

Enhancing DGA Detection with Machine Learning Algorithms

Hubert Biros and Mirosław Kantor

AGH University of Krakow, Kraków, Poland

<https://doi.org/10.26636/jtit.2025.FITCE2024.2033>

Abstract — The domain generation algorithm (DGA) is a popular technique used by malware to reliably establish a connection to a command and control (C&C) server. Pseudo-random domain names generated by DGA are used to bypass security measures and allow attackers to maintain control over malware-infected devices. In this work, we present a two-pronged approach to detecting character-based and word-based DGA domain names, creating classifiers specifically tailored to each type. For character-based DGA detection, we employed seven traditional machine learning methods: support vector machine, extremely randomized trees, logistic regression, Gaussian naive Bayes, nearest centroid, random forests, and k-nearest neighbors. We applied a featureful approach, using features extracted from the domain names themselves. Some of these features were drawn from existing literature, while others were newly proposed by authors. Feature selection techniques were used to retain only the best-performing ones. For the more complex task of detecting word-based DGA domain names, we used CNN and LSTM models, relying solely on word embeddings derived from the domain name components. Performance evaluation shows that proposed method gives high-performing, specialized DGA classifiers, which can be combined to create a more general-purpose classifier.

Keywords — character-based DGA, cybersecurity, DGA detection, DNS, machine learning-based DGA detection, malware, word-based DGA

1. Introduction

The domain name system (DNS) is a critical part of Internet infrastructure, translating human-readable domain names into machine-readable IP addresses. As the Internet evolves, securing the DNS against emerging threats becomes increasingly challenging. One common threat is the abuse of DNS through domain generation algorithms (DGAs), which malware uses to bypass security measures.

Devices infected by the malware, such as botnets or ransomware, need a reliable way to establish a connection with the command and control server (C&C) [1]–[3]. C&C plays a key role in operating malware-infected devices, allowing attackers to control victim machines and extract from them sensitive and valuable data [1]. Infected devices need a way to get the address of their C&C servers. Hard-coding the IP addresses or the domain names of these in the malware source code, is not a good solution, since once those are found by some security intelligence, blacklists can be created to shut down the operation of the malware [4], [5]. Instead, some

technique must be used by malware creators in order to easily relocate the C&C server to a different location in case of take-down of the working C&C server [3].

DGA is a popular technique used to establish a communication channel between infected devices and C&C servers [5]. For instance, many of the top 10 most popular financial malware families in the year 2023 were employed with some variant of DGA [6]. DGA is basically a piece of code that generates a large number of pseudo-random domain names that infected devices try to resolve to the address of the C&C server. In order to generate different sets of domains every time period, DGA typically uses some kind of seed in the form of a numerical hard-coded value or some time-dependent number [7], [8]. The key idea behind DGA is that malware operators having the same DGA algorithm and seed can register some of the generated domains and allow infected machines to connect with C&C server [8].

The process of using DGA is illustrated in Fig. 1, where the hacker or person who is put in charge of the infected devices uses the DGA algorithm implemented in the malware along with the particular seed to generate a set of domain names from which he selects one to register in the global DNS. In this case, the domain name is knosszts.ru. The hacker knows that in the future the infected device will use the same seed and thus generate the same set of domain names. The device will try to resolve all of them until one of them is correctly resolved and points to the C&C server.

This paper provides new insights into the detection of DGA domain names. Specifically, we introduce four new features that enhance the detection of character-based DGAs. Feature selection techniques revealed that some features used in previous works may be unnecessary and can be omitted without impacting performance. Additionally, our approach of using word embeddings for detecting word-based DGAs has shown very promising results. The strong performance of these DGA-focused classifiers suggests that they could serve as a solid foundation for future research and improvements in this area.

This paper is organized as follows. Section 2 introduces readers to the different types of DGAs and presents the approaches used to detect the domains generated by these algorithms. This section will additionally provide an overview of the related papers in the field and present the state-of-the-art solutions that were used for comparison with the results of our work. Section 3 provides a brief introduction to the

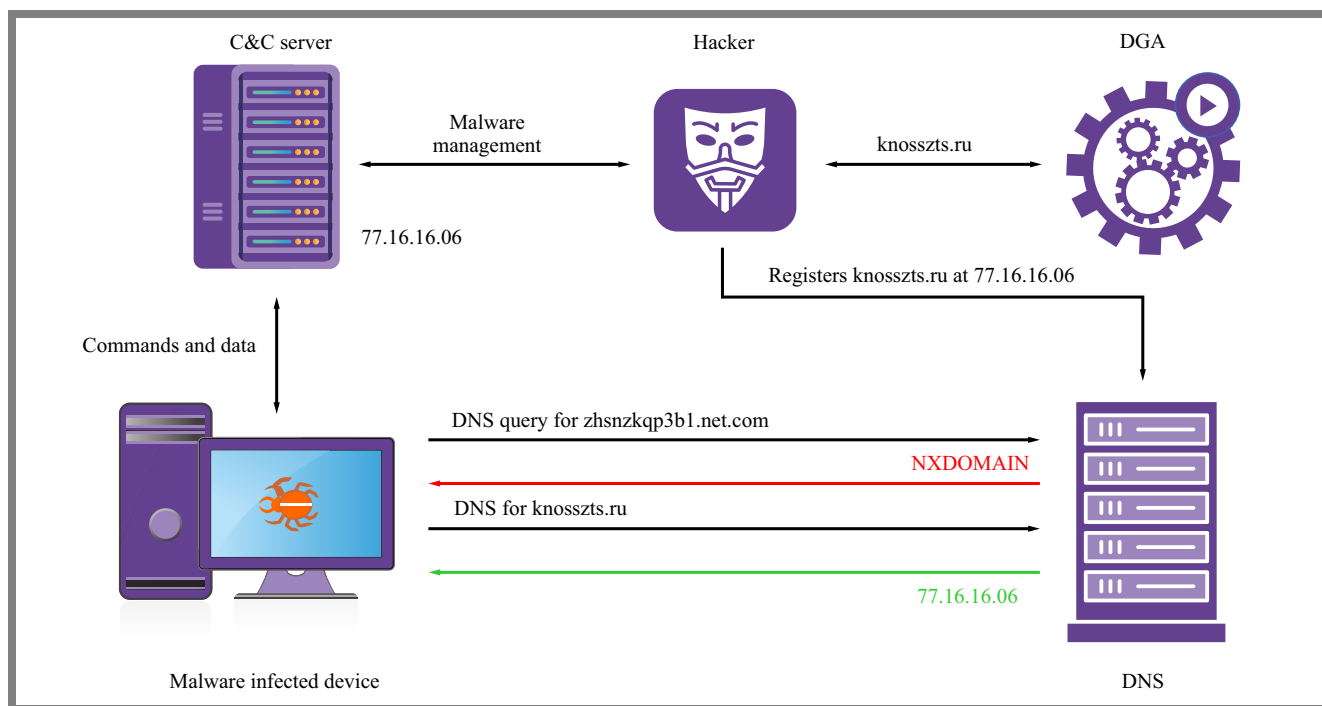


Fig. 1. Malware-infected device using DGA to generate domain names and contact C&C server.

