Inline DGA Detection with Deep Networks

Bin Yu†, Daniel L. Gray*, Jie Pan*, Martine De Cock^{*,1}, and Anderson C. A. Nascimento*
**Institute of Technology, University of Washington Tacoma*
Tacoma, Washington
Email: {dangray, jiep, mdecock, andclay}@uw.edu
†*CTO Office, Infoblox*
Santa Clara, California
Email: biny@infoblox.com

***Abstract*—Domain generation algorithms (DGAs) automatically generate large numbers of domain names in DNS domain fluxing for the purpose of command-and-control (C&C) communication. DGAs are immune to static prevention methods like blacklisting and sinkholing. Detection of DGAs in a live stream of queries in a DNS server is referred to as inline detection. Most of the previous approaches in the literature on DGA detection either: (i) are based on small synthetic data sets for training, rather than data collected from real traffic or (ii) require contextual information and therefore cannot be used for inline detection. In this work, we overcome these limitations by proposing a novel way to label a large volume of data collected from real traffic as DGA/non-DGA and by using deep learning techniques. Our classifiers can be trained with large amounts of real traffic, rather than small synthetic data sets, and therefore have better performance.**

1. Introduction

Domain generation algorithms (DGAs) generate large numbers of domain names of which few get resolved for command-and-control (C&C) communication. This method of communication between bot and master evades standard means for impeding botnet activity, which are largely based on static lists of domain names. The challenge to catch malicious domain names that are generated dynamically has led to recent interest in detecting DGA domains using machine learning algorithms. Models which predict based solely on the domain name string are of particular interest for their generality: context information beyond the domain name string is generally unavailable or expensive to acquire.

Detecting DGAs retrospectively through means such as clustering has been well explored, but has the disadvantage of being reactionary. These approaches are based on analysis of batches of traffic, which means DGA domains are likely to establish communication before being detected. Ideally we should detect DGA domains in real-time, predicting on a per domain basis, to prevent any C&C communication. Relatively little progress has been made on this challenge so far. Predicting on a per example basis is a supervised learning problem, and thus requires labeled data. Labeled DGA domains can be obtained through reverse engineering malware. Domains coming from the Alexa list [1] are usually used as examples of non-malicious domains. There are a couple of problems with this approach: (i) reverse engineering DGA malware is a tedious task, and models trained in this manner will become outdated as new DGA malware families emerge in real traffic quickly; (ii) there is no guarantee that domains from Alexa form a good representative set of non-malicious domains within a specific network.

To overcome these problems, we propose a novel technique for obtaining labeled data towards training an inline DGA detector. Our technique is based on real traffic and does not require the reverse engineering of DGA malware to obtain malicious domains. When a domain name is queried for which no IP address exists, the client is returned a Non-Existent Domain (NXDomain) response. Based on our observations of DGA behavior, we perform heuristic filtering on billions of NXDomains with the intent of creating a noise-not-free but practical data set of live DGA domains collected from real traffic. Specifically, we collect negatives (legitimate domains) from resolving traffic, and positives (potential DGA domains) from non-resolving traffic. The reasoning for using NXDomain traffic is as follows: (i) Only a small subset of DGA domains are actually intended for C&C communication and resolved. The vast majority of DGA domains do not resolve. This behavior means that DGA domains are largely isolated to NXDomain traffic. By the same reasoning, resolving traffic is generally non-DGA, and thus benign. (ii) Furthermore, looking solely at the domain name strings, DGA domains that resolve should be indistinguishable from those that do not (we expect that DGA domains used for C&C communication are generated in the same manner as non-resolving DGA domains).

To boost the signal of DGA domains, we filter based on how we expect to observe DGAs in real traffic (for instance, a DGA domain is disposable and will not be reproduced outside a very short span of time, e.g. 7 days, neither from the same host nor a different host). The binary classification models that we train over this filtered data set can be deployed at a DNS server to impede botnets by

^{1}*Guest Professor at Ghent University* preventing malicious responses to infected clients (see Fig. 1). Furthermore, domains which are predicted malicious, yet do resolve, are likely resolving to a C&C IP address. Thus our model is not only useful for passively blocking C&C communication, but also for detecting C&C servers.

Our novel strategy allows us to obtain an enormous amount of labeled data. This opens the door to use deep neural networks for this classification task, which boast many advantages over traditional methods. Deep learning has achieved state-of-the-art results in terms of predictive accuracy on a variety of complex tasks in recent years, while also performing efficient classification, essential for an inline detection scheme. In contrast to more traditional approaches, deep neural nets can learn features automatically, bypassing the human effort of feature engineering. This is useful for DGA detection, as malware authors could use knowledge of human engineered features to better blend with legitimate traffic. Our self-labeling technique makes online learning possible. Many traditional models must be fit from scratch in order to update them with new data. The weights of neural networks on the other hand can be tuned continuously. Thus a neural net can be deployed and used for DGA detection while simultaneously training on live traffic. This property should allow our model to keep up with DGAs as they change in the wild.

We train and evaluate two types of deep neural networks: convolutional and recurrent neural networks (CNN and LSTM respectively). Our networks operate at the character-level, and are given only the domain name string without other context. To create a baseline for our models, we also evaluate traditional machine learning models using human engineered features.

Low false positive rate (FPR) is a clear priority for an inline detection scheme. We set the threshold for the models at a FPR of 1 in 10,000 against our data, and report results in terms of accuracy, AUC, and true positive rate (TPR). At this restrictive threshold, our best deep learning model still achieves 40.31% TPR against our validation set.

In order to establish ground truth and evaluate the performance of our classifier, we turn to a repository of known DGA domains, DGArchive [2], for concrete validation. Against this sample of DGA/non-DGA truth marked data, our best deep learning model achieves 72.89% TPR and 0.31% FPR.

