DGA Detection Using Parallel CNNs

Matthew Jansen
Oregon State University
Corvallis, Oregon
jansemat@oregonstate.edu

Abstract

In the past decade, malware has implemented domain generation algorithms (or DGAs) to evade network intrusion detection systems with a higher success rate. Malware utilizes DGAs to randomly generate a fully-qualified domain name, which serves as a connection to a hacker controlled command and control (C2) server. This makes it difficult for cybersecurity analysts to rely on blacklists and cyber threat intelligence to detect and respond to compromises in their computer network. In this paper, previous work on the topic of DGA detection through machine learning and deep learning models is reviewed. Of these models, we examine and implement the Invincea model (using PyTorch) to detect DGA-generated domain names. Further, we implement changes to this model to examine its performance with respect to runtime and accuracy in comparison to the default model. Finally, the results derived from experiments using the altered models and the practicality of using this model for incident response investigations or for network intrusion detection systems are examined.

1. Introduction

Malware infections are on the rise, with 560,000 new pieces of malware detected every day [4]. Additionally, malware families have been seen implementing domain generation algorithms (DGAs), which allows the malware to generate and query fully-qualified domain names (FQDNs) on the fly, which often goes undetected by network intrusion detection systems. Currently, over 40 malware families employ DGAs, and in 2019 over 172 million randomly-generated domains were identified by SonicWall [2]. Typical network security monitoring solutions use signature-based detection which can raise alerts when specific file hashes, URLs, or other static indicators are found over the network. Due to the dynamic nature of DGAs, domains generated by DGAs can circumvent signature-based detection solutions.

Having the ability to detect DGA-generated FQDNs would be extremely useful to security operations centers, and would have a positive impact on the network security monitoring solutions utilized by security operations centers. In this paper, previous work done towards the detection of DGA-generated algorithms through machine learning and deep learning models is explored. Further, the Invincea model for detecting malicious strings [5] is implemented using *PyTorch*, and the performance of this re-implementation is examined. Additionally, alterations were done to the model and the corresponding performance is compared to the default model, as well as the performance outlined in the original Invincea paper. We conclude by showing how results from the altered models may prove beneficial towards the shift from static-based detection mechanisms to deep learning-based detection.

2. Related Work

Previous works that have been completed in the area of DGA detection can be classified into machine learning and deep learning approaches. Machine learning approaches utilize lexical features derived from FQDNs to train the model. In [7], researchers found that random forests trained with 100K examples using features such as character entropy levels, hex character ratio, vowel character ratio, etc. produced the best results among other feature-based models that they tested. For deep learning approaches, the majority of models used for DGA detection seems to include CNNs and LSTMs. In [6], researchers utilize LSTMs to detect DGAs, and researchers in [3] use both forward and backward LSTM layers to perform natural language processing of Twitter posts. Additionally in [5] and [9], researchers use parallel and stacked CNNs (resp.) to perform DGA detection and text classification (resp.). All of the above-mentioned deep learning models were part of a malicious domain name competition in 2018 [1]. The experimental results from these models were compared to previously mentioned work related to feature-based detection, and were shown to exceed in detection capabilities.

3. Technical Approach

For the purpose of detecting DGA domains, we decided to implement the Invincea model, which was developed by researchers at Invincea, Inc. in 2017 [5]. Originally this model was developed to detect all sorts of malicious strings that may exist on a computer, including full URLs, file paths, and registry keys. This model was re-implemented by Yu et. al. in the following year, where it was compared against other similar models for the purpose of detecting malicious DGA domains, and found to have the highest non-ensemble accuracy scores among the other competitors [8]. In the remainder of this section, we will summarize the architecture of the Invincea model.

3.1. Invincea Model Overview

In a general sense, the Invincea model takes FQDNs as input and produces a binary classficiation which translates to whether or not the FQDN is malicious (i.e. was generated by a DGA) or benign. To accomplish this the model first passes the FQDN into a character embedding layer, which embeds each character of the FQDN into a vector, the values of which are altered as part of the backpropagation phase. Each input FQDN is padded to always be 200 characters in length. The collection of character embeddings are formed into a matrix, which is then passed into four independent CNN layers, each with varying kernel sizes. The output of each CNN is then normalized through layer normalization (as opposed to batch normalization), sum pooling, and finally a 50% dropout layer. Each normalized CNN output is then concatenated together into a single vector. This vector is then passed as input into two fully-connected linear layers with ReLU activation. The output of the final fully-connected layer is then fed into a single linear layer with sigmoid activation, which will produce the classification score associated with the input FQDN. A graphical overview of the model from the original Invincea paper is shown in Figure 1.

