2021 IEEE 5th International Conference on Cryptography, Security and Privacy

DGA Domain Detection using Deep Learning

Haleh Shahzad
Cyber Security Analytics
TELUS
Toronto, Canada
haleh.shahzad@telus.com

Abdul Rahman Sattar
Cyber Security Analytics
TELUS
Toronto, Canada
Abdul_Rahman.Sattar@telus.com

Janahan Skandaraniyam
Cyber Security Analytics
TELUS
Toronto, Canada
Janahan.Skandaraniyam@telus.com

2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP) | 978-1-7281-8621-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSP51677.2021.9357591

***Abstract*—Domain generation algorithms (DGAs) are used by attackers to generate a large number of pseudo-random domain names to connect to malicious command and control servers ($C\&Cs$). These domain names are used to evade domain based security detection and mitigation controls. Reverse engineering of malware samples to discover the DGA algorithm and seed to generate the list of domains is one of the techniques used to detect DGA domains. These domains are subsequently preregistered and sinkholed, or published on security device blacklists to mitigate malicious activity. This technique is time-consuming and can be easily circumvented by attackers and malware authors. Statistical analysis is also used to identify DGA domains over a time window, however many of these techniques need contextual information which is not easily or feasibly obtained. Existing studies have also demonstrated the use of traditional machine learning techniques to detect DGA domains. Our goal was to detect DGA domains on a per domain basis using the domain name only, with no additional information. This paper presents a DGA classifier that leverages a recurrent neural network (RNN) based architecture for the detection of DGA domains without the need for contextual information or manually created features. We compared the performance of different RNN based architectures by evaluating them against a dataset of 2 million plus domain names. The results indicated little difference in performance metrics among the RNN architectures.**

***Index Terms*—Cybersecurity, DGA, deep learning, LSTM, Bi-LSTM, GRU**

I. Introduction

Attackers can control a group of malware-infected machines called botnets [1], remotely through command and control ($C\&C$) servers. Malware families typically use domain generation algorithms (DGAs) to remain elusive, prevent $C\&Cs$ from being taken down and elongate the potency of malware. DGAs are used to generate pseudo-random domain names and the $C\&C$ registers a subset of those domain names and uses them to communicate with the malware. The generated sequence of domain names are pseudo-random due to the initial seed used by the DGA to generate them. Both the malware on the infected machine and the $C\&C$ use the same DGA and seed to generate a sequence of DGA domain names. If the malware tries to communicate with a domain name registered by the $C\&C$, the communication and thus resultant malicious activity becomes successful. The $C\&Cs$ cycle through DGA domain names frequently and the malware on the infected machine will eventually connect to the $C\&C$ server. The process is shown in Fig. 1. Various techniques can be employed to build a DGA which result in a complexity range of generated domains from simple, uniformly generated domain names, to those that attempt to follow real domain name distributions [2].

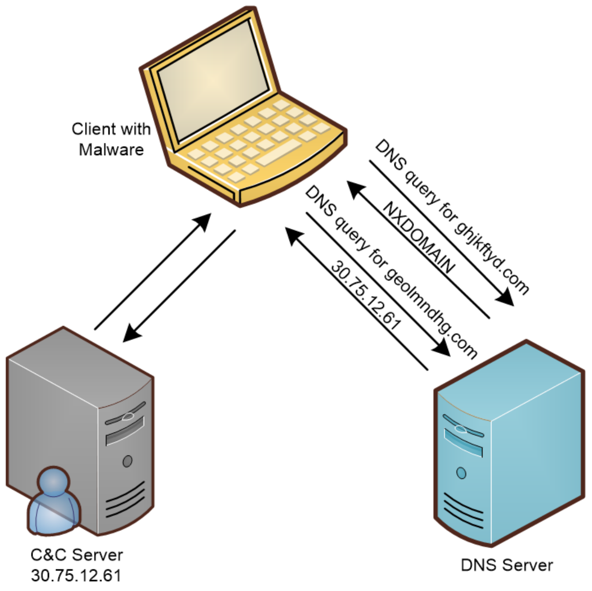

Fig. 1: Typical example of malware using a DGA to connect to a $C\&C$

Ideally, we should detect DGA domains on a per domain basis using a DGA classifier before any $C\&C$ communication begins. This classifier can live in the network and sniff DNS requests to identify DGA domains. When a DGA domain is detected, the classifier can notify other automated tools or network administrators to further validate the veracity of the detection while any further communication with the potential DGA domain is blocked. Such real-time detection is a considerably harder problem to tackle and often the performance of existing techniques is too low for real-world deployments.

