Nemesis: Detecting Algorithmically Generated Domains with an LSTM Language Model

Dunsheng Yuan[1,2], Ying Xiong[3], Tianning Zang[2(✉)], and Ji Huang[1,2]

[1] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{yuandunsheng,zangtianning}@iie.ac.cn
[3] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China

Abstract. Various malware families frequently apply Domain Generation Algorithms (DGAs) to generate numerous pseudorandom domain names to communicate with their Command and Control (C&C) servers. Security researchers make a lot of efforts to detect Algorithmically Generated Domains (AGDs) for fighting Botnets and relevant malicious network behaviors. In this paper, we propose a new AGD detection approach, Nemesis, based on a Long Short-Term Memory (LSTM) language model. Nemesis can identify whether given domain names are AGDs according to their string compositions, and without additional information. Nemesis first leverages an n-gram dictionary, which is built on real domain names, to tokenize domain names into n-grams. Then a pre-trained detector is used to classify domain names as real ones or AGDs according to the tokenized results. We evaluate Nemesis' abilities to detect domain names generated by known DGAs and to discover new DGA families. It turns out that Nemesis can accurately detect AGDs with the precision of 98.6% and the recall of 96.7%. Besides, we verify that Nemesis largely outperforms several existing effective approaches.

Keywords: Domain Generation Algorithm · LSTM · Language model · Deep learning

1 Introduction

The Domain Name System (DNS), which resolves domain names into IP addresses, is an important public infrastructure and significant for the collaboration of the Internet. However, this mechanism can also be abused by malware to communicate with their Command and Control (C&C) servers. Since it can be easily blocked by blacklists to use hard-coded IP addresses or domain names to establish C&C connections, a variety of malware families apply a more sophisticated mechanism known as Domain Generation Algorithms (DGAs) to hide

X. Wang et al. (Eds.): CollaborateCom 2019, LNICST 292, pp. 350–363, 2019.
https://doi.org/10.1007/978-3-030-30146-0_24

their C&C servers [1]. In a botnet, such as Conficker and Mirai, each compromised computer (bot) algorithmically generates a large set of domain names and queries each of them until one of them is resolved successfully, and then the bot contacts the resolved domain name, which is typically corresponding to the IP address of the C&C server (botmaster). Once the connection is established, the bots can be controlled by the botmaster to launch distributed denial-of-service (DDoS) attacks, steal data and privacy, mine digital currency illegally, *etc* [2]. For example, on October 21, 2016, a large DDoS attack on Dyn, a DNS provider was performed through a Mirai botnet, which involved 100,000 malicious endpoints. The Mirai botnet sent superfluous DNS requests to overload the DNS servers, and the serious consequences of this attack caused dozens of popular websites unreachable for the users in North America. Besides botnets, spammers also generate pseudorandom domain names in spam emails to avoid detection by regular expression based domain blacklists [3].

Since Algorithmically Generated Domains (AGDs) are involved in various malicious network behaviors mentioned above, it becomes a crucial topic of concern for researchers to detect AGDs automatically and accurately. Some traditional AGD detection approaches leverage the distribution of characters in the domain [3]. These approaches are simple but hard to achieve good effects. Others utilize human engineered lexical features [4,5], nevertheless, it is time-consuming to construct effective features.

Motivated by these reasons, we propose a new AGD detection approach called Nemesis[1], which is implemented based upon an LSTM language model. Nemesis first tokenizes a domain name into n-grams and then classifies it as a real domain name or an AGD according to these n-grams. The key insight of Nemesis lies in the truth that domain names are composed of syllables or acronyms for easy readability, and n-grams can represent both of them. Nemesis only mines the n-gram information of domain names but can still keep high precision and recall. Specifically, we make the following key contributions:

- Nemesis can identify whether a single domain name is an AGD according to its string composition. It does not need extra data or the association information of multiple domain names.
- Nemesis can detect AGDs of known DGA families with high precision of 98.6%, recall of 96.7%, an F1-Score of 97.6%, and it largely outperforms detection approaches based on KL, ED, and JI.
- We verified that Nemesis is also able to discover new DGA families, which are not seen in the training data. Specifically, we test Nemesis on AGDs of 10 new DGA families individually, and it can achieve high recall values for every DGA family.