We additionally deploy a CNN on a live stream of traffic to observe what is flagged malicious. We analyze IP addresses resolved by flagged domains in this stream, namely by inspecting the list of domains which resolve to them. In this way, we find many IPs that are, beyond reasonable doubt, malicious.

The contributions of this work are as follows:

- We propose a novel criteria for creating a noise-not-free DGA/non-DGA data set from real traffic that bypasses the difficulty of explicit labeling. To the best of our knowledge, all the previous approaches for creating labeled data sets of DGAs were based on synthetic data rather than real traffic. Thus, the performance metrics (accuracy, false-positive, false-negative rates) computed for synthetic data based systems did not provide insight on their real world performance.
- We train and evaluate character-level deep learning techniques for DGA detection using this filtered data. Our results are practical and can be used for blocking/filtering the communication between bots and the command-and-control center in real time.

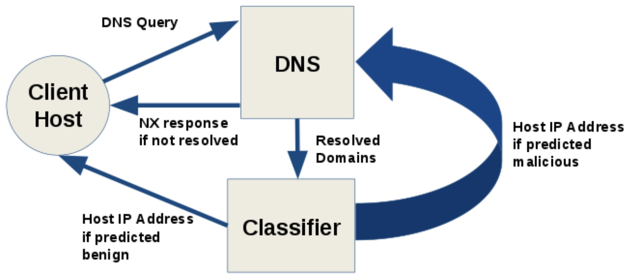
DNS Query
DNS
Client Host
NX response if not resolved
Resolved Domains
Host IP Address if predicted malicious
Host IP Address if predicted benign
Classifier

Figure 1. Diagram depicting the deployment of our classifier for inline detection, showing a transaction with an infected client, or client host. Domains which resolve at the DNS server are checked against the classifier to predict if they are malicious. Benign and NXDomain responses can be returned to the client as usual while potentially malicious responses are blocked to impede communication with a C&C server. IP addresses associated with potentially malicious domains may also be further investigated.

2. Related work

DNS data is useful for detecting malicious activities in a network, including fast-fluxing, tunneling and DGAs [3], [4], [5], [6]. In the case of DGA detection, a large number of previous works are based on a retrospective approach: that is, data is collected over a time period and then a classification algorithm is performed on the collected data [5], [6].

In [7], clustering is used during an off-line phase where domains are grouped based on domain-related features. From each cluster of domains, linguistic features are extracted and used to build signatures for catching DGA domains during an on-line phase.

A hybrid (off-line, on-line) approach is used by [8], where potentially malicious domains are first clustered, and then properties of these clusters are extracted and subsequently used for catching the command-and-control center during an on-line phase.

All of these works exhibit the following characteristics:

- They are based on human engineered features.
- They are trained and evaluated on synthetic data sets (e.g. Alexa and a list of domains obtained by reverse-engineered malware).

Whenever human engineered features are used, it is obvious that it opens the door for an adversary to carefully craft its DGA to avoid detection by using the aforementioned features. Moreover, synthetic data sets might be substantially different from what is present in real traffic. Thus, validation

TABLE 1. OVERVIEW OF DATA SETS. THE POSITIVE EXAMPLES IN THE GOLD DATA SET ARE REVERSE ENGINEERED DGA DOMAIN NAMES, WHILE THE NEGATIVE EXAMPLES ARE FROM THE ALEXA TOP 1M LIST. IN ALL OTHER DATA SETS, EXAMPLES ARE TAKEN FROM REAL TRAFFIC. IN PARTICULAR, POSITIVE EXAMPLES ARE NXDOMAINS, AND NEGATIVE EXAMPLES ARE DOMAINS THAT RESOLVED. THE TRAIN, VALIDATION AND PROSPECTIVE DATA SETS ARE ALL DISTINCT: EACH DOMAIN NAME OCCURS IN AT MOST ONE DATA SET.

Set	Positives	Negatives	Observation Dates
Train	9,121,642 (9.1M)	8,767,638 (8.8M)	Sept. 2015 - Jan. 2017
Val	2,280,573 (2.3M)	2,192,259 (2.2M)	Sept. 2015 - Jan. 2017
Pros1	1,982,843 (2M)	2,993,048 (3M)	Feb. 2017
Pros2	2,621,980 (2.6M)	1,466,152 (1.5M)	Mar. 2017
Pros3	4,685,382 (4.7M)	23,633 (24K)	Apr. 2017
Gold	4,739,563 (4.7M)	489,030 (0.5M)	N/A

based solely on such data sets might not reflect how well these classifiers would perform in a real world situation.

In [9], DGAs are detected by observing the number of DNS requests by a client that obtain an NX answer and applying hypothesis testing. Despite avoiding the use of human engineered features, the approach proposed in [9] needs to segregate DNS requests by IP addresses of the clients. IP addresses can be seen as private information and it is desirable to have methods that do not need such information.

The most similar work in the literature to ours is the exploration of character-level LSTM for DGA detection by Woodbridge et al. [10]. The primary difference between this work and ours is that they train with synthetically generated DGA/non-DGA labeled data. For non-DGA examples, they use Alexa Top 1M domains. For DGA domains, they use the OSINT DGA feed from Bambenek Consulting [11]. The sample used contains varying numbers of examples from thirty DGA families with a total of approximately 750,000 examples [10]. Our work is based on real traffic data.

To the best of our knowledge, our work is the first to apply deep learning for real-time DGA detection (i) without the use of human engineered features, and (ii) with the use of real traffic, rather than synthetic data, for training.