This paper compares the performance of a feature-less technique using different RNN architectures to classify DGAs. The RNN architectures whose performance were compared are: Long Short-Term Memory networks (LSTMs), bidirectional LSTMs (BiLSTMs) and GRUs. This DGA classifier is featureless and operates on raw domain names (e.g., google, yahoo). If a new family of DGA domains is discovered, then the classifier can be retrained without the need for manually extracting features. Our classifier works largely as a black box making it very difficult for adversaries to reverse engineer with the aim to circumvent DGA domain detection. This DGA classifier can also be utilized in a multiclass classification to identify the unique family that a DGA domain belongs to with absolutely no contextual information.

II. Related Work

Reverse engineering malware binaries is one approach to detect DGA domains as presented in [3]. Once the DGA algorithm and the seed are known, the sequence of DGA domain names that it will try to connect to, and thus those that will be registered by its $C\&C$ can be known and appropriate actions implemented for mitigation, subversion, and intelligence gathering purposes. Network administrators can blacklist DGA generated domains on security devices to block connections to potential $C\&Cs$ and prevent malicious network activity. Sinkholing is a technique that can be used to redirect traffic from its intended destination to another server. This server can pose as an impostor $C\&C$ by registering the DGA domain names. Once a successful sinkhole is in place, malware activity can be monitored, tracked, and hijacked to the extent of rendering the malware impotent. For example, a campaign involving botnets using DGAs needs to redeploy new botnets with new seeds to continue [2].

Reference [4] discussed the effectiveness of the blacklisting approach which is based on adding DGA generated domains into a blacklist used by security control devices to block connections to the $C\&C$. Both static blacklist creation and sinkholing are only effective when both the DGA and seed used by a campaign are known [2]. Furthermore, the amount of domain names that can be generated by a DGA is significant enough such that blacklisting and sinkholing all of them when only a subset may actually be used, makes for an inefficient and tedious process. Also, for instances where the seed gets changed dynamically, it becomes even more cumbersome.

Due to the deficiencies of the previous approaches, supervised and unsupervised machine learning approaches were proposed to overcome those deficiencies. Retroactive detection systems and real-time detection systems have been proposed in machine learning based methods. References [5] [6] [7] well explored the first category through means such as clustering, but these approaches have disadvantages. These approaches are based on the analysis of batches of traffic data which implies a time delay before detection and thus that detection will happen after the malware has communicated with the $C\&C$ enough to likely cause damage. These techniques also incorporate contextual information such as HTTP headers, NXDomain responses, and historical DNS resolution data to improve performance, but gathering this information is costly and sometimes not feasible.

Manually extracted features such as string length, vowel ratio, digit ratio, character probability distribution are fed into a machine learning model in many real-time approaches [8] [9]. Manual feature extraction is a time consuming process and these features are easy to circumvent by adversaries. Adversaries can create a new DGA which takes the feature set of a classifier into account in order to circumvent detection. An adversary creating such a DGA should have access to the benign ground truth to mimic the features of benign domains when generating DGA domains. Adversaries can also utilize a classifier implementation to test their DGA domains to make sure they can effectively lower the accuracy of the classifier [10]. If an attacker creates a new DGA family using such a process, new features would need to be identified to combat them.

Reference [11] utilized an autoencoder to generate the 16 bit domain representation and then classified the domain name using SVM. In [12], a two level architecture using machine learning is proposed to detect DGA domains. Domains are first classified into DGA and non-DGA classes using a machine learning algorithm. Seven different machine learning classifiers are tested and it is claimed that the decision tree based classifier has the best performance. The class of DGA domains are then fed into the DBSCAN clustering algorithm to identify the families that the DGA domains belong to. It is clear that using any machine learning algorithm needs an efficient feature engineering process and the features might also need to be updated.