machine learning methods that were used to create the DGA classifiers. Sections 4 and 5 provide a detailed description of the created models for character-based and word-based DGA detection, respectively. The evaluation results of the obtained DGA classifiers can be found in Section 6. The conclusions are presented in Section 7.

2. Related Works

This section provides a comprehensive analysis of domain generation algorithms. It offers insights into the intricacies of DGAs, their mechanisms, and the challenges of detecting them. Following subsections offers an overview of existing research in DGA detection along with the various approaches used in the field and briefly describes other works that were used for comparison with proposed models. A full comparison of the evaluation metrics can be found in Tab. 6.

2.1. Overview of DGA Types

We can distinguish three main categories of DGAs: character-based DGAs, word-based DGAs (sometimes called dictionary DGAs), and mixed DGAs, which combine elements of the first two types [8], [9]. Though another classification scheme, as outlined in [10], identifies four types: arithmetic-based DGAs, hash-based DGAs, wordlist-based DGAs, and permutation-based DGAs, we will adhere to the former classification. This is because it groups domain names based on how they appear to a human observer, which aligns better with the focus of our study.

The character-based DGA domains are constructed, by concatenating random characters into strings and then adding top level domain (TLD) to form the domain name. *Conficker*,

Necurs, or *Cryptolocker* are examples of malware that use this type of DGA. Some character-based DGAs are slightly more sophisticated, and in the process of creating the domain names, they distinguish between vowels and consonants to make generated domains more pronounceable. Examples of malware that uses this type of DGA are *Pitou* and *Symmi*. Word-based DGA are more dangerous in the sense they are more difficult to detect even by humans since they use a pre-defined list of words in the process of generating the domain names.

Word-based DGA can distinguish between different parts of speech such as adjectives, nouns, or verbs in order to create even more benign-looking domains. Examples of malware using word-based DGA are *Matsnu*, *Rovnix*, or *Suppobox*.

The last category is mixed DGA which is a combination of the two previous categories. Domains generated by mixed DGA have one part of the domain name generated by character-based DGA and another part is some combination of words. An example of malware that uses this category is *Banjori*. Table 1 contains examples of domains generated by different DGA-based malware families.

2.2. DGA Detection Approaches

The methods of detecting DGA domains can be divided into 2 categories: inline and retrospective [8], [11], [12]. In the retrospective method, collected DNS traffic can be analyzed in order to find DGA domains. This approach is a typical example of the intrusion detection system (IDS), since it works on past DNS data and cannot be used to stop malware from its operation. The latter inline approach can detect DGA domains as soon as the DNS query is made.

Tab. 1. Samples of different DGA domain names (C – character-based DGA, W – word-based DGA, M – mixed DGA).

Malware family	DGA type	Sample domain name
Banjori	W	uotvestnessbiophysicalohax.com
Cryptolocker	C	xpbfsnbuabrne.co.uk
Pitou	C	xoaomasat.us
Matsnu	W	mirrorhusbandboxconflict.com
Suppobox	W	weatheranother.net

The methods of detecting DGA domains can be further divided based on the information they need to classify the domain as DGA or non-DGA. The simplest approach is to use the information contained in the domain name string itself. While this method can effectively create a DGA classifier for character-based domains, such as “x1df6f33a99a10f9c7fdc5d176cd405ed7.so” generated by *Dyre* malware, word-based DGA domains, like “emilsmusic.com” generated by *Emotet*, may often appear more benign and less indicative of artificial origin. Although they can still be detected using their domain name, incorporating additional side information as parameters from DNS traffic (e.g., TTL, IP addresses) or information from the WHOIS database can enhance classification accuracy [12], [13].

The state-of-the-art solutions for detecting DGA domain names are based on machine learning methods to create DGA classifiers. Machine learning-based classifiers can be created in two ways. One way is the featureful approach in which we create a set of features from a domain name or other side information and use it to design the detection model. Examples of works that follow this direction include: [4], [9], [14], [15].

Another method used by [16]–[18] is the featureless approach in which we use some deep learning techniques like neural, convolutional, or LSTM networks to create classifiers. In this approach, the DGA classifier works on the information directly contained in the domain name and no features need to be constructed based on it. Some works like [8], [13] present a combined approach in which the classifier is trained using both featureless and featureful manner.

In detecting DGA domains, many works use n-gram analysis, e.g., [4], [14], [19]–[21]. N-grams are sequences of n consecutive characters in a string. For example, we can extract the following list of 2-grams from the label “google”: [“go”, “oo”, “og”, “gl”, “le”]. There are methods for detecting DGA domains only by using statistics of n-gram distribution like [21], which presents a model for detecting DGA domains based on a reputation score calculated by segmenting the domain names into n-grams and then calculating a weight value for each resulting n-gram based on the occurrence of the corresponding n-gram in non-DGA domains.

2.3. Works Used for Comparison with Proposed Models

Many DGA detection proposals focus on proposing a universal classifier capable of detecting both character-based

and word-based DGA domains. However, as highlighted in the findings of [4], some such models only work well in detecting only a specific type of DGA. We reviewed several state-of-the-art models in the literature that are either strictly designed to detect one type of DGA or have been trained using only one type of DGA.