4. Experiments

The goal the upcoming experiments is to see how changes to the Invincea model can impact the performance of the model (or have no impact on performance) when detecting DGA-generated FQDNs. In this section, we will discuss the Invincea architecture in further detail in order to establish a baseline model, and we will also define a series of alterations to be made from the baseline. The datasets used to train these models will be discussed, as well as the results of running the models using this data. Finally, we finish by discussing future works.

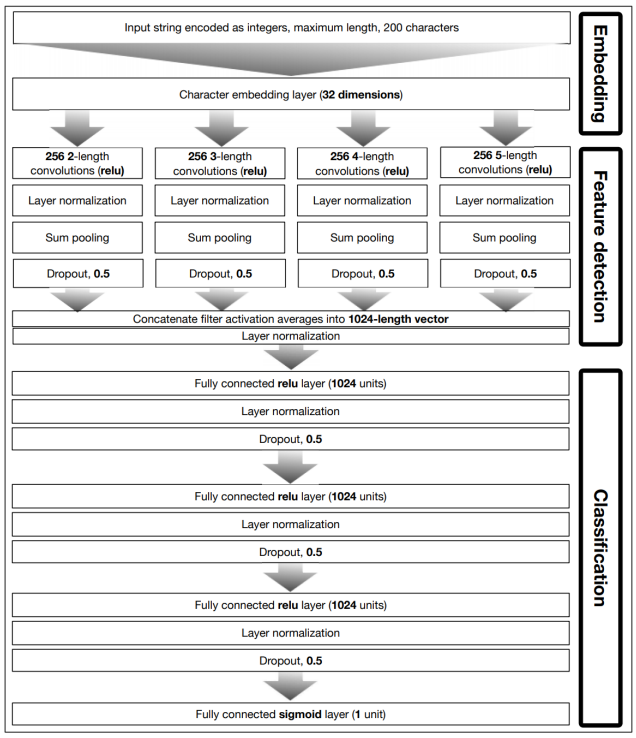

Figure 1. Invincea Model Architecture Overview

4.1. Datasets

For our dataset we require sets of two types of domains, malicious domains generated by DGAs, and legitimate (benign) domains. For the benign domain dataset, we downloaded the Cisco Umbrella 1 million dataset [1], which contains the top 1 million most visited domains on the internet. For the malicious domain dataset, we used the Netlab-360 dataset[2] which olds over 1.4 million malicious domains generated from malware which utilizes DGA functionalities. With 1 million benign domains and over 1 million malicious domains, for each batch we take 50% of the batch size of benign domains and 50% of malicious domains, and randomly permute the batch before running it through our model.

4.2. Experimental Design

In our experiments, alterations will be made to batch sizes, the character embedding layer, the size of CNN outputs, and the number of nodes in a linear layer. For each model we define, we will run the model for 40 epochs. After each epoch, the model will be evaluated with respect to accuracy, binary cross-entropy loss score, and false positive/negative rates for both the training and testing sets. The

[1] https://umbrella.cisco.com/blog/cisco-umbrella-1-million

[2] https://data.netlab.360.com/dga/

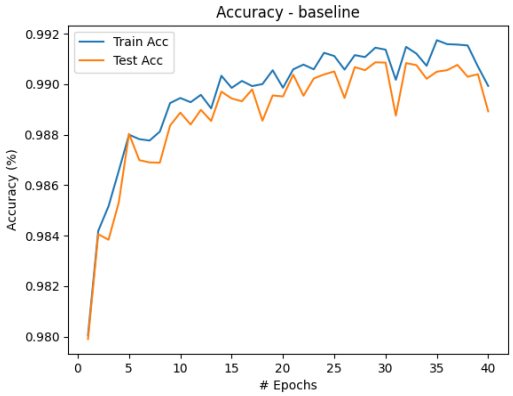

Figure 2. Baseline Model Accuracies

models derived from [5] and [8] were implemented using *PyTorch*.