A network intrusion detection model based on deep belief networks (DBN) is proposed in [13]and trained in three stages using attribute features that are provided in the KDD CUP 1999 dataset. In [14], a contextual LSTM deep neural network is introduced to take both contextual features and metadata to detect bots. Reference [2] proposed a method using a character level LSTM with a shallow architecture to predict DGA domain names. The architecture consists of an embedding layer, one LSTM layer and one logistic regression layer. They used synthetically generated labeled data as samples of DGA and non-DGA domains for training. Reference [15] explored the performance of both CNN and LSTM architectures to detect DGA domains. Both models performed at the character level with only the domain name string used as input without any other contextual information. But here, the models were only tuned for a target false positive rate using a shallow architecture. They also filtered real DNS traffic to obtain samples of DGA and non-DGA domains for training.

III. Model Architecture

In this paper, we compared the performance of a DGA classifier based on the following RNN architectures: Unidirectional LSTM network, bidirectional LSTM (Bi-LSTM) and GRU. The problem of vanishing gradients is a key motivation behind the application of the Long Short-Term Memory (LSTM) architecture [16], [17], [18], which consists of LSTM cells with a set of gates to control the flow of information. The design of the LSTM cells allows LSTM to learn long-term dependencies. The input in unidirectional LSTM is from the past therefore only information from the past is preserved while in Bi-LSTM, the input is two-way, one from the past to the future and the other one from the future to the past and

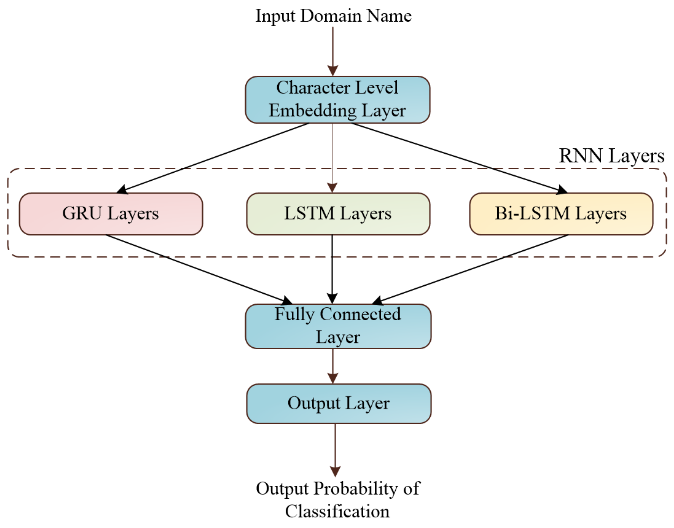

Fig. 2: System Architecture

therefore information from the future can be preserved, i.e. the original data will be fed from the end to the beginning as well. The GRU is quite similar to LSTM but it only has two gates, a reset gate and an update gate. The update gate acts similar to the forget and input gates of LSTM. It decides what information to discard and what new information to add. The reset gate is used to decide how much past information to forget. In the use of domain names as the input sequence, the RNN based architectures will learn the important combination of letters that differentiate DGA from non-DGA domains.

The input to our model is only the second level domain (SLD) which is directly below a top level domain(TLD), i.e the input is “google” for “google.com”. The model works with variable-length character sequences as input and is comprised of a character level embedding layer, RNN layers (LSTM, Bi-LSTM or GRU layers), a fully connected layer and finally a single node output layer with sigmoid activation; The architecture is presented in Fig. 2. Each unique character in the input domain name is mapped to its own vector representation of dimension d in embedding layer where d can be tuned. The input characters are non-redundant, lowercase alphanumeric, period, dash and underscore. The inputs to the embedding layer are indexes of the input domain name characters.

We employed Torchtext library [19] to tokenize the input domain name into separate characters and assign indexes to them. We pad zeros to the end of the domain name string so that all inputs have a fixed length. We then embed the input characters using the embedding layer and the embedding is learned during training. The RNN layers learn patterns of characters from embedded vectors to distinguish the DGA domain names from non-DGA domain names. Dropout is applied after the the RNN layers and prior to the fully connected layer and also in between the RNN layers to prevent overfitting.

IV. Experimental Setup

In the following section, we describe details of our experimental setup for evaluating our DGA classifier in a binary experiment (DGA vs. non-DGA) using publicly available benign and DGA domain name datasets. We trained our model using Python 3, Pytorch [20] and an NVIDIA DGX Station (GPU) that provided the computational power [21].