3. Creation of the Data Set

Unlike most of the traditional approaches that use reverse engineered DGA algorithms to generate positive samples, we use real and live traffic data as samples to train a classifier that can target active DGAs. The raw data used in this work is a real time stream obtained from Farsight Security [12]. It consists of roughly 10 billion DNS queries per day collected from subscribers including ISPs, schools, and businesses, from September 2015 through April 2017. This data is private, and no sample is publicly provided. Since it is impossible to have human annotators truth mark this large amount of data, we propose a novel approach based on the assumption that a DGA generated domain will not be regenerated over a long period of time, to A and AAAA type queries (these queries account for 78% of the overall volume). We take as negative examples (legitimate domains) those domain names which have been resolved at least once, never resulted in an NXDomain response, and span more than 30 days (we define span as the number of days between the first and last query for a given domain). We take as positive examples those which never resolved and consistently resulted in NXDomain at least 10 times, and have all occurrences within the span of 7 days with standard deviation 3 days or less. The filtering parameters are chosen based on statistical observation of DGA behavior in real traffic. Furthermore, we assume that a DGA domain is a primary domain that can be registered under a publicly available top level domain (TLD). We take the second level domain (SLD) as the domain name string, and exclude all examples where the domain extends past the SLD (i.e. has a third level domain). In order to further minimize noise, only domains which have at least 10 characters are selected. Filtering by length should also be applied when the model is deployed, whereas the statistics based filters become infeasible in a production scenario.

We further separate our data into retrospective and prospective sets. The retrospective data is a balanced set of 12M positive and 12M negative examples from September 2015 through January 2017. Randomly, 80% is used for training, and 20% is reserved for validation. The prospective sets are reserved to evaluate the performance of our classifier on future traffic, consisting of domains observed from a later time period and not present in the training set. Specifically, the prospective sets contain positive and negative examples from February, March, and April 2017.

It is also important to note that the number of negatives drops considerably across the prospective sets. This is due to the time at which the sets were generated. Our filter for span of days takes all observations into account across the entire span of data (September 2015 through April 2017), and thus unique domains observed in more recent months have less opportunity to span over 30 days. The retrospective and prospective sets corresponding to the results of this paper were generated in May 2017.

For ground truth validation, we leverage DGArchive [2] and Alexa Top 1M [1] to create a small truth marked data set (we refer to this as the Gold set; see Table 1). DGArchive is a small repository of known DGA domains. These are collected using reverse engineered malware, forecasted seeds, and active dictionaries to match actual DGA domains in real traffic. The Alexa Top 1M web service provides access to lists of web sites ordered by Alexa Traffic Rank, which can be assumed legitimate. We filter this data, limiting to domains which are not observed in the training set, and have at least 10 characters in the SLD label. Before any filtering, we have 6,598,492 DGArchive domains and 1,000,000 Alexa domains. First we remove domains which overlap with the training set, leaving 5,893,625 positives and 986,886 negatives. Second, we enforce the filter by SLD length to match the form of domains in the training data, leaving 4,739,563 positives and 489,030 negatives to make predictions on. When using this set for evaluation, we count all domains removed in the second filtering step as negative predictions.

4. Feature Based Approach

In addition to our deep learning approach, we investigate how traditional, feature based machine learning algorithms perform on the classification task at hand. These algorithms do not present the advantages we are looking for in the deep learning approach, such as online learning and automatic feature learning, but can be used to provide a baseline against which the deep learning approach can be compared. We explored many approaches, including K-Nearest Neighbor, SVM, and AdaBoost. We also experimented using various training sizes in the feature based approach, checking performance when using 10K, 50K, 100K, 200K, 500K, 1M, and 5M random samples of the training set to fit the models. The most effective feature based method found is a random forest trained with 100K examples, and we present this as our baseline.

To fit a feature based model, we extract the following 11 features from each domain name string (see below for a more detailed description of each feature) [3], [4]: 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).

In information theory, entropy quantifies the information in message represented by a random variable [13]. Given a character set $\{x\}$ within a text string, we estimate the *normalized entropy of characters* in terms of its estimated distribution $D(x)$ and its length len:

$$\text{ent} = \frac{-\sum_{x} D(x) \log D(x)}{\log len}$$

The summation above is over all x in $\{x\}$.

The *Gini index of characters* is computed as:

$$\text{gni} = 1 - \sum_{x} D^2(x)$$

The *classification error of characters* is a measure which provides some indication of the diversity of characters in the string. It is defined as: $\text{cer} = 1 - \max\{D(x)\}$

The distribution of n-grams (sequences of n-characters) is not uniform in natural language, and thus n-gram frequency is a powerful feature for our classification task. From Google's Ngram Viewer [14], we created lookup tables for the frequencies of 2-grams and 3-grams. Given a string, we use these tables to look up the frequency of each 2-gram and 3-gram. We take nl2 as the *median of 2-gram* frequencies, and nl3 as the *median of 3-gram* frequencies.

In addition, we compute several *character ratio features* for a given text string as follows:

- naz (symbol character ratio): the number of characters that do not exist in the English alphabet divided by the string length.
- hex (hex character ratio): the number of hexadecimal characters (A-F) divided by the string length.
- vwl (vowel character ratio): the number of vowels divided by the string length.

The remaining features are defined as follows:

- tld (TLD hash): a hash value is computed for each potential TLD and normalized to the range 0 to 1.
- dgt (first character digit): a flag for whether the first character is or is not a numerical digit.
- len: the length of the string taken as the domain name.

Most of the features above are chosen based on their discrimination power for distinguishing natural language sequences, since many DGAs utilize sequences which are clearly distinct from natural language. Exceptions are the len, tld, and dgt features. Length can be used for grouping DGA families. TLD information can help indicate malicious domains, since DGAs tend to use certain TLDs to avoid conflict with legitimate traffic. Whether the first character is a number should be in and of itself a strong signal for malicious domains. Fig. 2 shows the distributions of these features in the retrospective data (observed Sept. 2015–Jan. 2017). The red curves represent the distributions on positive domains; the green curves represent the distributions on negative domains.