The performance metrics of these models, as reported in the cited papers, should not be directly compared with our models since they were evaluated using different datasets, consisting of varying benign DNS traffic and distinct sets of DGAs. However, we include these in Tab. 4 to provide a broader perspective on the landscape of DGA detection models. Table 4 also shows some other general approaches to detecting DGA domains for a more comprehensive overview. Below is a brief description of these state-of-the-art solutions. Paper [19] proposed several DGA detection models, the best of which was the random forest classifier achieving an accuracy of 90.80%. The authors used a dataset of 30 000 benign domains and 30 000 character-based DGA domains that are used by four malware families: *Cryptolocker*, *Goz*, *Newgoz*, and *Conficker*, which are all character-based DGAs. Of the character-based DGA detection models we proposed, only two (Gaussian naive Bayes and nearest centroid) proved inferior. The same algorithm used in this work achieves an accuracy of 97.03% while maintaining a lower false positive rate (FPR).

The authors of [14] proposed a random forest classifier that uses 24 classification features extracted from each domain name. The model was trained on a dataset of 200 000 and tested on a set of 53 200 DGA domains of 39 different malware families. The model achieved a good result with 97.03% accuracy but performed poorly in detecting word-based DGA domains. In addition, the classifier was unable to correctly classify any *Banjori* botnet domains that implement mixed DGA. In our case, the random forest model with an extended set of 25 features designed to detect character-based DGA was able to detect 98.88% of *Banjori* DGA domains.

Article [21] did not use machine learning techniques to build a DGA classifier, but instead n-gram analysis of the domain names was proposed. It used 8 000 benign and 2 265 DGA domains belonging to various unspecified DGAs to evaluate the model.

The authors of [22] used only character-based and mixed DGA-generated domains, so it can be directly compared with the seven DGA classifiers proposed in our work. The proposed model is based on the LSTM network and uses a rather large dataset with a total of 1 675 404 domains, 10% of which serves as a test dataset. The authors of the referenced study reported only precision (PPV), true positive rate (TPR), F1-score, and accuracy in their results. Three of our seven character-based DGA domain detection models (KNN, RF, and ET) outperform it, although KNN achieves a slightly lower TPR (94.98% vs. 95.14%).

The paper [23] focused on the detection of word-based DGA domains therefore can be directly compared to the CNN and LSTM models we presented. Domains produced by DGA *Matsnu* and *Suppobox* were used in the evaluation of the

proposed models. The best classifier turned out to be random forest, which is outperformed by our two proposed models for word-based DGA detection in terms of ACC value.

Several classifiers for detecting word-based DGA domains was proposed in [9]. The best obtained classifier was the J-48 decision tree and it proved to be slightly better than our LSTM network model, but the CNN network model outperforms it in all available evaluation metrics.

In [15] DGA detection models based on Kullback-Leibler divergence and Jaccard index-based metrics are presented. The best multilayer perceptron (MLP) classifier was tested using 25 different DGAs, including character-based DGAs and word-based DGAs.

In [18] an LSTM model is proposed using both character-based and word-based DGA domain names in the dataset, but with a much smaller share of the latter.

The authors of [24] present a neural network model using BiLSTM and CNN layers with an attention mechanism (ATT-CNN-BiLSTM). The dataset contains 24 different DGA domain names, both character-based and word-based.

Article [17] proposed another approach to detect only the domain names generated by word-based DGAs. The authors present an ensemble learning-based model using both LSTM and CNN networks. The dataset consisted of domain names generated by *Supobox*, *Gozi*, and *Matsnu* malware. Similar to the paper [9], our proposed LSTM solution is slightly inferior to this ensemble model, but the CNN model outperforms it.

[25] presented a deep neural network (DNN) model for creating a DGA classifier based on features extracted from the domain names and DNS traffic. The paper used 5 different character-based DGAs in the dataset, allowing us to compare the performance of the proposed DNN classifier with our models for the detection of character-based DGA domain names. The DNN model performed slightly better in terms of ACC than the best model proposed in our work, which is the random forest classifier. However, it should be noted that the dataset used in our work to train and evaluate the models contains DGA domains used by 56 malware families.

The paper [11] proposed two DGA classifiers based on LSTM networks: binary classifiers, such as those presented in our work, and multi-class classifiers that allow a domain to be assigned to the specific DGA that generated it. The dataset included both word-based and character-based DGA domains.

3. Insights into the Machine Learning Models Used

This section provides a concise but informative overview of the machine learning algorithms used to detect DGA domains. Adapting our approach to the nuances of character-based and word-based DGA domains, we use classical methods such as logistic regression, Gaussian naive Bayes, support vector machine, random forest, extremely randomized trees, k-nearest neighbors, and nearest centroid for character-based DGA domains detection. In addition, long short-term memory (LSTM) and convolutional neural network (CNN) are

specifically used for the complex task of word-based DGA detection.

3.1. Logistic Regression

Logistic regression is a type of regression algorithm tailored to be used in classification tasks. This algorithm can be used to compute the probability that an instance represented by the set of features belongs to a particular class. In the training process, we aim to optimize values of vector θ , which is a vector of weights plus the bias term, so that training instances that represent DGA domains are assigned class 1, and instances representing benign domains are assigned class 0. During the training, regularization terms like l_1 or l_2 can be employed, to prevent the model from overfitting [26].

3.2. Gaussian Naive Bayes

Gaussian naive Bayes is another example of a supervised machine learning algorithm that can be used for DGA detection and it is based on the Bayes theorem [27]. It is a model that predicts a class of instances based on conditional probabilities. During the training phase, the model builds probability distributions of values of features from training instances for two classes of domains (DGA and non-DGA). The likelihood of the features is assumed to be Gaussian [28]. After training, the model assigns a new instance x to either class calculating posterior probabilities that x belongs to DGA and non-DGA domains. The term “naive” comes from the fact that the model assumes feature-wise conditional independence given the class variable [28].

3.3. Support Vector Machine

Support vector machine (SVM) is an approach used for classification tasks and is frequently regarded as one of the best classifiers that do not require extensive customization [29]. SVM aims to construct a hyperplane of dimension $n - 1$ given that, each instance in our dataset has n features in order to separate instances belonging to different classes. The classification decision is made by looking at which side of the hyperplane the instance lies.

The SVM classifier is sometimes called a soft margin classifier because the hyperplane it constructs has margins - that is perpendicular distance from some of the training observations [29]. The term “soft” comes from the fact the separating hyperplane may not perfectly separate two classes i.e., some training instances can be on the wrong side of the margin, but the obtained separation can generalize better on new instances. In some cases, the data points of two classes may even not be separable. With SVM, a kernel function can be applied to transform feature space. For instance, with a polynomial kernel, we can transform feature space into a higher dimension, where the separation of classes can be done more easily [26].