A. Performance Metrics

Our DGA classifier using LSTM, BiLSTM and GRU has been evaluated on popular datasets that are presented in Section IV-B. The performance of a model can be evaluated according to certain metrics. A confusion matrix is shown in Table I and different performance metrics such as $Accuracy$ can be calculated based on that. The number of instances accurately or inaccurately predicted by a model can be tabulated in the form of a confusion matrix. The confusion matrix is generally represented by four values which are TP, TN, FP and FN.

TABLE I: Confusion Matrix Represented by four Parameters

		Predicted	
		Non-DGA	DGA
Actual	Non-DGA	TN	FP
	DGA	FN	TP

The following are the performance metrics based on the confusion matrix. $Accuracy$ is the ratio of the accurately classified DGA and benign domain names to the entire dataset. $Accuracy$ is defined as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

FPR, False Positive Rate, is defined as the proportion of benign domains mistakenly predicted as DGA domains to all benign domains:

$$FPR = \frac{FP}{FP+TN} \quad (2)$$

TPR, True Positive Rate, also called sensitivity or recall, is defined as the proportion of DGA domains that are correctly identified to all DGA domains. It gives the likelihood that the classifier can predict positive instances (DGA domains) correctly.

$$TPR = \frac{TP}{TP+FN} \quad (3)$$

The trade-off between TPR to FPR is measured by a ROC curve. AUC shows the area under the ROC curve, and the closer the AUC is to 1, the better our classifier is performing. The metrics that are used to evaluate our model are $Accuracy$, TPR, FPR and AUC.

B. Data

We used the Alexa Top 1M domains [22] and the Cisco umbrella popularity list [23] for non-DGA samples. The OSINT DGA feed from Bambenek consulting was used for DGA domains and contains varying numbers of samples from thirty DGA families [24]. 44 families of DGA domains that are provided in Netlab 360 [25] were also used as DGA domain samples in our experiments. The data sources that we used are summarized in Table II. The number of samples in the original dataset is shown in the second column of Table II. In our experiments, we only used the SLD as the input, i.e. "google" from "google.com". As a result, for instance, the entries of "google.com" and "google.ca" in the original dataset resulted in duplication in our case. After deduplication, the number of unique samples in each dataset for our experiments is shown in the third column of Table II.

TABLE II: Description of data sources

Data Sources	**Number of samples before deduplication**	**Number of samples after deduplication**
Alexa	1,000,000	905,832
Cisco	1,000,000	492,840
OSINT	833,222	787,442
Netlab 360	72,101	69,325

V. Results

Pytorch, an industry preferred deep learning research library, was used to implement the model. The model's performance was assessed using a dataset that was culled from the data sources presented in Section IV-B. The dataset contained legitimate samples from the Alexa dataset [22] and Cisco umbrella popularity list [23], and DGA domain samples from the OSINT dataset [24] and Netlab 360 [25], together totaling 2 million plus samples. We used the percentages of 0.8, 0.1, 0.1 for training, validation and test data, respectively, in our experiment.

The Pytorch built-in embedding layer was used and the model was trained using a batch size of 16 using the architecture outlined in Section III. After the embedding layer (first layer), the RNN layer is comprised of LSTM, BiLSTM or GRU (a number of cells with default Tanh activation), a fully connected layer (100 nodes), and finally a single node output layer with sigmoid activation function. If the output probability is greater than 0.5, the domain name is identified as a DGA domain name, otherwise, it is identified as a non-DGA domain name. We obtained better results using the RMSprop optimization algorithm as opposed to the Adam optimizer, and the learning rate was set to 10^{-5}. We also applied a dropout of 30 % at the RNN layer.

The model was trained for 15 epochs with 3 layers and 400 cells. An $Accuracy$ of 87% and a TPR of 81% over a test set which contained 10% of the samples of the dataset were obtained for the LSTM-based model. The Bi-LSTM architecture achieved an $Accuracy$ a little less than the LSTM based model while the GRU architecture did not have a relatively higher performance although it decreased the training time due to its simpler architecture. The ROC curve of the test set for the LSTM based architecture is shown in Fig. 3. The AUC of about 0.98 is achieved for the LSTM based architecture which shows the good performance of the classifiers over more than 200,000 test samples.