5. Deep Learning Approach

We train two types of neural networks: convolutional and recurrent neural networks. Both take raw data as input, bypassing the need for manual feature extraction.

5.1. Convolutional Neural Networks

CNNs are known for state-of-the-art advances in image processing, and apply to inputs of grid-like topology [15]. One-dimensional ("temporal") CNNs are a natural fit when the input is text, treated as a raw signal at the character level [16]. CNNs automatically learn filters to detect patterns that are important for prediction. The presence (or lack) of these patterns is then used by the quintessential neural network (multilayer perceptron, or MLP) to make predictions. These filters, (also called kernels) are learned during backpropagation. An intuitive example in image processing is a filter which detects vertical edges, while in text processing the filters detect sub-strings, or n-grams.

The underlying operation of CNNs is elementwise multiplication, performed between each filter and sub-sections of the input. The resulting values indicate the degree to which these sub-sections match the filters. In this manner, the filters are convolved over the input to form an activation map, which represents the locations of discovered features.

5.2. Long Short Term Memory Models

RNNs are designed to learn dependencies across an input sequence, featuring an information loop that takes as input

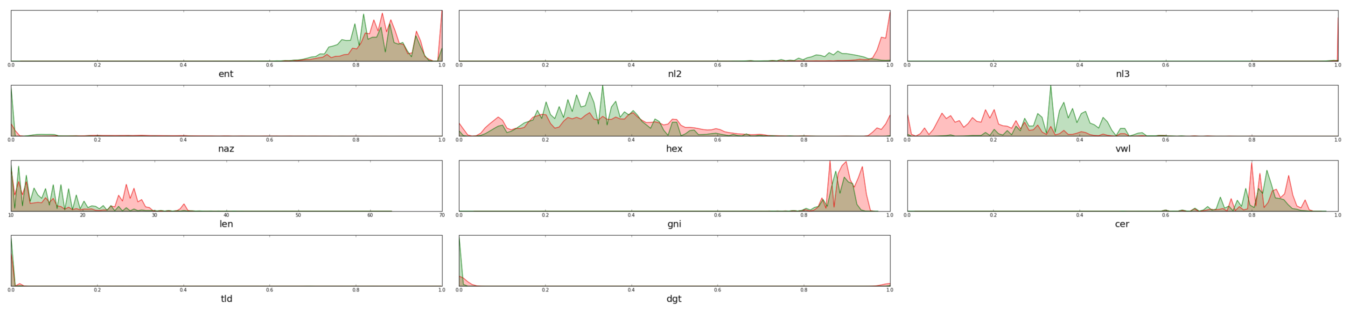

Figure 2. The charts above show the distributions of all 11 features in the retrospective data with filtering. The red curves correspond to positive examples (NXDomains). The green curves correspond to negative examples, which are likely legitimate.

both the current timestep of a sequence and the output from the previous timestep. LSTMs [17] are a special type of so-called gated RNNs. Instead of ordinary neurons, LSTMs are made up of LSTM cells that contain an internal recurrence (a self-loop) and a system of gating units that controls the flow of information. This makes LSTMs especially suitable to learn long-term dependencies, resulting in state-of-the-art advances in natural language processing (see e.g. [15] for an overview).

5.3. Implementation Details

We trained our neural nets using Python2 and Tensorflow [18]. Specifically, we use Keras [19], an API for rapid prototyping with Tensorflow and Theano. The platform used for training is an AWS virtual machine with access to multiple GPUs.

5.4. Representation

Both models are trained given only the domain name string as input. Each string is converted to lowercase, and then represented as a sequence of ASCII values corresponding to its characters. The maximum number of characters in a domain name is 63, so we pad zeros to the end such that all inputs have a fixed input width of 63. At this point, we then embed the input characters using Keras' built-in embedding layer. Embedding techniques such as Word2Vec [20] that map inputs to some vector space are now common practice in natural language processing. Generally, the goal is to map inputs which are similar for the given task closely together, resulting in improved performance. The embeddings are learned during training, separately for each model.

5.5. Deep Learning Models

Here we present the architecture of the best performing CNN and LSTM networks that we trained in the deep learning approach. Both operate at the character level and start with an embedding layer. Both models are trained using a batch size of 64. The LSTM is taken from the architecture proposed in [10], but with a few modifications.

As mentioned above, both CNN and LSTM networks include an embedding layer which learns to map 256 unique ASCII characters, each to its own 128 dimensional vector embedding. Note that there are only 37 allowed characters in domain names, and thus we could reduce from 256 to 38 unique inputs (one extra input for the padding value). The embedding layer expects inputs in the form of indexes (each index maps to a unique embedding). In our experiments, we reduced to 38 inputs and mapped the ASCII inputs to integers in the range 0-37 inclusive, but observed no difference in performance. Thus we left this parameter set to 256 for simplicity and potential for new unique inputs.

Following the embedding layer, the LSTM architecture is comprised of an LSTM layer (128 LSTM cells with default Tanh activation), a fully connected layer (100 nodes with default of linear activation), and finally a single node output layer with sigmoid activation. We additionally make use of the 'unroll' option, which accelerates training speed at the cost of memory use. Each layer defaults to 'glorot_uniform' weight initialization [21]. We obtained better loss convergence results using Adam [22] as the optimization algorithm, instead of RMSProp as proposed in [10]. Uniquely for the LSTM, we modified the Adam optimizer to have 0.001 decay, as opposed to the default of no decay (the rest of the parameters shown in the code below regarding Adam are defaults, and used in the CNN).