3.4. Random Forest

Random forest is an example of an ensemble learning algorithm that combines predictions of multiple decision trees.

Decision tree on the other hand is a very simple yet very powerful algorithm that can be used for classification tasks [26]. To train the decision tree in Scikit-learn the classification and regression tree (CART) algorithm is used.

The training process of the decision tree begins by splitting the training dataset into two subsets using a single feature f_n , and some threshold t_{f_n} . The algorithm selects the pair (f_n, t_{f_n}) , that produces the lowest value of the cost function, which reflects the impurity of the resulting subsets. After the first split, the two nodes representing the two subsets of the training dataset are obtained. The splitting operation is then performed on these nodes and on the resulting nodes recursively.

The process stops if the split minimizing the cost function cannot be found or based on some criteria like the maximum depth of the tree or maximum number of leaves.

The random forest algorithm trains multiple instances of decision trees, introducing some randomness into the process of creating individual trees. The goal of this randomness is to obtain decision trees that produce as independent decisions as possible. One approach to introduce this randomness is to use different training datasets for each classifier or use random feature subset for splitting decisions. The predictions of each decision tree classifier are combined to give the final results. Such an approach often results in better accuracy than the best classifier in the ensemble [26].

3.5. Extremely Randomized Trees

Extremely randomized trees or extra trees are another example of the ensemble learning algorithm. This algorithm is basically a random forest algorithm that introduces more randomness into the process of building individual trees by using random thresholds for each feature rather, than searching for the threshold that best minimizes the cost function [26].

3.6. K-Nearest Neighbors

Classification using the k-nearest neighbors algorithm is a type of instance-based learning. This model stores the training instances of the training data and performs classification on new instances based on a majority vote of the k-nearest neighbors of training instances. An instance is assigned a class, that is a majority among k-nearest neighbors [28].

3.7. Nearest Centroid

The nearest centroid classifier is a very simple algorithm, which assigns each class a mean (centroid) of their instances. The class of the new instance is assigned based on the centroid of which class is closer to the new observation [28].

3.8. Convolutional Neural Networks

In the last few years convolutional neural networks (CNNs) have managed to achieve very good performance on some complex visual tasks, such as image classification [26]. Though CNNs originated from the exploration of the visual cortex of the brain, they can be deployed in other tasks,

such as voice recognition or natural language processing (NLP) [26], [30].

CNNs use convolutional layers which consist of a set of filters also called kernels. The kernel can be treated as a matrix that is used in convolution operation with portions of the input sequence such as pixels of the image or an array of characters. The objective of this process is to extract patterns that are important for predictions. For example in the image processing task, the convolutional layer can extract some high-level features such as edges [31]. The values in the matrix representing the kernel, are learned during the training process.

3.9. Long Short-Term Memory Networks

Long short-term memory networks are a special type of recurrent neural networks (RNN) capable of learning long-term dependencies [32]. Traditional RNNs suffer from the problem of a vanishing and exploding gradient during backpropagation when dealing with more contextual data [32]–[34]. Long short-term memory networks, or simply LSTMs, use separate paths for long-term and short-term memory to avoid the vanishing and exploding gradient problem [33].

A single LSTM network module consists of three different gates that control the flow of information: forget gate, input gate, and output gate. LSTM networks are structured as chains of repeating modules. The output path from one module serves as the input for the corresponding path for the next module.

4. Proposed Models for Detecting Character-based DGA Domains

This section focuses on the use of classical machine learning methods for character-based DGA domains detection. We begin with a description of the dataset used. This is followed by a description of the process of constructing and selecting features to train the models. We conclude by describing the training process. The models were built using the Scikit-learn Python library.

4.1. Dataset

The dataset used consists of training and test subsets. Both contain non-DGA and DGA domains in a 1:1 ratio. A total of 450 000 benign domains (400 000 used for training and 50 000 for model evaluation) were derived from the top one million domain names ranked by Majestic [35]. As for the DGA domains, samples from 56 malware families were used. The DGA domains were obtained by executing reverse-engineered DGA code snippets available online or using predefined domain lists.

Table 2 shows the DGA datasets used to train and test the models. The source column serves as a reference as to where the domain names came from.

Tab. 2. Character-based and mixed DGA domains comprising the training and test dataset (C – character-based DGA, M – mixed DGA).

Malware family	Size of training dataset	Size of test dataset	Type	Source	Malware family	Size of training dataset	Size of test dataset	Type	Source
Orchard variant 3	9080	1135	C	[36]	Dyre	8860	1108	C	[39]
Vawtrak variant 1	9058	1132	C	[37]	Enviserv	8854	1107	C	[40]
Zeus Newgoz	9029	1129	C	[36]	Ranbyus variant 2	8850	1106	C	[37]
Qsnatch variant 1	9013	1127	C	[36]	Shiotob	8847	1106	C	[36]
Conficker	9007	1126	C	[38]	Chinad	8829	1104	C	[40]
Padcrypt v2.2.97.0	8993	1124	C	[36]	Cryptolocker	8825	1103	C	[38]
Ramdo	8989	1124	C	[37]	Murofet variant 3	8819	1102	C	[37]
Dircrypt	8989	1124	C	[36]	Vidro	8818	1102	C	[40]
Padcrypt v2.2.86.1	8987	1124	C	[36]	Pitou	8812	1101	C	[36]
Ramnit	8980	1122	C	[36]	Necurs	8772	1096	C	[36]
Qsnatch variant 2	8979	1122	C	[36]	Sison	8544	1068	C	[36]
Tinba	8979	1122	C	[37]	Pykspa	8516	1065	C	[37]
Kraken variant 2	8978	1122	C	[36]	Banjori	6812	851	M	[36]
Fobber variant 2	8976	1122	C	[37]	Torpig	6157	770	C	[41]
Murofet variant 2	8969	1121	C	[37]	Mydoom	5592	699	C	[37]
Locky variant 3	8968	1121	C	[37]	Simda	5180	647	C	[36]
Kraken variant 1	8961	1120	C	[36]	Zloader	3139	392	C	[36]
Proslikefan	8954	1119	C	[36]	Tempedreve	2823	353	C	[37]
Locky variant 2	8931	1117	C	[37]	Sharkbot v2.8	2582	323	C	[36]
Symmi	8928	1116	C	[37]	Zeus	889	111	C	[38]
Pushdo	8925	1116	C	[37]	Sharkbot v1.63	348	44	C	[36]
Ranbyus variant 1	8922	1115	C	[37]	Sharkbot v0.0	317	40	C	[36]
Nymaim variant 1	8916	1115	C	[37]	Sharkbot v2.1	317	40	C	[36]
Qadars variant 3	8914	1114	C	[36]	Vawtrak variant 3	267	33	C	[36]
Verblecon	8896	1112	C	[37]	Vawtrak variant 2	267	33	C	[36]
Murofet variant 1	8881	1110	C	[37]	Ccleaner	149	19	C	[40]
Fobber variant 1	8876	1109	C	[37]	Alueron Dnschanger	4	1	C	[36]
Corebot	8867	1108	C	[36]	Total	400,000	50,000		
Qakbot	8866	1108	C	[37]					