The rest of the paper proceeds as follows. Section 2 summarizes related work in DGA detection and discusses their limitation. Section 3 introduces an overview of our approach and then describes how to implement each module in detail. Section 4 presents the experimental setup, including datasets and evaluation metrics. After that, we compare Nemesis with a typical prior work in Sect. 5. Finally, we conclude our work in Sect. 6.

[1] Nemesis is the goddess of retribution for evil deeds in ancient Greek mythology.

2 Related Work

Researchers propose a variety of approaches [3–10] to detect AGDs. Antonakakis *et al.* [4] develop a detection system called Pleiades to identify DGA-based bots. Their insight is that most DGA-generated domains queried by bots will result in Non-Existent Domain (NXDomain) responses, and that bots using the same DGA algorithm will generate similar NXDomain traffic. Pleiades is implemented based on a combination of clustering and classification algorithms. At first, Pleiades clusters domains based on the similarity in the lexical features of domain names and the groups of machines that queried these domains. Then Pleiades uses the classification algorithm to assign the generated clusters to models of known DGAs. A new model indicating a new DGA family will be produced if a cluster cannot be assigned to a known model.

Schiavoni *et al.* [5] propose Phoenix, a mechanism to detect and classify AGDs. Phoenix first proceeds a binary classification to identify AGDs and real domains based on two linguistic features, namely, the meaningful character ratio and the N-gram normality score. These features are able to measure the meaning and pronounceability of domain names. Then they calculate the probabilistic distribution for legitimate domains based on those linguistic features, and cluster suspicious domain names with known AGDs to classify them as belonging to specific malware families.

Although previous work mentioned above can obtain good effects in terms of detection precision or accuracy, they rely strongly on string-based and host-based features of domain names. These features are time-consuming to construct and extract, and host-based features, such as IP addresses, may change over time. Thus, to build a simple AGD detection approach based on string-based information only, Yadav *et al.* [3] leverage three metrics of distance to detect AGDs, including Kullback-Leibler (KL) distance, Edit distance (ED) and Jaccard Index (JI). These metrics can be easily calculated based solely on sets of domain name strings, and researchers do not need to spend a lot of time on feature engineering for domain names. Although detection method based on these metrics can achieve high accuracy for some DGA families, they have two essential shortcomings. On the one hand, approaches based on KL and JI only leverage the character distributions of domain names. Therefore, the can hardly work on AGDs that have different character patterns but similar distribution as real domain names. On the other hand, approaches based on ED can only detect domain names generated with regular grammar.

3 Proposed Approach

As discussed above, some existing AGD detection approaches are not efficient enough since they spend a lot of time and resources on constructing string-based and host-based features. Thus, in this paper, we propose a new AGD detection system called Nemesis, which does not rely on human-engineered features. Specifically, Nemesis is implemented based upon an LSTM language model, and it only

leverages the string-based information of domain names. In this section, we first introduce the architecture of Nemesis, then describe the implementation of each component in detail.

3.1 Architecture of Nemesis

The ultimate purpose of Nemesis is to identify whether unlabeled domain names are real ones or AGDs. Nemesis takes unlabeled domain names as the input, then it outputs the corresponding class label of each unlabeled domain name. To this end, Nemesis first tokenizes each domain name into n-grams according to a predefined n-gram dictionary, then trains an LSTM neural network based on the tokenized results, and uses this network to identify unlabeled domain names at last. As shown in Fig. 1, Nemesis generally consists of three modules: the Modeling module, the Training module, and the Detecting module.

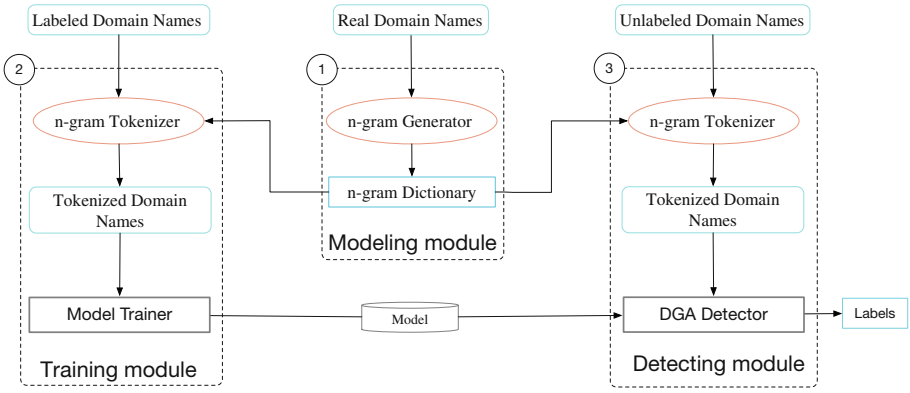

Fig. 1. The architecture of Nemesis.