The CNN is structured similarly. It features a 1-Dimensional convolutional layer in place of the LSTM layer, uses 'glorot_normal' weight initialization [21], and uses the rectified linear unit (ReLU) as the activation function on all but the output layer. The CNN has 1000 kernels, each of size 2. We experimented with max-pooling, but did not see performance gains with this technique.

We also employ dropout for both networks. Dropout is a technique to improve model performance and overcome over-fitting by randomly excluding nodes during training, which serves to break up complex co-adaptations in the network [23]. This is confined to the training phase; all nodes are active during testing and deployment. We apply 50% dropout at the convolutional and LSTM layers.

Our models are defined in Keras as follows:

5.5.1. CNN.

```
model = Sequential(name='Seq')
model.add(Embedding(256, 128, input_length=63))
model.add(Conv1D(1000, 2, padding='same',
          kernel_initializer ='glorot_normal',
         activation ='relu'))
model.add(Dropout(0.5))
model.add( Flatten ())
model.add(Dense(100, activation ='relu',
          kernel_initializer ='glorot_normal'))
model.add(Dense(1, activation ='sigmoid',
          kernel_initializer ='glorot_normal'))
model.compile( loss=' binary_crossentropy ',
        optimizer='adam', metrics=['accuracy'])
```

5.5.2. LSTM.

```
model = Sequential(name='Seq')
model.add(Embedding(256, 128, input_length=63))
model.add(LSTM(units=128, unroll=True))
model.add(Dropout(0.5))
model.add(Dense(100))
model.add(Dense(1))
model.add( Activation ('sigmoid'))
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
        epsilon=1e-08, decay=0.001)
model.compile( loss=' binary_crossentropy ',
        optimizer=adam, metrics=['accuracy'])
```

6. Validation and Results

6.1. Validation

The metrics used to evaluate our models are true positive rate (TPR), false positive rate (FPR), and AUC (area under the ROC curve).

Low false positive rate is the most important quality for our models. If they are to be deployed, blocking legitimate traffic must be a rare occurrence. We choose to limit the false positive rate to 0.01% and tune the neural nets accordingly. This strict threshold means our models must be extremely confident before we flag a domain as malicious, and comes at the cost of severely reduced TPR. To do this, we simply choose a threshold probability which gives us 0.01% FPR on the validation set. We assume that negatives are largely consistent over time, and thus this FPR is roughly maintained on the prospective sets.

We obtain these metrics using the data sets detailed in Section 3. The size of the training data set and the variety of different test sets obviate the need for K-fold cross-validation. Results against the retrospective and prospective data sets (see Table 2) are taken at a high threshold for 0.01% FPR as described above. Results against truth marked data (see Tables 4 and 5) are taken at a threshold of 0.5, without tuning for low FPR.

For both deep learning models, we also present training convergence curves, which show how the accuracy improves after each iteration over the training data; see Fig. 4 and 5. Note that these accuracies are also computed at a prediction threshold of 0.5. It is important to point out that the training accuracy is lower than the validation accuracy in these curves. Accuracy against the training set is computed in an averaged way across batches while dropout is being applied, whereas performance on the validation set is determined at the end of each epoch with dropout disabled.

How the random forest compares to our neural nets is seen in the ROC plot (see Fig. 6), and we further highlight the performance of each model at 0.1% FPR in Table 3.

TABLE 2. NEURAL NET RESULTS AGAINST DATA FILTERED FROM REAL TRAFFIC (THRESHOLD SET FOR 0.01% FPR) (TPR_0: RETROSPECTIVE; TPR_{1-3}: PROSPECTIVE)

Model	TPR_0	TPR_1	TPR_2	TPR_3
CNN	40.31%	5.37%	10.14%	8.17%
LSTM	40.46%	2.43%	8.49%	6.64%

TABLE 3. BASELINE VS NEURAL NETS AGAINST DATA FILTERED FROM REAL TRAFFIC (THRESHOLD SET FOR 0.1% FPR) (TPR_0: RETROSPECTIVE; TPR_{1-3}: PROSPECTIVE)

Model	TPR_0	TPR_1	TPR_2	TPR_3
CNN	64.04%	35.65%	41.05%	37.64%
LSTM	57.71%	37.14%	42.88%	40.35%
RF	63.77%	62.06%	63.48%	20.79%

TABLE 4. RESULTS AGAINST TRUTH MARKED GOLD SET FROM ALEXA AND DGARCHIVE (THRESHOLD: 0.5)

Model	TPR	FPR
CNN	72.89%	0.31%
LSTM	74.05%	0.54%
Random Forest	71.28%	1.33%

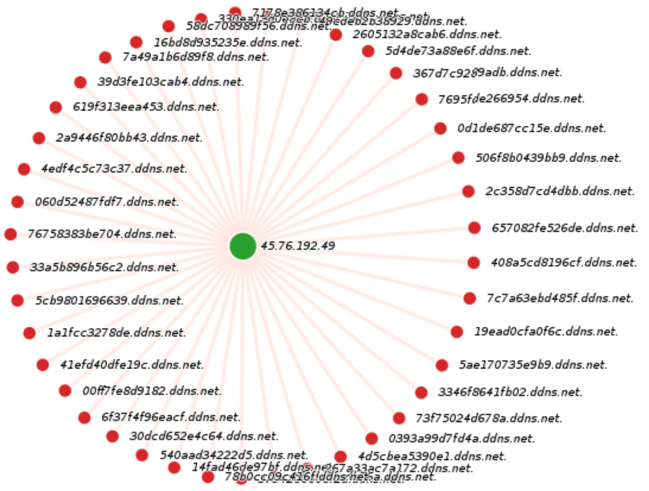

Figure 3. Above is a graph of domains which resolve to a potentially malicious host discovered through deployment of our CNN on a live stream of traffic.