4.2. Features

In the process of constructing the features, the approach chosen was to use the information contained in the domain name itself. Before extracting the features from the domain, each was stripped of its TLD, and the remaining labels were converted to lowercase and combined into a single string without dots. So, for example, the domain “gmail.google.com” would be converted to the string “gmailgoogle”.

The approach of not including TLDs can be found in many works, but as the paper [21] shows, the domain TLD also contains information that can be useful for DGA detection. Besides, some TLDs are often associated with malicious activities [8]. Therefore, each TLD domain has been encoded with a single number and included as such in the feature set. Below is a list of the initial 35 proposed features. Features 1–22 and 29–30 were previously used in [4], [14], [19], while feature 35 was proposed in [21]. Feature 28 is the domain length and it was used in [8] and [42]. In addition, features 23–27, 31 and 33–34 were proposed in this work.

After applying feature elimination techniques, some of the features were removed from the final set. Although it may seem unnecessary to include them in the list below, since they were eventually removed, we retain their mention to provide readers with a complete description of the classifier development process.

Here is the list of features numbered as follows, along with a brief description:

- 1) $\text{count_2gram}(d)$: A number of 2-grams of the domain name d , which are also found in the list of 500 most frequent 2-grams found in the 10 000 most popular non-DGA domains.
- 2) $\text{m_2gram}(d)$: The 2-gram frequency distribution of the domain name d . $f(i)$ is the total number of occurrences of 2-gram found in the 500 most common 2-grams found in the 10 000 most popular non-DGA domain names. $\text{Index}(i)$ is the rank of 2-gram among all total possible 2-grams found in the 10 000 most popular non-DGA domains. For example, if 2-gram “a0” is the second most popular 2-gram found in the 10 000 non-DGA domains it gets the rank of 2.

$$\text{count_2gram}(d) = \sum_{i=1}^{\text{count_2gram}(d)} f(i) \cdot \text{index}(i) .$$

- 3) $\text{s_2gram}(d)$: The 2-gram weight of the domain name d . $\text{vt}(i)$ is the rank of 2-gram among the 500 most common 2-grams found in the 10 000 most popular non-DGA domain names.

$$\frac{\sum_{i=1}^{\text{count_2gram}(d)} f(i) \cdot \text{vt}(i)}{\text{count_2gram}(d)} .$$

4) $ma_2gram(d)$: The average 2-gram frequency distribution of the domain name d . $len_2gram(d)$ is the total number of 2-grams in d .

$$\frac{m_2gram(d)}{len_2gram(d)} .$$

5) $sa_2gram(d)$: The average 2-gram weight distribution of the domain name d .

$$\frac{s_2gram(d)}{len_2gram(d)} .$$

6) $tan_2gram(d)$: The average number of popular 2-grams in the domain name d .

$$\frac{count_2gram(d)}{len_2gram(d)} .$$

7) $taf_2gram(d)$: The average frequency of popular 2-grams in the domain name d .

$$\frac{\sum_{i=1}^{count_2gram(d)} f(i)}{count_2gram(d)} .$$

8) $count_3gram(d)$: A number of 3-grams of the domain name d , which are also found in the list of 500 most frequent 3-grams found in the 10 000 most popular non-DGA domains.

9) $m_3gram(d)$: The 3-gram frequency distribution of the domain name d . $f(i)$ is the total number of occurrences of 3-gram found in the 500 most common 3-grams found in the 10 000 most popular non-DGA domain names. $index(i)$ is the rank of 3-gram among all total possible 3-grams found in the 10 000 most popular non-DGA domains. For example, if 3-gram “a0-” is the second most popular 3-gram found in 10 000 non-DGA domains it gets the rank of 2.

$$\sum_{i=1}^{count_3gram(d)} f(i) \cdot index(i) .$$

10) $s_3gram(d)$: The 3-gram weight of the domain name d . $vt(i)$ is the rank of 3-gram among the 500 most common 3-grams found in the 10 000 most popular non-DGA domain names.

$$\frac{\sum_{i=1}^{count_3gram(d)} f(i) \cdot vt(i)}{count_3gram(d)} .$$

11) $ma_3gram(d)$: The average of 3-gram frequency distribution of the domain name d . $len_3gram(d)$ is the total number of 3-grams in d .

$$\frac{m_3gram(d)}{len_3gram(d)} .$$

12) $sa_3gram(d)$: The average of 3-gram weight distribution of the domain name d .

$$\frac{s_3gram(d)}{len_3gram(d)} .$$

13) $tan_3gram(d)$: The average number of popular 3-grams in the domain name d .

$$\frac{count_3gram(d)}{len_3gram(d)} .$$

14) $taf_3gram(d)$: The average frequency of popular 3-grams in the domain name d .

$$\frac{\sum_{i=1}^{count_3gram(d)} f(i)}{count_3gram(d)} .$$

15) $tanv(d)$: The distribution of vowels in the domain name d . $countv(d)$ is the number of vowels found in the domain d . $len(d)$ is a total number of characters in d .

$$\frac{countnv(d)}{len(d)} .$$

16) $tanco(d)$: The distribution of consonants in the domain name d . $countco(d)$ is the number of consonants found in the domain d .

$$\frac{countco(d)}{len(d)} .$$

17) $tandi(d)$: The distribution of digits in the domain name d . $countdi(d)$ is the number of digits found in the domain d .

$$\frac{countdi(d)}{len(d)} .$$

18) $tansc(d)$: The distribution of special characters in the domain name d . $countsc(d)$ is the number of occurrences of the “-” character in d .

$$\frac{countsc(d)}{len(d)} .$$

19) $tanhe(d)$: The distribution of hexadecimal characters in the domain name d . $counthe(d)$ is the number of hexadecimal characters found in the domain d .

$$\frac{counthe(d)}{len(d)} .$$

20) $ent_char(d)$: Character entropy of the domain name d . $D(x)$ is the probability distribution of the character x in the domain name d .

$$- \sum_x D(x) \cdot \log(D(x)) .$$

21) $EOD(d)$: The expected value of the domain name d . $n(x)$ is the frequency of occurrence of character x in the domain name d , and $p(x)$ is the probability distribution of character x calculated based on the 10 000 most popular benign domains.

$$\frac{\sum_x n(x) \cdot p(x)}{\sum_x n(x)} .$$

22) $is_first_char_digit(d)$: 1 if the first character of the domain d is a digit, else 0.