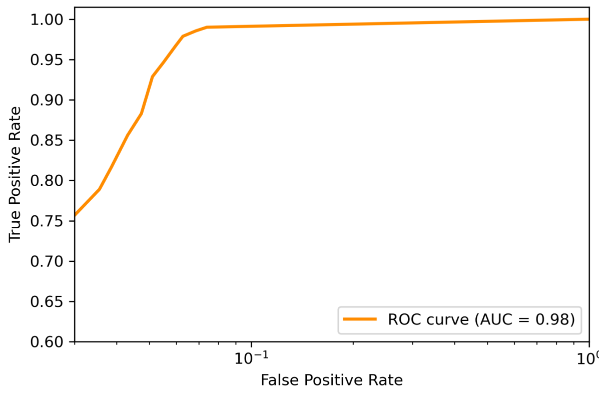

Fig. 3: ROC curve for LSTM based model

VI. Conclusion

This paper presented an approach which uses RNN based architectures to detect DGA generated domains. The performance of the following RNN architectures were compared: LSTM, Bi-LSTM and GRU. This approach uses only the raw domain names as input, and therefore does not require manual feature creation which is cumbersome to maintain. RNN based DGA classifiers require only the domain name from DNS queries as input and therefore can be deployed with minimal complexity in an IT environment. They are also highly scalable and adaptable to changes in real-world traffic. Evaluation on publicly available datasets have shown that RNN based DGA classifiers performed very well. TLD information is also highly valuable for DGA detection and will be incorporated in our future work. Detecting dictionary-based DGAs via the RNN based model more effectively will also be the focus of our future work.

References

[1] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets.," *SRUTI*, vol. 5, pp. 6–6, 2005.

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

[3] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna, "Analysis of a botnet takeover," *IEEE Security & Privacy*, vol. 9, no. 1, pp. 64–72, 2010.

[4] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 1–21, Springer, 2014.

[5] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: detecting the rise of dga-based malware," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pp. 491–506, 2012.

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

[7] 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.

[8] 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*, pp. 192–211, Springer, 2014.

[9] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting p2p botnets through network behavior analysis and machine learning," in *2011 Ninth annual international conference on privacy, security and trust*, pp. 174–180, IEEE, 2011.

[10] J. Spooren, D. Preuveneers, L. Desmet, P. Janssen, and W. Joosen, "Detection of algorithmically generated domain names used by botnets: a dual arms race," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 1916–1923, 2019.

[11] B. Dahal and Y. Kim, "Autoencoded domains with mean activation for dga botnet detection," in *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, pp. 208–212, IEEE, 2019.

[12] Y. Li, K. Xiong, T. Chin, and C. Hu, "A machine learning framework for domain generation algorithm-based malware detection," *IEEE Access*, vol. 7, pp. 32765–32782, 2019.

[13] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *2014 Second International Conference on Advanced Cloud and Big Data*, pp. 247–252, IEEE, 2014.

[14] S. Kudugunta and E. Ferrara, "Deep neural networks for bot detection," *Information Sciences*, vol. 467, pp. 312–322, 2018.

[15] B. Yu, D. L. Gray, J. Pan, M. De Cock, and A. C. Nascimento, "Inline dga detection with deep networks," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 683–692, IEEE, 2017.

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

[17] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[18] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[19] Torch Contributors , "Torchtext package." https://pytorch.org/text/.

[20] Facebook's AI Research lab, "Pytorch library." https://pytorch.org/text/.

[21] Nvidia, "Nvidia dgx station." https://docs.nvidia.com/dgx/dgx-station-user-guide/.

[22] Amazon, "Does alexa have a list of its top ranked websites." https://support.alexa.com/hc/en-us/articles/200449834.

[23] Cisco, "Cisco umbrella popularity list." http://s3-us-west-1.amazonaws.com/umbrella-static/index.htm.

[24] Bambenek Consulting, "Osint bambenek consulting feed." http://osint.bambenekconsulting.com/feeds/.

[25] Network Security Research Lab at 360, "Netlab dga project." https://data.netlab.360.com/.