Modeling Module. The purpose of the Modeling module is to build an n-gram dictionary which includes common n-grams of real domain names. The input to this module is a set of known legitimate domain names, and the output is an n-gram dictionary. This dictionary will be used by the n-gram Tokenizer in the Training module and the Detecting module. The only component of this module is the n-gram Generator, which will be detailly described in Sect. 3.2.

Training Module. The input to the Training module is a set of labeled domain names consisting of known AGDs and legitimate domain names, and the output is a trained LSTM network model. There are two important components in this module: the n-gram Tokenizer and the Model Trainer. We use the n-gram Tokenizer, which is implemented based on the n-gram dictionary obtained in the Modeling module, to tokenize domain names into n-grams, and then employ the

Model Trainer to train the LSTM network. The trained model can automatically distinguish AGDs in unlabeled domain names and will be used as the DGA Detector in the Detecting module.

Detecting Module. The input to the Detecting module is a set of unlabeled domain names, and the output is the corresponding class label of each domain name. This module contains two major functional components, the n-gram Tokenizer and the DGA Detector. The former is the same as the n-gram Tokenizer in the Training module, and the latter is the LSTM model trained in the Training Module. They will be described in Sects. 3.3 and 3.4 respectively. We classify unlabeled domain names in two steps. First, we utilize the n-gram Tokenizer to tokenize unlabeled domain names into n-grams. Second, we leverage the DGA Detector to label these tokenized unlabeled domain names as real ones or AGDs.

3.2 *n*-gram Generator

For readability and pronounceability, real domain names are usually composed of syllables or acronyms. Since it is hard to collect all syllables and acronyms, we compromise to represent them with n-grams. Nemesis learns the rules that how syllables or acronyms form real domain names and identifies domain names that disobey the rules as algorithmically generated. The n-gram Generator is tasked with extracting n-grams from real domain names and build a dictionary containing the most common n-grams.

The input to the n-gram Generator is a set of real domain names, such as Alexa top 1 million, and the output is an n-gram dictionary providing for the n-gram Tokenizer. First, we count the occurrences of each n-gram in real domain names, with n = {1, 2, 3, 4}. For example, in domain name "google", 1-grams include {'g', 'o', 'l', 'e'}, and the corresponding occurrences are respectively {2, 2, 1, 1}. 2-grams include {'go', 'oo', 'og', 'gl', 'le'}, and the corresponding occurrences are respectively {1, 1, 1, 1, 1}. Similarly, 3-grams and 4-grams can be counted in this way. Next, we sort all these n-grams by their occurrence frequencies in ascending order and collect the top 5000 into a dictionary. By extracting n-grams from domain names, we are trying to acquire the most frequent combinations of characters (including lower-case letters, digits, and hyphens) in domain names.

For a fully qualified domain name (e.g. www.example.com), the rightmost segment (e.g. *com*) is the top-level domain (TLD), and *example.com* is the second-level domain name (2LD). Since legitimate TLDs come from a well-known list [11], it is unnecessary for DGAs to generate a TLD. Therefore, we refer to effective 2LD (e.g. *example*) as "domain name" in this paper.

3.3 *n*-gram Tokenizer

The n-gram Tokenizer splits domain names into n-grams according to the dictionary obtained from the n-gram Generator. The input to the n-gram Tokenizer is

unlabeled domain names, and the output is the corresponding tokenized domain names (i.e., an n-gram list) for each of the unlabeled domain names.

In this paper, we adopt the Bi-direction Maximum Matching (BMM) Method to tokenize domain names into n-grams. The BMM Method fits languages that have no space to delimit words or characters, such as Chinese and Japanese. The principle of the BMM Method is a combination of the Forward Maximum Matching Method and the Backward Maximum Matching Method. The Forward Maximum Matching Method aims to partition a sentence into words from left to right as long as possible. In contrast, the Backward Maximum Matching Method partitions a sentence into words from right to left as long as possible. For the same sentence, the BMM Method compares the results from the Forward Maximum Matching Method and the Backward Maximum Matching Method, and chooses the better one according to the following rules:

- If the segmentation results from those two methods are the same, choose either of the two.
- If the number of words in the segmentation results from those two methods are not equal, choose the result that has fewer words.
- If the number of words in the segmentation results from those two methods are equal but these two results are not the same, choose the result that has less individual characters.