23) $vcds_entropy(d)$: Only four categories of characters are considered in this entropy: digits, special character “-”, consonants, and vowels. $K(k)$ is the probability distribution of category k in the domain name.

$$- \sum_k^{v,c,d,s} K(k) \cdot \log(K(k)) .$$

- 24) `conditional_vcds_entropy(d)`: In this measurement, the domain name d is divided into 2-grams, which are denoted by a pair of categories k and l . The category can be a vowel, consonant, digit, or special character “-”. $K(k|l)$ is the probability distribution of a 2-gram in which the first character belongs to category k and the second to category l . $K(k, l)$ is the probability distribution of a 2-gram in which one character belongs to the k category and the other to the l category.

$$- \sum_k^{v,c,d,s} \sum_l^{v,c,d,s} K(k, l) \cdot \log \frac{1}{K(k|l)}.$$

- 25) `double_consonants(d)`: The number of occurrences of two consonants next to each other in the domain name d .
- 26) `double_vowels(d)`: The number of occurrences of two vowels next to each other in the domain name d .
- 27) `double_chars(d)`: The number of occurrences of two of the same characters next to each other in the domain name d .
- 28) `len(d)`: The total number of characters in the domain name d .
- 29) `entropy_2gram(d)`: The 2-gram entropy of the domain name d . $vt(i)$ is the rank of 2-gram among the 500 most common 2-grams found in the 10 000 most popular non-DGA domain names.

$$- \sum_{i=1}^{\text{count_2gram}(d)} \frac{vt(i)}{500} \cdot \log \frac{vt(i)}{500}.$$

- 30) `entropy_3gram(d)`: The 3-gram entropy of the domain name d . $vt(i)$ is the rank of 3-gram among the 500 most common 3-grams found in the 10 000 most popular non-DGA domain names.

$$- \sum_{i=1}^{\text{count_3gram}(d)} \frac{vt(i)}{500} \cdot \log \frac{vt(i)}{500}.$$

- 31) `vc_bigram_ratio(d)`: The ratio of the number of 2-grams that comprise a vowel-consonant or consonant-vowel pair $vc(d)$ to the number of 2-grams of the domain name d .

$$\frac{vc(d)}{\text{len_2gram}(d)}.$$

- 32) `tld`: Encoded top level domain.
- 33) `jsd_2gram(d)`: The 2-gram Jensen-Shannon divergence of the domain name d . $P(x)$ is the probability distribution of 2-grams in the domain name d . $Q(x)$ is the probability distribution of 2-grams in the 10 000 most popular domain names ranked by Majestic. $M(x)$ is a mixture distribution of P and Q .

$$\frac{1}{2} \sum_x P(x) \log \frac{P(x)}{M(x)} + \frac{1}{2} \sum_x Q(x) \log \frac{Q(x)}{M(x)}.$$

- 34) `jsd_3gram(d)`: The 3-gram Jensen-Shannon divergence of the domain name d . $P(x)$ is the probability distribution of 2-grams in the domain name d . $Q(x)$ is the probability distribution of 3-grams in the 10 000 most popular domain

names ranked by Majestic. $M(x)$ is a mixture distribution of P and Q .

$$\frac{1}{2} \sum_x P(x) \log \frac{P(x)}{M(x)} + \frac{1}{2} \sum_x Q(x) \log \frac{Q(x)}{M(x)}.$$

- 35) `ji(d)`: Jaccard’s index of the domain name d . $2\text{grams}(d)$ is the set of 2-grams that make up the domain name d . $DN_1, DN_2, \dots, DN_{10000}$ are the 10 000 most popular domains ranked by Majestic.

$$\sum_{i=1}^{10000} \frac{|2\text{grams}(d) \cap 2\text{grams}(DN_i)|}{|2\text{grams}(d) \cup 2\text{grams}(DN_i)|}.$$

4.3. Feature Selection

In order to remove irrelevant and redundant features, a combination of feature selection techniques was employed to reduce the number of features from an initial set of 35. Firstly, ANOVA (analysis of variance) was utilized, which is particularly useful when dealing with categorical target variables and continuous features [43]. This statistical test helps identify features that are dependent on the target variable (class 0 or 1). Subsequently, the random forest and extra trees algorithms were leveraged to assess feature importance. By ranking features based on their contribution to predictive accuracy, this technique aids in the identification of less impactful features that may be considered for removal [26].

The last technique used for feature selection was to use the Pearson correlation method to calculate correlations between each pair of features. This helped to uncover and eliminate redundant features, that convey the same type of information.

As a starting point for eliminating redundant features, we took a correlation coefficient value greater than 0.9 or less than -0.9. Thus, we removed features 9, 11, 29, 30, and 33. Then, taking into account the results of the ANOVA test and the feature importance produced by random forest and extra trees algorithms, we got rid of more features through the process of elimination. In this way, features 7, 22, 25, 26, and 27, were removed. In this way, we removed 10 features from the initial set of 35.

4.4. Training

In the training phase of our character-based DGA detection models, we used a technique known as 10-fold cross-validation to select the best model hyperparameters. Dividing our dataset into ten groups, or folds, the learning process is repeated ten times, with each fold serving once as a test set. This strategy provides a more comprehensive assessment of model performance across different subsets of the data [26], [28], [29].

After selecting the best-performing hyperparameters, the models were re-trained using the entire training dataset.

Tab. 3. Word-based and mixed DGA domains comprising the training and test dataset (W – word-based DGA, M – mixed DGA).