TABLE 5. RESULTS AGAINST DGARCHIVE BREAKUP BY DGA FAMILY

DGA Family	Size Before Filter	Size After Filter	CNN	LSTM	Random Forest
bamital	225,911	225,911	100.00%	100.00%	99.96%
banjori	421,383	421,383	17.02%	30.54%	15.31%
bedep	14,473	14,473	97.19%	96.14%	93.87%
beebone	210	105	0.00%	0.00%	50.00%
blackhole	623	623	99.04%	99.20%	96.95%
bobax	300	128	42.00%	42.00%	42.67%
conficker	1,511,912	359,783	23.10%	23.10%	22.67%
corebot	147,030	147,030	91.84%	99.63%	95.46%
cryptolocker	1,183,256	1,183,256	98.17%	98.14%	96.41%
darkshell	49	0	0.00%	0.00%	0.00%
dircrypt	538	451	82.90%	82.71%	81.04%
dnsbenchmark	5	5	100.00%	100.00%	100.00%
dnschanger	1,499,578	1,499,578	96.79%	96.84%	92.70%
downloader	60	0	0.00%	0.00%	0.00%
dyre	746,962	746,962	100.00%	100.00%	100.00%
ekforward	1460	0	0.00%	0.00%	0.00%
emotet	137,687	137,687	99.01%	98.96%	96.37%
feodo	192	192	99.48%	100.00%	98.44%
fobber	1,996	1,996	98.25%	98.15%	95.34%

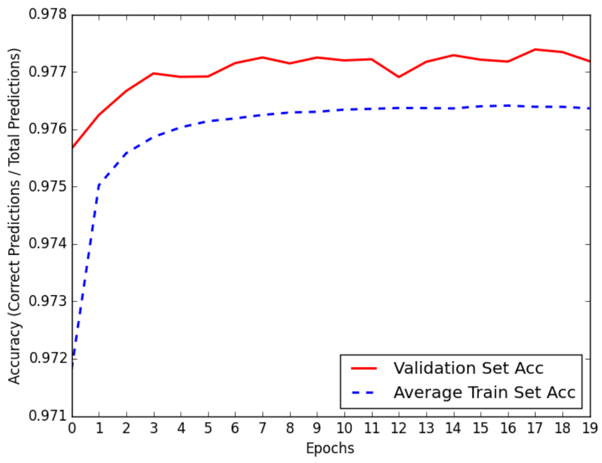

Figure 4. Accuracy over Epochs for CNN

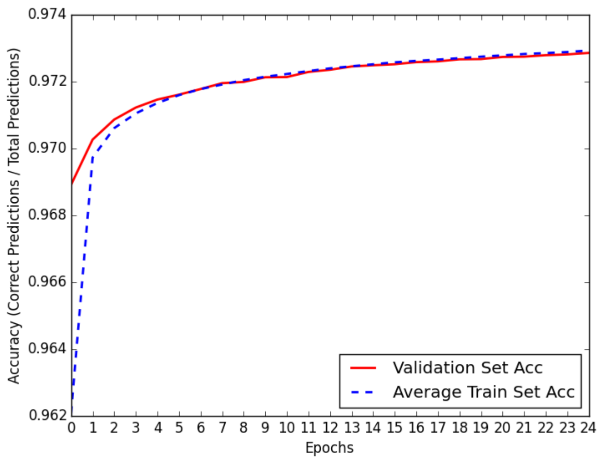

Figure 5. Accuracy over Epochs for LSTM

6.2. Evaluation by Metrics

Performance of the neural nets against our data from real traffic is shown in Table 2. The four columns give TPR against the validation set, Pros1, Pros2, and Pros3 from left to right (each TPR is given at 0.01% FPR). The CNN outperforms the LSTM model by each metric except for slightly worse TPR on the validation set.

Table 2 is structured in the same manner and compares the neural nets to the baseline random forest model. Here it is seen that at 0.1% FPR, the CNN is instead beaten by the LSTM on each metric but the validation TPR. Prospective results for the random forest would seem to indicate superior generalization, but the poor performance on Pros3 is peculiar. The neural nets achieve similar scores for each month of the prospective data, and consistently do best on Pros2, followed by Pros3 and Pros1.

6.3. Evaluation on Truth Marked Data

Due to the lack of ground truth in our filtered data, we also evaluate against a small truth marked set for concrete validation (see Gold set in Section 3). We evaluate each model against this set using a prediction threshold of 0.5

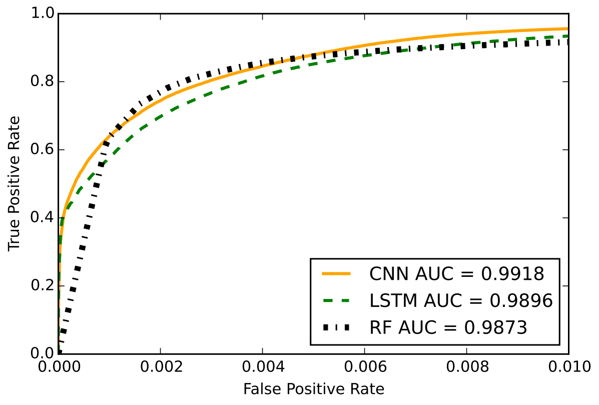

Figure 6. ROC Curves

TABLE 6. Comparison of False Positives to DGArchive (FP: Total number of FP on retrospective and prospective sets; FP on DGArchive: Number of such FP found to be known DGA domains; Percent Match: FP on DGArchive / FP)

CNN

Data Set	FP	FP on DGArchive	Percent Match
Train	638	256	40.1%
Val	219	67	30.6%
Pros1	468	123	26.3%
Pros2	643	172	26.7%
Pros3	36	2	5.6%