As we know, a domain name is a contiguous string without spaces, which is similar to the writing convention of Chinese and Japanese. Thus, in the n-gram Tokenizer, we regard domain names and n-grams as sentences and words respectively, then we can apply the BMM Method to domain names. In practice, the n-gram dictionary produced by the n-gram Generator contains all individual characters (1-gram) that appear in real domain names. Therefore, it is unnecessary to worry about meeting an n-gram that does not exist in the dictionary.

3.4 DGA Detector

The purpose of the DGA Detector is to distinguish AGDs from real domain names. After getting tokenized domain names from the n-gram tokenizer, we input them to the DGA Detector and expect it to output the corresponding class label of each domain name. The DGA Detector is implemented mainly based on an LSTM neural network which is widely used in natural language processing. The architecture of the DGA Detector is shown in Fig. 2.

The Numeric Encoder is employed to convert the tokenized domain name (an n-gram list) to a numeric vector since the neural network can only deal with tensors but not strings. It can be simply accomplished by replacing each n-gram with its sequence number (range from 1 to 5,000) in the n-gram dictionary to form a numeric vector. Then we pad the vector with 0 to make all of them have the same length l. The parameter l is the maximum number of n-grams in a single tokenized domain name.

The embedding layer converts the l-length vector to a 2-D tensor in the shape of $l * d$. The tunable parameter d is the output dimension of this layer. We

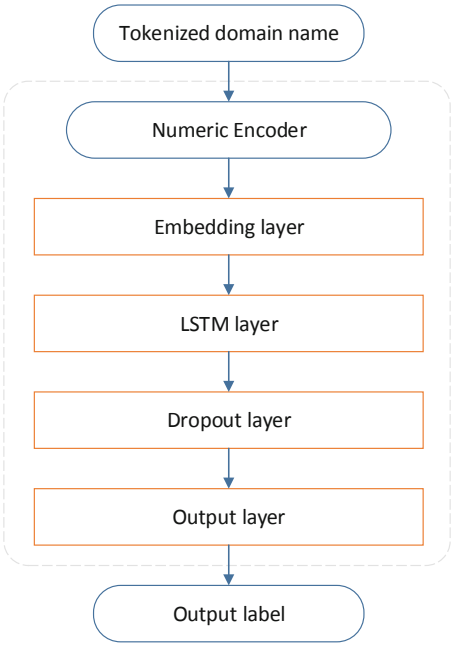

Fig. 2. The architecture of DGA detector

conduct several experiments with different parameter d and find the detection results are not very sensitive to it. Thus it is not important to choose the best parameter d and we set a proper value $d = 64$ to provide enough degrees of freedom to the model in subsequent experiments.

LSTM can capture meaningful temporal relationships among tokens in a sequence [12]. The LSTM cell consists of a state that can be read, written or reset via a set of programmable gates, thereby mitigating the vanishing gradients problem [13]. The LSTM layer can implicitly extract features of domain names and learn the contextual information of n-grams. This layer takes the output of the embedding layer as input, and a dropout layer is applied to the output to avoid overfitting when training the neural network.

The fully connected output layer is a simple logistic regression. It outputs the probability that the input domain name is algorithm generated. At last, the DGA Detector can label the domain name by comparing the probability with a proper threshold. We describe how to choose the optimal threshold in Sect. 5.1.

4 Experimental Evaluation

In this section, we design two experiments to evaluate the effectiveness of Nemesis. We first introduce the data sets used in our experiments, then define the evaluation metrics, and finally describe the experimental setup in detail.

4.1 Data Sets

To evaluate Nemesis, we collect 30 typical DGA families used by live botnets and select ten of them (i.e., Conficker, Banjori, Corebot, Cryptolocker, Dircrypt, Kraken_v1, Locky_v2, Pykspa, Qakbot, and Simda) that are more difficult to detect. We build two data sets for subsequent experiments:

- *malicious dataset*: We use each of these ten DGA families to generate 10k AGDs respectively, and regard all of these 100 k AGDs as the *malicious dataset*.
- *legitimate dataset*: We randomly sample 100 k domain names from Alexa top 1 million on 28th, Oct 2018 as *legitimate dataset*.