Malware family	Size of training dataset	Size of test dataset	Type	Source
Nymaim variant 2	66853	8356	W	[36]
Banjori	17246	2156	M	[36]
Suppobox	66844	8356	W	[37]
Gozi	47939	5992	W	[37]
Matsnu	66912	8364	W	[38]
Rovnix	66770	8346	W	[38]
Bigviktor	66780	8348	W	[40]
Emotet	656	82	W	[44]
Total	400,000	50,000		

5. Proposed Models for Detecting Word-based DGA Domains

This section delves into the topic of word-based DGA detection, extending our exploration beyond character-based DGA classifiers. Using advanced neural network architectures, particularly long short-term memory networks and convolutional neural networks, the methodology for learning classifiers will be described. Initially, we describe the dataset used to train and evaluate the classifiers. We then examine the training process and delve into the architectural nuances of the models used. The models were constructed using the Keras library with the TensorFlow backend in Python.

5.1. Dataset

The dataset used to train and evaluate the classifiers includes a total of 900 000 domain names. Within this dataset, 450 000 benign domain names were taken from the one million most popular domain names ranked by Majestic [35], with a subset

Listing 1: LSTM model code

```

model=Sequential(name='lstm_model')
model.add(Embedding(input_dim=56_000,
                    output_dim=128, input_length=64))
model.add(LSTM(units=128, unroll=True,
               return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=128, unroll=True))
model.add(Dropout(0.2))
model.add(Dense(units=128, activation='relu',
                 kernel_initializer='glorot_normal'))
model.add(Dropout(0.2))
model.add(Dense(units=64, activation='relu',
                 kernel_initializer='glorot_normal'))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='sigmoid',
                 kernel_initializer='glorot_normal'))
opt = Adam(learning_rate=0.005)
model.compile(loss='binary_crossentropy',
              optimizer=opt, metrics=['accuracy'])

```

of 50 000 domains reserved for the purpose of model evaluation. At the same time, 450 000 domains associated with 8 different malware families were collected to form the DGA domain set. As in the dataset used to create the character-based DGA classifier, the DGA domains were obtained by executing reverse-engineered DGA code snippets available online or using predefined domain lists. Notably, similar to the approach taken in creating character-based DGA classifiers, one mixed-type DGA was included in the dataset. Details of the DGA domains used are shown in Tab. 3.

5.2. Training

During the process of building the classifiers, different architectures were tested, and those that showed the best performance are presented in this paper. Code snippets showing the

Listing 2: CNN model code

```

model=Sequential(name='cnn_model')
model.add(Embedding(input_dim=56_000,
                    output_dim=128, input_length=64))
model.add(Conv1D(200, 4, padding='same',
                 activation='relu',
                 kernel_initializer='glorot_normal'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2, strides=2,
                       data_format='channels_first'))
model.add(Conv1D(100, 2, padding='same',
                 activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2,
                       data_format='channels_first'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='relu',
                 kernel_initializer='glorot_normal'))
model.add(Dropout(0.5))
model.add(Dense(10, 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'])

```

Tab. 4. Compilation of evaluation metrics for our proposed models and state-of-the-art solutions (C – character-based DGA, W – word-based DGA, G – general approach for detecting both word-based and character-based DGA domain names).

Classifier	Type	PPV	TPR	FPR	FNR	F1	ACC	AUC
RF [19]	C	90.7%	91%	9.3%		90.8%	90.8%	
RF [14]	G	97.08%	96.98%	2.92%	3.02%	97.03%	97.03%	
N-Gram [21]	G			6.14%	7.42%		94.04%	
LSTM [22]	C	95.05%	95.14%			94.58%	95.14%	
RF [23]	W						78.2%	
J48 [9]	W	98.25%	95.81%	1.78%	4.19%	97.01%	96.99%	
MLP [15]	G	99.5%	99.55%			99.5%	99.5%	
LSTM [18]	G	98.43%	98.4%			98.42%		
ATT-CNN-BiLSTM [24]	G	99.01%	99.07%			98.79%	98.82%	0.9990
CNN+LSTM [17]	W	95.57%	97.66%	4.54%		96.6%	96.56%	0.9944
DNN [25]	C	89.24%	99.14%				97.79%	0.9900
LSTM [11]	G	96.74%	85.71%			89.13%		0.9993
The proposed models in our work								
ET	C	96.84%	95.48%	3.12%	4.52%	96.16%	96.18%	0.9937
SVM	C	94.39%	93.08%	5.53%	6.92%	93.73%	93.77%	0.9846
LR	C	94.28%	93.18%	5.66%	6.82%	93.72%	93.76%	0.9847
GNB	C	83.28%	91.27%	18.33%	8.73%	87.09%	86.47%	0.9278
NC	C	85.36%	86.50%	14.83%	13.50%	85.93%	85.83%	0.9380
RF	C	97.60%	96.43%	2.37%	3.57%	97.01%	97.03%	0.9954
KNN	C	96.33%	94.98%	3.62%	5.02%	95.65%	96.00%	0.9901
LSTM	W	94.34%	96.50%	5.78%	3.50%	95.41%	95.36%	0.9905
CNN	W	97.84%	98.78%	2.19%	1.22%	98.31%	98.30%	0.9975

definitions of LSTM and CNN models in Python using the Keras library are shown in Listing 1 and 2, respectively.

The process of training the neural networks began with proper domain preparation, which involved converting them to lowercase, removing TLDs, and then breaking them down into lists of words that make up the domain name using the wordninja package in Python [45]. For example, the domain “watchfire.com” would be converted to a list [“watch”, “fire”]. Words to be input to neural networks must be uniquely encoded. During data processing, it was determined that 56 000 unique values could be used to encode all the words making up the training and test set. In fact, in the dataset, the number of unique words forming all domains was 55 027, but experiments showed that the size of the vocabulary does not affect the performance of the models, so for simplicity it was decided to set the number known in the Keras embedding layer as `input_dim`, which refers to the size of the vocabulary, to 56 000. Similarly, we set the maximum number of words a domain can contain to 64 (`input_length` parameter in the embedding layer).

Both CNN and LSTM models have an embedding layer that learns to map the corresponding values representing the words

that make up the domain to 128-dimensional vectors. A number of works, such as [11], [13], [16]–[18], [46], use deep learning models with embedding layers that rely on character embedding, which is different from the approach we used.