For the baseline model, we use the configuration that was described in both [5] and [8]. Specifically, we have batch sizes of 256 (128 benign and 128 malicious domains), 32-length character embeddings, CNNs which output in 256 dimensions, and each linear layer having 1024 nodes.

For the altered models, we change every numerical value defined in the baseline model. For batch sizes, we define 4 models which vary from the baseline by having 64, 128, 512 and 1024 batch sizes. An additional 3 models will have character embedding sizes of length 8, 16, and 64. An additional 4 models will have convolutional output sizes of 32, 64, 128 and 512. Finally we define 4 more models where the first linear layer has 256 or 512 nodes, and the second linear layer has either 256 or 512 nodes as well - note that only one linear layer is changed for these altered models.

4.3. Results

Now we will discuss the how the baseline and altered models performed both independently, and compared to one another. Additionally, the observations made from these experiments and how they have implications towards the advancement of DGA-detection technologies will be examined.

4.3.1 Baseline Model

The baseline model was shown to score extremely well in all fields that we were looking for - high accuracy, low loss scores, and low false positive/negatives rates. Specifically, the baseline model had scored consistently over 98.0% accuracy, with peak accuracy in the testing dataset being around 99.0%. Additionally, the baseline model's binary cross-entropy loss scores were consistently below 0.00015 (low scores due to averaging over large datasets), and had

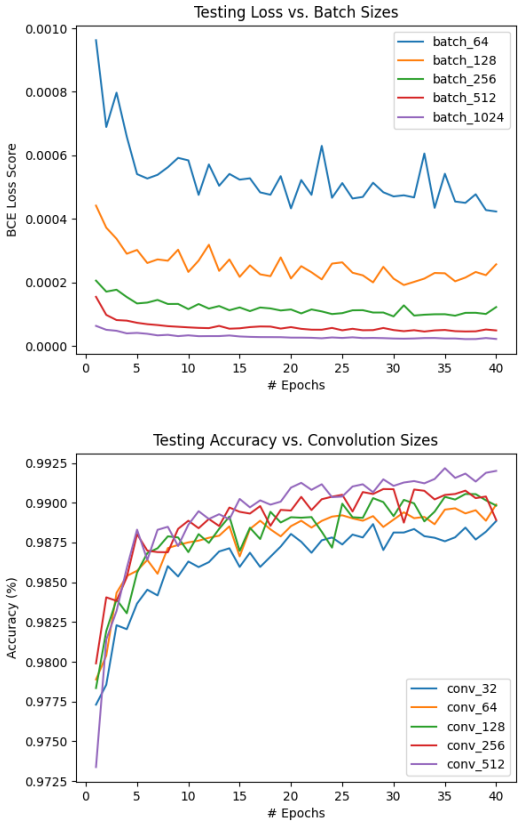

Figure 3. Batch Size & Convolution Size Accuracies

false positive/negative rates consistently below 1.0%. The accuracy rates over all 40 epochs used to train the baseline model over both the training and testing dataset can be found in Figure 2. These results line up with results from [8], where the researchers found their implementation of the Invincea model having 98.9% accuracy with 0.1% false negative rates.

4.3.2 Altered Models

For the altered models, we noticed that changing the size of the convolutions and changing the batch sizes had a potentially negative impact on the performance of the model. With respect to changing the batch size, there was a clear pattern in the loss scores which indicated that decreasing the batch sizes resulted in increased average loss scores. However, this is believed to be due to the averaging of the loss scores, as although the loss scores seem to be impacted by this change, it was also noted that there was no clear pattern of changes in the accuracy associated with those loss scores. Although, it was also noted that changes in the size of the convolution outputs had an impact on the performance of the model. Specifically, as convolution size decreased, there