4.2 Evaluation Metrics

To evaluate the effectiveness of an AGD detection approach, we first make three definitions for further analysis:

- True Positives (TP): the number of domain names that are classified as AGDs by a detection approach and are indeed AGDs;
- False Positives (FP): the number of domain names that are classified as AGDs by a detection approach but are real ones in fact;
- False Negatives (FN): the number of domain names that are classified as real ones by a detection approach but are AGDs in fact.

Based on these definitions, we use three evaluation metrics, Precision (P), Recall (R) and the F1-Score (F1), to measure the effectiveness of a certain AGD detection approach. They can be calculated as follows:

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$R = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times P \times R}{P + R} \tag{3}$$

Precision and recall reflect the ability of the detection system in two aspects respectively. We expect the two values to reach the maximum. However, these two metrics are often contradictory, so we use F1-Score as a compromise between the two.

4.3 Experimental Setup

We conduct two different experiments, Experiment I and Experiment II, to evaluate Nemesis from two aspects.

Experiment I. The purpose of this experiment is to evaluate the ability of Nemesis to accurately identify domain names generated by known DGAs. Specifically, We use 90% of the domain names in *legitimate dataset* and *malicious dataset* as the training dataset, and the rest 10% as the test dataset. We use precision, recall, and F1-Scores to evaluate Nemesis in this experiment. At the same time, we compare Nemesis with previous work in this experiment.

Experiment II. The purpose of this experiment is to evaluate the ability of Nemesis to identify AGDs generated by new DGA families which does not appear in the training dataset. To this end, we apply the leave-one-out cross-validation (LOOCV) method at the level of DGA families. Specifically, for each DGA family, we collect domain names generated by this DGA as test dataset, and the training dataset consists of domain names from *legitimate dataset* and those generated by nine other DGA families. Since we only care about the proportion of domain names that are correctly identified, only recall is used to evaluate Nemesis in this experiment. Similarly, we also compare Nemesis with previous work in this experiment.

The results of Experiments I and II will be presented in Sect. 5.

5 Comparisons

To show the effectiveness of Nemesis, we compare it with a prior AGD detection approach in the process of Experiment I and II. In this section, we introduce the prior work at first, then presents the results of Experiment I and II.

5.1 Prior Work

As described in Sect. 2, Yadav *et al.* [3] use Kullback-Leibler (KL) Distance, Edit Distance (ED) and Jaccard Index (JI) to detect AGDs. Recently, Fu *et al.* [14] also apply these three forms of distance to detect real-life DGA families and achieve good results for five of them. This approach only requires the string-based information of domain names, which is similar to Nemesis, thus we compare it with Nemesis. The main idea of this approach is easy to understand. Given an unlabeled domain name d, the detection procedure can be divided into three steps: (1) Calculate the distance between d and a set of real domain names. (2) Calculate the distance between d and a set of AGDs. (3) Classify d as a real one or an AGD according to which distance is closer. Next, we will describe how to calculate these distances in detail.

Kullback-Leibler Distance. KL distance is a metric to measure the divergence between one probability distribution diverges and a reference probability distribution. In terms of a domain name, the probability distribution refers to the distribution of occurrence frequencies of all valid characters. For simplification, suppose that the list of valid characters in domain names is [a, b, c, d, e, f],

then the probability distribution of domain name "*faddec*" is [1/6, 0, 1/6, 1/3, 1/6, 1/6]. For discrete probability distributions P and Q defined on the same probability space, the KL distance between them can be calculated as follows:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P_i \log \frac{P_i}{Q_i} \tag{4}$$

where i is the index of the value in a discrete random variable. It is evident that KL distance between P and Q is not symmetric, so we can define a symmetric KL distance for uniformity:

$$D_{\mathrm{KL}_{\mathrm{sym}}}(PQ) = \frac{1}{2}(D_{\mathrm{KL}}(P\|Q) + D_{\mathrm{KL}}(Q\|P)) \tag{5}$$

Edit Distance. The Edit Distance [15] between two strings A and B are defined as the minimum times of edit operations required to convert A to B. Only three edit operations are allowed: insertion, deletion, and substitution of single characters. ED is symmetric and can be easily calculated by dynamic programming methods.