LSTM

Data Set	FP	FP on DGArchive	Percent Match
Train	1047	265	25.3%
Val	218	44	20.2%
Pros1	545	113	20.7%
Pros2	638	75	11.8%
Pros3	34	0	0%

(this is intended to maximize accuracy over prioritizing low FPR). The resulting true and false positive rates are shown in Table 4. We also show the percentages of each DGA family found in the Gold set in Table 5. Note that the totals for the Gold set in Table 1 only show the number of examples given to our classifier. Many examples are filtered out so that inputs follow the same form as domains in the training data (see Section 3). In computing results for the Gold set, we count any domains which were filtered out as negative predictions (except those initially removed for having overlap with the training data). These additional negative predictions are accounted for in Table 4 and Table 5 to better represent the actual performance of our approach, since the the same filtering should be applied in deployment. To summarize, we have:

- DGArchive:
  - Before deleting duplicates: 6,598,492
  - After deleting duplicates: 5,893,625
  - Positives input to classifier: 4,739,563
  - Positives counted as negative predictions from filtering: 1,154,062
- Alexa 1M:
  - Before deleting duplicates: 1,000,000
  - After deleting duplicates: 986,886
  - Negatives input to classifier: 489030
  - Negatives counted as negative predictions from filtering: 497,856

We additionally check for false positives from our retrospective and prospective sets that exist on DGArchive. Our data is not noise-free, and thus it is reasonable that some DGA domains in our data are labeled negative. Our classifier may successfully identify these domains, but these instances will be counted as false positives due to an incorrect label. Table 6 shows the total number of false positives in each set, followed by the number and percentage of false positives which were found to be real DGA domains. The significant overlap suggests that our actual FPR on real traffic may be lower than the target rate of 0.01%.

We expect these DGA domains that appear as negatives are mostly honeypots. To monitor malware activity, researchers have registered known DGA domains to track when they are queried. In our selection logic, we choose negatives as domains which are queried over a 30 day span and never get an NXDomain response. Hence active DGA domains will appear as negatives if registered as honeypots throughout our dates of observation.

6.4. Live Stream Deployment

Finally, we deploy our CNN on a live stream of resolving traffic. Each domain flagged resolves to some host IP address, which can then be inspected manually. Fig. 3 is a screenshot of our UI, which shows domains from the live stream that resolved to a given IP address within the previous seven days (the UI updates daily to show a seven day history for any given host IP encountered in the live stream). The particular IP address shown is a host resolved to by one of the domains flagged in this experiment. In this manner, we are able to gain further confidence that our model catches DGA domains on live traffic.

6.5. Analysis

At 0.01% FPR, the CNN is found to be the best performing model. However, we also observe that at different FPRs, the LSTM can attain better generalization (see Table 3). The deep learning models see a significant drop in TPR going from validation to prospective sets, which is not initially observed from the random forest. This is not surprising; with well selected features, a feature based approach should be more generalized than featureless approach, which attempts to model the training distribution directly. Still, traditional methods lack the adaptability required of a practical solution to DGAs. As for TPR_3 of the random forest, we do not have an explanation for the particularly low result.

In Table 2, the drop in prospective results appears extreme at 0.01% FPR. This effect is consistent, however. The further we tune the neural nets towards low FPR, the greater the drop in prospective results. At threshold 0.5, we observe no significant drop in performance compared to the validation set. Intuitively, slight differences in the prospective distributions is magnified by the highly restrictive threshold, resulting in what is initially perceived as an extreme drop in performance.

The deep learning approach shows markedly better performance on the truth marked Gold set (see Table 4). This may indicate the deep learning models better handle noise in our data. The CNN achieves 72.89% TPR and 0.31% FPR on the Gold set. The LSTM results in slightly better TPR (74.05%), but at significantly worse FPR (0.54%).

Inspecting classification examples, we find that our models, in general, flagged domains which contain effectively gibberish sequences (.i.e. sequences which clearly differ from natural language). Strictly dictionary-based DGA domains (those crafted by concatenating words selected from a set vocabulary) were not detected upon a manual inspection possibly due to their relatively small presence in traffic. This will be addressed in a separate project. However, domains which contain both words and gibberish sequences are caught. The only DGA to incorporate words in the Gold set is banjori, which seems to feature a mix of natural and non-natural sequences, and we see far worse performance on these domains compared to others (see Table 5). Our models are also unable to catch any instances of beebone, which (at least in our sample) features two concatenated words followed by a number, with a large variety of TLDs. Other than these, we see low performance where large portions of the DGA family are filtered out.

We originally anticipated that a CNN would be able to detect dictionary-based DGAs by learning their vocabulary. We observe that dictionary-based DGA domains make up a small portion of DGA traffic, and it is unlikely our filtering gives a strong enough signal for these types of domains to be learned. However, we have come to believe that catching these domains with high accuracy is very hard in a real-time approach. The reason for this is what we have begun to refer to as active dictionaries. The sets of words observed from dictionary-based DGAs can change on a daily basis: the dictionary is swapped such that the same DGA produces domains from new vocabulary.

Looking into classification examples, there is no obvious difference between the CNN and LSTM in terms of what is and is not classified malicious. We find that the neural networks largely agree; we show how their predictions compare in Tables 7 and 8. On the validation set, the neural nets agree on 99.985% of negative examples and 90.120% of positive examples. This suggests we can deploy both models with voting to further increase the accuracy.

An important note not yet discussed is that blocking suspicious host IPs may be left to the client's discretion. We have suggested thus far that the DNS server would refuse to return IPs which exceed a certain threshold of suspicion. Instead, a DNS response could return the host IP address as usual, but also include a score indicating the likelihood that it is malicious. This system would allow clients to choose their own threshold to better suit their security requirements as needed.

