT [35]	96.137	0.557	97.54
CMU [33]	90.732	0.311	99.30
Invincea [32]	94.723	0.519	99.24
3-gram with 5-layer DNN	90.758	0.445	99.28
DBD	95.908	0.565	99.02

curve. To generate ROC, we estimated the trade-off between the true positive rate $TPR \in [0,1]$ (Equation 7) on the vertical axis to false positive rate ($FPR \in [0,1]$) on the horizontal axis across varying threshold in the range of $[0,1]$, where

$$TPR = \frac{TP}{TP+FN}, \tag{9}$$

and

$$FPR = \frac{FP}{FP+TN}. \tag{10}$$

We set the threshold for both ROC and precision-recall curves to 0.5 without fine-tuning for low FPR. Low FPR is an important measure in DGA detection, particularly in aplications where there is a live stream of queries in a DNS server. If

the FPR is high, the system may block legitimate traffic in real-time systems. To highlight the differences in methods, FPR is adopted on a log scale.

The ROC and the precision-recall curves of various models are depicted in Figs. 5 and 6. These curves are obtained using using different test data. A high score of precision and recall indicates a more accurate detection. The results of various tests are averaged. Among all 3 data sets, the CMU model shows the lowest FPR, as reported in Table 7. There exists a clear variation in the TPR that the models achieve against that low FPR. All the models are able to catch DGA domain names with a TPR of 70-80 range on Data set 1 (Test 1), 90-98 range on Data set 1 (Test 2) and Data set 2 (Test 1). However, our proposed method, even though the threshold parameter has not been fine-tuned, shows a better performance in regards to AUC (more than 0.97) in both ROC and precision-recall curves. Similarly, the precision-recall curves shown in Fig 6 contain larger AUCs.

Endgame [18] model has performed well in comparison to all other models in all 3 different types of testing data sets. The performance obtained by all models on Data set 1 (Test 1) is higher compared to other data sets. Moreover, the performances of all models on Data set 1 (Test 2) is higher compared to Data set 2 (Test 1). This is mainly due to the differing statistics of datasets. Both Data set 1 (Test 1) and Data set 1 (Test 2) are collected from publically available sources and Data set 2 (Test 1) is collected from a private environment. Thus, the patterns in Data set 1 (Test 1) are entirely different from those of other data sets.

The performances obtained by all models have marginal differences in terms of accuracy, the area under the receiver operating characteristic (ROC) curve and precision-recall (AUC). The methods under study used Keras embedding as the text representation method, and they performed better compared to the 3-gram text representation method with DNN. This is mainly because the Keras embedding has the capability to learn the sequential representation of characters in domain names.

6 Conclusion

Perpetrators use various families of domain generating algorithms to hide the location of their C&C servers and maintain the efficiency of their botnets. To this end, we proposed a method to detect such algorithms, which disguise servers from being blacklisted or shut down. Our proposed method uses a deep learning architecture with Keras embedding, and it has the capability to detect the newly generated DGA based domain names or the variants of existing DGA generated domain names. The proposed method does not rely on the use of contextual information, such as NX-Domains, domain reputation, and feature engineering. The method analyses DNS data recursively and discovers the C&C server with the hosts it has infected. The proposed method provides DNS administrators a defensive capability to detach the rapidly growing botnets at a premature stage and lessens their adverse impacts. Our method is highly scalable and is capable of handling Ethernet LAN data with low false positive rates. We evaluated the performance of our method over various well-

known character-based, short-text classification models. We used 3-gram with DNN network for our evaluations. We found that most of the models under study have marginal differences in terms of accuracy. In terms of training speed and computational cost, the DBD method is more efficient compared to other models. Our deep learning based solution is very efficient as it captures the most significant features. Furthermore, we showed that it would be difficult for perpetrators to craft malwares, which can avoid our detection method.

Acknowledgment

This research was supported in part by Paramount Computer Systems and Lakhshya Cyber Security Labs. We are grateful to NVIDIA India, for the GPU hardware support to research grant. We are also grateful to Computational Engineering and Networking (CEN) department for encouraging the research.