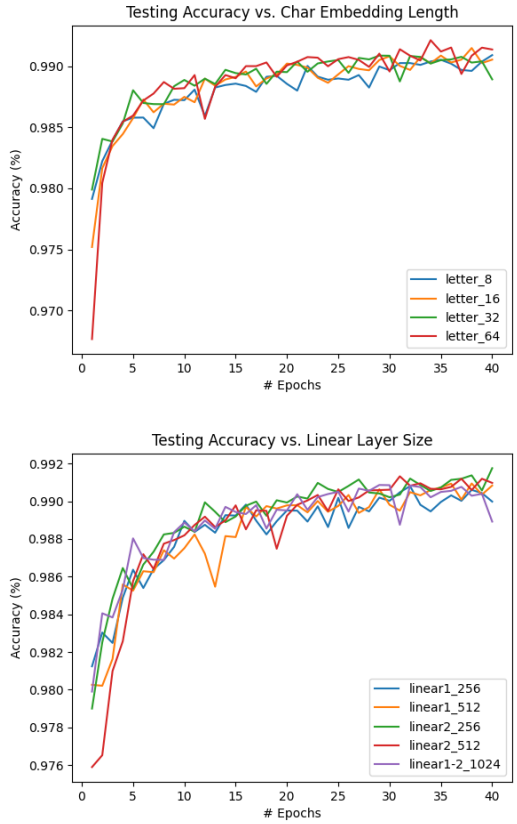

Figure 4. Character Embedding & Linear Layer Accuracies

was a noticable decrease in accuracy and increase in loss scores. In Figure 3, both of these impacts are shown which indicate clear trends or separations in either loss scores or accuracy for the altered models with respect to the results of the baseline model.

Additionally, we noticed that changing the length of the character embedding vectors or changing the number of nodes in a single linear layer had no effect on accuracy, loss scores, or false positive/negative rates. We found this to be insightful, as even reducing a single linear layer from 1024 nodes to 256 (in either the first or second linear layer) had no impact on the models performance. Figure 4 shows the accuracy rates over the testing set for both character embedding sizes, and number of linear nodes. Notice that there are no trends between the given embedding/linear layer sizes and increasing/decreasing accuracy rates. This behavior can also be seen in the loss scores, false positive and false negative rates.

Acknowledging when changing the model will impact performance is important, however keeping track of when the model is not impacted should also be a priority. Thinking practically, this model can likely be used as a tool to sift through huge amounts of network traffic to find malicious domains, or be utilized by a real-time network traffic monitor to raise alerts upon detecting malicious domains. In both of these cases, speed is of the upmost importance - especially for real-time monitoring. Larger models may impact the speed of evaluating a single domain, so observing when it's possible to reduce to a simpler model can be of the upmost importance.

4.4. Future Work

We save enumerating all possible models stemming from the baseline Invincea model as future work, along with combining reductions to obtain even simpler models (e.g. changing number of nodes in both linear layers instead of just one). Additionally, finding more benign datasets would be helpful, especially ones with a focus on users in a particular sector, for example residential or business environments. Finally we hope to create a lightweight evaluation tool which uses this model to sift through DNS data and find malicious domains, in order to assist security operations centers acting in an incident response capacity.

5. Conclusion

In conclusion, we have introduced related work to the detection of DGA-generated domains, and have focused on the Invincea model to do so. We re-implemented this parallel CNN model using the outline from the original Invincea paper, and were able to produce results comparable to the 2018 re-implementation. From this baseline, we defined a set of alternative models which all had a single minor change applied to them, in order to observe any impact on performance. We identified that changes to the convolutional output and batch sizes may impact performance negatively, while changing the size of the character embeddings or a single linear layer had no noticable impact on performance. We also identify how reducing the size of this model can positively impact security operations centers in an incident response capacity, and outline building tools using these models for future work.

References

[1] C. Choudhary, R. Sivaguru, M. Pereira, B. Yu, A. C. Nascimento, and M. De Cock. Algorithmically generated domain detection and malware family classification. In S. M. Thampi, S. Madria, G. Wang, D. B. Rawat, and J. M. Alcaraz Calero, editors, *Security in Computing and Communications*, pages 640–655, Singapore, 2019. Springer Singapore.

[2] S. Cook. Malware statistics and facts for 2021, 2021.

[3] B. Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen. Tweet2vec: Character-based distributed representations for social media, 2016.

[4] B. Jovanović. A not-so-common cold: Malware statistics in 2021, 2021.

[5] J. Saxe and K. Berlin. expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv preprint arXiv:1702.08568*, 2017.

[6] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. Predicting domain generation algorithms with long short-term memory networks, 2016.

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

[8] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock. Character level based detection of dga domain names. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[9] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification, 2016.