TABLE 7. COMPARISON OF CNN AND LSTM PREDICTIONS ON NEGATIVE EXAMPLES FROM VALIDATION SET

	LSTM-Neg	LSTM-Pos
CNN-Neg	2,191,881	159
CNN-Pos	159	60

TABLE 8. COMPARISON OF CNN AND LSTM PREDICTIONS ON POSITIVE EXAMPLES FROM VALIDATION SET

	LSTM-Neg	LSTM-Pos
CNN-Neg	1,246,892	114,427
CNN-Pos	110,885	808,369

6.6. CNN Prediction Time

Here we assess the time cost of prediction using the CNN model. We time prediction for 1000 domains, and repeat this test 20 times. All domains used pass filtering (i.e. SLD length $\geq$ 10) and trigger detection. The hardware used is an Amazon EC2 p2.8xlarge instance with 8 NVIDIA Tesla K80 GPUs. Averaging across the 20 trials, we find that prediction takes a single GPU 50-70ms per domain.

6.7. Future Work

Many possibilities exist for improving performance. In our experiments, we set a predetermined number of epochs for simplicity. However, the training history of the LSTM (Fig. 5) indicates it would likely benefit from further epochs. Furthermore, the CNN achieved its best accuracy on the validation data after 18 epochs (see Fig. 4). Generally, only the model weights which perform best on the validation data during training should be retained. Our results are intended to validate the hypothesis of this work, and better training practices may be used for a deployment model. However, unlike many other deep learning projects, we do not look for higher accuracy on training nor validation data sets since they are not noise-free. The objective is to have higher TPR on prospective test data sets while keeping a very low FPR.

Neural networks themselves have many parameters to tune. Primary parameters include choices such as network architecture, weight initialization, and regularization methods. Secondary parameters include choice of optimizer (and its respective parameters), activation function for each layer, loss function for weight updates, and batch size. After independently tuning these models, it may be rewarding to merge LSTM and CNN architectures for an ensemble deep learning approach. Also notable is that LSTMs can accept variable length inputs, as opposed to the fixed width padded inputs of this work. We leave exploring these options to future work.

Finally, the domain name string contains a key piece of additional information which we did not provide our networks: the TLD. DGAs tend to select TLDs to avoid collisions with legitimate traffic, and thus TLD information is highly valuable for DGA detection. We leave incorporating TLD information into our neural networks to future work.

Both CNN and LSTM models are underperforming on dictionary-based DGAs. One possible reason can be biased sample distribution between traditional and dictionary-based DGAs. An unsupervised machine learning technique could be applied to separate positive samples using above mentioned features. This can turn the two-class supervised machine learning problem into three-class. Therefore, detection performance on dictionary-based DGAs can be significantly improved in the future.

7. Conclusion

In this paper, we proposed and explored a means of applying supervised learning for real-time DGA detection. In this approach, we used simple filtering steps to obtain sufficiently pure DGA/non-DGA samples from real DNS traffic which are more representative. Such an approach is necessary for a deployable model; approaches which depend on reverse engineered malware are simply not scalable and adaptive to changes in real traffic. Moreover, we leveraged deep learning for the advantages of automatic feature extraction and the potential for online learning to keep up with changes in DGA domain patterns. We further implemented traditional methods for comparison and found that our deep networks are also superior DGA detectors. We tuned our models for a target false positive rate as low as 0.01%, and tested our approach on three prospective data sets as well as truth marked data from third parties.

The design of deep neural networks involves a myriad of design choices related to the architecture of the network, the parameters, and the optimization algorithms. Given the encouraging results obtained in this paper, a natural direction for future research is a systematic study of different character-level based CNN and LSTM architectures, and the impact of various design choices on the network's predictive accuracy for DGA detection.

References

[1] "Does Alexa have a list of its top-ranked websites?" Amazon, accessed: 2017-05-28. [Online]. Available: https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-

[2] "DGArchive," Fraunhofer FKIE, accessed: 2017-05-28. [Online]. Available: https://dgarchive.caad.fkie.fraunhofer.de/

[3] B. Yu, L. Smith, M. Threefoot, and F. Olumofin, "Behavior analysis based DNS tunneling detection with big data technologies," in *Proc. of the International Conference on Internet of Things and Big Data*, 2016, pp. 284–290.

[4] B. Yu, L. Smith, and M. 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*, 2014, pp. 258–271.

[5] S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 48–61.

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

[7] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: Dga-based botnet tracking and intelligence," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2014, pp. 192–211.

[8] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *USENIX Security Symposium*, vol. 12, 2012.

[9] S. Krishnan, T. Taylor, F. Monrose, and J. McHugh, "Crossing the threshold: Detecting network malfeasance via sequential hypothesis testing," in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2013, pp. 1–12.

[10] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," *preprint arXiv:1611.00791*, 2016.

[11] "OSINT feeds from Bambenek Consulting," Bambenek Consulting, accessed: 2017-05-28. [Online]. Available: http://osint.bambenekconsulting.com/feeds/

[12] Farsight Security, accessed: 2017-05-28. [Online]. Available: https://www.farsightsecurity.com/

[13] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, 1948.

[14] Google Books Ngram Viewer. [Online]. Available: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, accessed: 2017-05-28. [Online]. Available: http://www.deeplearningbook.org

[16] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, 2015, pp. 649–657.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, accessed: 2017-05-28. [Online]. Available: http://tensorflow.org/

[19] F. Chollet. Keras. Accessed: 2017-05-28. [Online]. Available: https://github.com/fchollet/keras

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *preprint arXiv:1301.3781*, 2013.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. of the 13th International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *preprint arXiv:1412.6980*, 2014.

[23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.