References

1. Broadhurst, R., Grabosky, P., Alazab, M., Bouhours, B., and Chon, S. (2014). An Analysis of the Nature of Groups Engaged in Cyber Crime (2014). An Analysis of the Nature of Groups engaged in Cyber Crime, International Journal of Cyber Criminology, vol. 8, no. 1, pp. 1-20. Available at SSRN: https://ssrn.com/abstract=2461983
2. Alazab M., Venkatraman S., Watters P., Alazab M., Alazab A. (2012) Cybercrime: The Case of Obfuscated Malware. In: Georgiadis C.K., Jahankhani H., Pimenidis E., Bashroush R., Al-Nemrat A. (eds) Global Security, Safety and Sustainability and e-Democracy. e-Democracy 2011, ICGS3 2011. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, Berlin, Heidelberg, vol, 99.
3. Alazab, M., Broadhurst, R. (2016). Spam and Criminal Activity (2015). Trends and Issues in Crime and Criminal Justice, No.526, Australian Institute of Criminology. https://aic.gov.au/publications/tandi/tandi526
4. Ahmad U., Song H., Bilal A., Alazab M., and Jolfaei A. (2018). Secure Passive Keyless Entry and Start System Using Machine Learning. In: Wang G., Chen J., Yang L. (eds) Security, Privacy, and Anonymity in Computation, Communication, and Storage. SpaCCS 2018. Lecture Notes in Computer Science, vol 11342.
5. R, V., Alazab, M., Jolfaei, A., KP, S., and Poornachandran, P. (2018). Ransomware Triage Using Deep Learning: Twitter as a Case Study, In International Conference on Cyber Security and Communication Systems, Springer, Melbourne, Australia.
6. Alazab, M. (2015) "Profiling and Classifying the Behaviour of Malicious Codes", Journal of Systems and Software, Elsevier, Volume 100, February 2015, pages 91102.
7. Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., and Feamster, N. (2010). Building a Dynamic Reputation System for DNS. In USENIX security symposium, pp. 273-290.
8. Ollmann, G. (2009). Botnet communication topologies. Retrieved September, 30, 2009. Available at http://www.technicalinfo.net/papers/PDF/WP_Botnet_Communications_Primer_(2009-06-04).pdf
9. Wang, T.S., Lin, H.T., Cheng, W.T., and Chen, C.Y. (2017). DBod: Clustering and detecting DGA-based botnets using DNS traffic analysis. Computers and Security, vol. 64, pp. 1-15.

10. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., and Dagon, D. (2012). From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In USENIX security symposium, vol. 12.
11. Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys, vol. 51, no. 4, 81.
12. Joyce, J. M. (2011). Kullback-leibler divergence. In International encyclopedia of statistical science (pp. 720-722). Springer, Berlin, Heidelberg.
13. Tang, M., Alazab, M., Luo, Y. (2017) Big data mining in cybersecurity', IEEE Transactions on Big Data. https://ieeexplore.ieee.org/document/7968482
14. Abadi, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In OSDI, vol. 16, pp. 265-283.
15. Silva, S. S., Silva, R. M., Pinto, R. C., and Salles, R. M. (2013). Botnets: A survey. Computer Networks, vol. 57, no. 2, pp. 378-403.
16. Jolfaei, A., Vizandan, A. and Mirghadri, A. (2012). Image encryption using HC-128 and HC-256 stream ciphers. International Journal of Electronic Security and Digital Forensics, vol. 4, no. 1, pp. 19-42.
17. Ahluwalia, A., Traore, I., Ganame, K., and Agarwal, N. (2017). Detecting Broad Length Algorithmically Generated Domains. In International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments, Springer, pp. 19-34.
18. Woodbridge, J., Anderson, H. S., Ahuja, A., and Grant, D. (2016). Predicting domain generation algorithms with long short-term memory networks. arXiv preprint arXiv:1611.00791.
19. Yu, B., Gray, D. L., Pan, J., De Cock, M., and Nascimento, A. C. (2017). Inline DGA detection with deep networks. In IEEE International Conference on Data Mining Workshops, pp. 683-692.
20. Lison, P. and Mavroeidis, V. (2017). Automatic Detection of Malware-Generated Domains with Recurrent Neural Models. arXiv preprint arXiv:1709.07102.
21. Vinayakumar, R., Soman, K. P., Poornachandran, P., and Sachin Kumar, S. (2018). Evaluating deep learning approaches to characterize and classify the DGAs at scale. Journal of Intelligent and Fuzzy Systems, vol. 34, no. 3, pp. 1265-1276.
22. Vinayakumar, R., Soman, K. P., and Poornachandran, P. (2018). Detecting malicious domain names using deep learning approaches at scale. Journal of Intelligent and Fuzzy Systems, vol. 34, no. 3, 1355-1367.
23. Vinayakumar, R., Poornachandran, P., and Soman, K. P. (2018). Scalable Framework for Cyber Threat Situational Awareness Based on Domain Name Systems Data Analysis. In Big Data in Engineering Applications, pp. 113-142.
24. Mohan, V. S., Vinayakumar, R., Soman, K. P., and Poornachandran, P. (2018). Spoof net: Syntactic patterns for identification of ominous online factors. In IEEE Security and Privacy Workshops, pp. 258-263.
25. Vinayakumar, R., Soman, K. P., Poornachandran, P., Mohan, V. S., and Kumar, A. D. (2019). ScaleNet: Scalable and Hybrid Framework for Cyber Threat Situational Awareness Based on DNS, URL, and Email Data Analysis. Journal of Cyber Security and Mobility, vol. 8, no. 2, pp. 189-240.
26. Maaten, L. V. D., and Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, vol. 9, pp. 2579-2605.
27. Curtin, R. R., Gardner, A. B., Grzonkowski, S., Kleymenov, A., and Mosquera, A. (2018). Detecting DGA domains with recurrent neural networks and side information. arXiv preprint arXiv:1810.02023.
28. Tran, D., Mac, H., Tong, V., Tran, H. A., and Nguyen, L. G. (2018). A LSTM based framework for handling multiclass imbalance in DGA botnet detection. Neurocomputing, vol. 275, pp. 2401-2413.
29. Feng, Z., Shuo, C., and Xiaochuan, W. (2017). Classification for DGA-Based Malicious Domain Names with Deep Learning Architectures. In 2017 Second International Conference on Applied Mathematics and information technology, p. 5.