Jaccard Index. JI [16] is a metric to measure the similarity between two finite sets A and B. It is defined as:

$$JI(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{6}$$

In our experiments, we can refer the sets of characters in two domain names as set A and B respectively. Note that JI measures similarity while KL and ED measures dissimilarity, we can define JI Distance for uniformity:

$$d_{JI}(A,B) = 1 - JI(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}. \tag{7}$$

After figuring out the meaning of each metric, we now focus on how to detect AGDs with them. Given a test domain name d_t, we can use one of the three statistical distances to determine whether it is an AGD by following steps:

(1) Calculate $d(L)$ and $d(M)$ which denote respectively the average distance from d_t to each domain names in the *legitimate dataset* and the average distance from d_t to each domain names in the *malicious dataset*.
(2) Calculate Δd which denotes the difference between $d(L)$ and $d(M)$, namely $\Delta d = d(L) - d(M)$. The greater Δd, the more likely d_t is legitimate than malicious.
(3) Select a proper threshold t_{opt} for Δd and build a classifier. If Δd is greater than or equal to t_{opt}, the classifier will judge d_t as real. Otherwise, d_t is judged as an AGD.

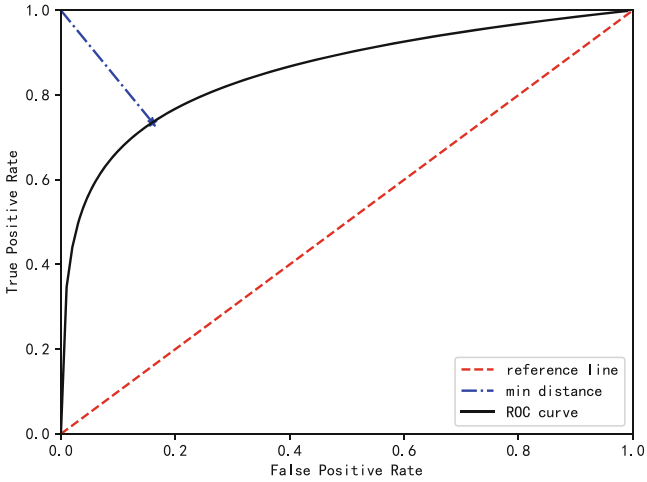

Fig. 3. An ROC curve demo

Then we leverage the Receiver Operating Characteristic (ROC) curve [17] to select the optimal threshold t_{opt} as follows: (1) Sample nine-tenth of the training dataset in Experiment I to train the classifier and use the rest one-tenth as the test dataset. (2) Select different t to calculate true positive rates (TPRs) and false positive rates (FPRs). (3) Draw the ROC curve based on the series of TPRs and FPRs, as shown in Fig. 3. (4) Find the closet point A on the ROC curve to the point (0,1), and the corresponding threshold is the optimal threshold t_{opt}.

5.2 Experimental Results

Experiment I. This experiment compares the ability of several approaches to accurately detect domain names generated by known DGA families. Figure 4 shows the results comparison among Nemesis and previous approaches based on KL, ED and JI. As illustrated in Fig. 4, Nemesis reaches the highest values of precision (98.6%), recall (96.7%) and the F1-Scores (97.6%), while those of the other approaches range from 72% to 83%. Obviously, the experiment result suggests that Nemesis has a considerable advantage in terms of detecting AGDs generated by known DGAs over approaches based on KL, ED and JI.

Experiment II. This experiment compares the ability of several approaches to discover new DGA families. Table 1 displays the recall values of Nemesis and approaches based on KL, ED and JI. In terms of certain DGAs, previous approaches are even inferior to an untrained binary classifier since they get recall values lower than 50%, though they may achieve better results than Nemesis on

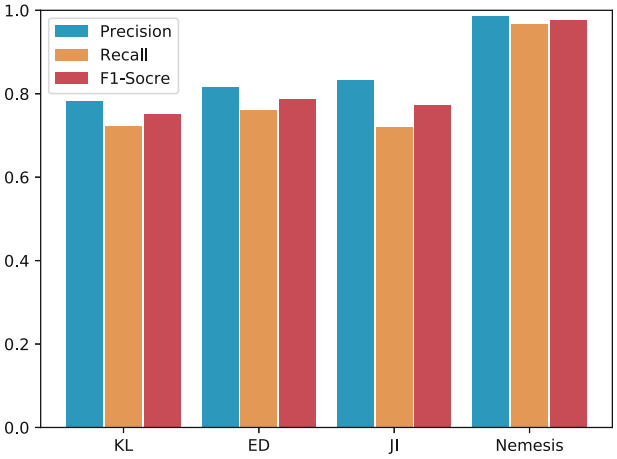

Fig. 4. Results of Experiment I

Table 1. Recall of four detection approaches on 10 DGA families in Experiment II

DGA name	Nemesis	KL	ED	JI
Conficker	0.833	0.879	0.402	0.902
Banjori	0.945	0.876	0.922	0.884
Corebot	0.975	0.572	0.743	0.473
Cryptolocker	0.903	0.821	0.835	0.794
Dircrypt	0.968	0.901	0.873	0.893
Kraken_v1	0.994	0.912	0.822	0.904
Locky_v2	0.983	0.886	0.929	0.813
Pykspa	0.985	0.473	0.895	0.402
Qakbot	0.942	0.899	0.793	0.845
Simda	0.834	0.858	0.729	0.784

other DGAs. In contrast, recall values of Nemesis are all greater than 83%, which means Nemesis has better stability in discovering different DGA families. In a word, Nemesis is stable and effective to discover unknown DGAs.

6 Conclusion

This paper represents a new AGD detection approach, Nemesis, which is implemented based on an LSTM language model. It takes unlabeled domain names as inputs, and outputs whether the domain names are AGDs or not. Nemesis employs theories and techniques in natural language processing and deep learning. It can learn from labeled domain names automatically and does not require any other additional information. We conduct two experiments on ten DGA

families to measure the abilities of Nemesis, i.e., the ability to detect AGDs of known DGAs and to discover new DGA families. The experiment results show that Nemesis outperforms the prior work in both respects.

Acknowledgments. This work is supported by the National Key Research and Development Program of China under Grant No. 2018YFB0804702 and No. 2018YFB0804704. The corresponding author is Tianning Zang.

References

1. Plohmann, D., Yakdan, K., Klatt, M., Bader, J., Gerhards-Padilla, E.: A comprehensive measurement study of domain generating malware. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, 10–12 August 2016, pp. 263–278 (2016)
2. Gai, K., Qiu, M., Tao, L., Zhu, Y.: Intrusion detection techniques for mobile cloud computing in heterogeneous 5G. Secur. Commun. Netw. **9**(16), 3049–3058 (2016)
3. Yadav, S., Reddy, A.K.K., Reddy, A.L.N., Ranjan, S.: Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. IEEE/ACM Trans. Netw. **20**(5), 1663–1677 (2012)
4. Antonakakis, M., et al.: From throw-away traffic to bots: detecting the rise of DGA-based malware. In: Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, 8–10 August 2012, pp. 491–506 (2012)
5. Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: DGA-based botnet tracking and intelligence. In: Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, 10–11 July 2014, Proceedings, pp. 192–211 (2014)
6. Sharifnya, R., Abadi, M.: Dfbotkiller: domain-flux botnet detection based on the history of group activities and failures in DNS traffic. Digit. Invest. **12**(12), 15–26 (2015)
7. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting domain generation algorithms with long short-term memory networks. CoRR, vol. abs/1611.00791 (2016)
8. Huang, J., Wang, P., Zang, T., Qiang, Q., Wang, Y., Yu, M.: Detecting domain generation algorithms with convolutional neural language models. In: TrustCom/BigDataSE, pp. 1360–1367 (2018)
9. Yu, B., Pan, J., Hu, J., Nascimento, A.C.A., Cock, M.D.: Character level based detection of DGA domain names. In: 2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, 8–13 July 2018, pp. 1–8 (2018)
10. Sood, A.K., Zeadally, S.: A taxonomy of domain-generation algorithms. IEEE Secur. Priv. **14**(4), 46–53 (2016)
11. Root zone database. https://www.iana.org/domains/root/db
12. Bengio, Y., Boulanger-Lewandowski, N., Pascanu, R.: Advances in optimizing recurrent networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, 26–31 May 2013, pp. 8624–8628 (2013)
13. Gers, F.A., Schmidhuber, J., Cummins, F.A.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
14. Fu, Y., et al.: Stealthy domain generation algorithms. IEEE Trans. Inf. Forensics Secur. **12**(6), 1430–1443 (2017)

15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. **10**(8), 707–710 (1966)
16. Small, H.: Co-citation in the scientific literature: a new measure of the relationship between two documents. J. Am. Soc. Inf. Sci. **24**(4), 265–269 (1973)
17. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)