30. Mac, H., Tran, D., Tong, V., Nguyen, L. G., and Tran, H. A. (2017). DGA Botnet Detection Using Supervised Learning Methods. In Proceedings of the Eighth ACM International Symposium on Information and Communication Technology, pp. 211-218.
31. Yu, B., Pan, J., Hu, J., Nascimento, A., and De Cock, M. (2018). Character level based detection of DGA domain names. In IEEE World Congress on Computational Intelligence, pp. 41684175.
32. Saxe, J., and Berlin, K. (2017). eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. arXiv preprint arXiv:1702.08568.
33. Dhingra, B., Zhou, Z., Fitzpatrick, D., Muehl, M., and Cohen, W. W. (2016). Tweet2vec: Character-based distributed representations for social media. arXiv preprint arXiv:1605.03481.
34. Jolfaei, A. and Mirghadri, A. (2011) Substitution-permutation based image cipher using chaotic henon and baker's maps. International Review on Computers and Software, vol. 6, no. 1, pp. 40-54.
35. Vosoughi, S., Vijayaraghavan, P., and Roy, D. (2016). Tweet2vec: Learning tweet embeddings using character-level cnn-lstm encoder-decoder. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, pp. 1041-1044.
36. Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In Advances in neural information processing systems, pp. 649-657.
37. Manos Antonakakis , Roberto Perdisci , Yacin Nadji , Nikolaos Vasiloglou , Saeed Abunimeh , Wenke Lee , David Dagon (2012). From throw-away traffic to bots: detecting the rise of dga-based malware (2012). In Proceedings of the 21st USENIX conference on Security symposium.
38. Does Alexa have a list of its top-ranked websites?, Available at Alexa: https://support.alexa.com, Accessed May 09 2018.
39. OpenDNS domain list, Available at OpenDNS: https://umbrella.cisco.com/blog, Accessed May 09 2018.
40. OSINT DGA feeds, Available at OSINT: https://osint.bambenekconsulting.com/, Accessed 09/05/2018.
41. DGArchive, Available at DGArchive: https://dgarchive.caad.fkie.fraunhofer.de/, Accessed 15/06/2018.
42. DGA Algorithms, Available at Github: https://github.com/baderj/domain generation algorithms, Accessed 01/05/2018
43. Gulli, A. and Pal, S. (2017). Deep Learning with Keras. Packt Publishing Ltd.