The LSTM model implements two long short-term memory layers, each consisting of 128 units. The first LSTM layer is set to return the full sequence of output data for each time step. By applying dropout layers at a rate of 20% after each LSTM layer, the model alleviates over-fitting during learning.

Following the LSTM layers are three dense layers, containing 128, 64, and 1 unit(s), respectively. The first two use the rectified linear unit (ReLU) activation function. These dense layers are interspersed with dropout layers to increase the robustness of the model. The last dense layer with a sigmoidal activation function makes the model act as a binary classifier.

The CNN model employs two Conv1D layers containing 200 and 100 filters, respectively. These convolution layers are activated using the ReLU function. Dropout layers with a 50% dropout rate follow each convolution layer to mitigate over-fitting. The convolution layers are followed by two Max-

Tab. 5. Accuracy for each character-based DGA detection model relating to different DGA families and benign domains.

Origin of domains	ET [%]	SVM [%]	LR [%]	GNB [%]	NC [%]	RF [%]	KNN [%]
Majestic	96.88	94.47	94.34	81.67	85.17	97.63	96.38
Orchard v3	99.47	97.00	97.27	100.00	99.56	99.65	99.65
Vawtrak v1	92.49	92.49	92.84	89.93	85.34	93.82	93.29
Zeus Newgoz	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Qsnatch v1	91.30	92.64	92.81	94.94	93.70	92.19	89.97
Conficker	82.15	82.33	82.24	92.63	88.10	84.81	79.31
Padcrypt v2.2.97.0	98.67	98.49	98.58	92.35	75.09	99.11	98.93
Ramdo	99.38	99.82	99.73	93.24	78.11	99.47	99.29
Dircrypt	97.86	97.78	97.69	96.26	92.70	98.04	96.89
Padcrypt v2.2.86.1	98.84	98.58	98.58	92.35	76.25	99.64	99.64
Kraken v2	94.56	92.16	92.25	94.39	91.00	95.37	92.78
Ramnit	99.55	98.66	98.75	98.04	94.39	99.82	98.66
Qsnatch v2	50.53	40.29	39.75	79.14	80.66	60.96	48.31
Fobber v2	96.17	95.01	95.37	94.74	90.29	96.79	94.56
Tinba	97.86	97.95	98.13	96.88	91.62	97.42	96.52
Murofet v2	99.91	99.82	99.73	98.39	95.90	99.82	99.38
Locky v3	96.79	95.36	95.81	95.63	92.15	96.97	95.09
Kraken v1	96.96	97.14	97.14	93.66	86.61	97.14	97.05
Prosilkefan	89.54	87.31	87.40	92.14	89.28	91.42	86.68
Locky v2	88.90	88.81	88.81	93.82	90.06	90.87	87.38
Symmi	95.97	97.49	97.49	75.36	49.28	98.12	97.31
Pushdo	83.96	71.68	72.67	55.20	42.47	98.57	90.50
Ranbyus v1	99.10	99.64	99.64	97.22	92.20	99.01	98.92
Nymaim v1	88.25	86.01	85.83	92.65	87.98	88.07	84.30
Qadars v3	99.64	99.73	99.73	99.46	95.78	99.64	99.37
Verblecon	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Murofet v1	99.91	100.00	100.00	99.91	99.55	99.91	99.91
Fobber v1	99.91	100.00	100.00	99.01	96.30	100.00	99.91
Dyre	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Qakbot	99.10	98.74	98.74	97.74	94.95	99.19	98.65
Corebot	100.00	100.00	100.00	99.91	100.00	100.00	100.00
Enviserv	99.82	99.91	99.91	100.00	99.55	99.91	99.73
Shiotob	99.19	99.82	99.82	99.64	98.01	99.19	98.19
Ranbyus v2	99.55	100.00	100.00	97.92	93.58	99.64	99.19
Chinad	100.00	100.00	100.00	99.91	99.73	99.91	99.73
Cryptolocker	99.09	99.37	99.27	96.46	91.30	99.09	98.01
Murofet v3	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Vidro	95.55	94.28	94.37	96.64	92.83	94.74	93.92
Pitou	96.91	61.22	63.03	46.23	30.52	98.27	98.27
Cecurs	97.35	96.44	96.53	95.62	91.24	97.35	96.81
Sison	100.00	100.00	100.00	100.00	98.60	100.00	100.00
Pykspa	89.58	83.38	83.47	75.68	69.95	91.17	86.67
Banjori	100.00	94.24	93.07	0.00	0.00	99.88	100.00
Torpig	98.44	93.77	94.29	93.12	85.19	98.83	97.01
Mydoom	93.13	88.27	88.84	77.68	67.24	94.85	93.85
Simda	94.44	55.64	55.49	93.82	92.43	93.97	94.44
Zloader	99.74	100.00	100.00	98.72	96.17	99.74	99.49
Tempedreve	90.08	87.82	88.95	90.08	86.69	90.93	89.24
Sharkbot v2.8	100.00	100.00	100.00	100.00	100.00	100.00	99.69
Zeus	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Sharkbot v1.63	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Sharkbot v2.1	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Sharkbot v0.0	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Vawtrak v3	18.18	21.21	24.24	36.36	30.30	15.15	3.03
Vawtrak v2	36.36	30.30	36.36	48.48	33.33	36.36	30.30
Ccleaner	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Alueron Dnschanger	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Pooling1D layers with pool sizes of 2 and steps of 2. The CNN model, like the LSTM, adds 3 dense layers. The first two contain 100 and 10 units respectively with ReLU activation functions and one dropout layer. The last dense layer with a sigmoidal activation function transforms the model into a binary classifier. Dense layers in both models and convolution layers in the CNN model are initialized with glorot normal weights.

The compilation of both models uses binary cross-entropy loss, Adam’s optimizer, and accuracy as evaluation factors. The LSTM model set the learning rate to 0.005, while the CNN model used the default value of 0.001. The LSTM model was trained for 5 epochs with a batch size of 128, while the CNN model underwent a longer learning period of 10 epochs under the same batch size conditions.

6. Evaluation of the Models

After the training process, we proceeded to evaluate models’ performance using a dedicated test dataset containing 100 000 domain names. To measure the performance of the models, we used seven key metrics: ACC (overall accuracy), PPV (positive predictive value) or precision, TPR (true positive rate) or recall, FPR (false positive rate), FNR (false negative rate), F1 score and AUC (area under the ROC curve). The formulas used to calculate these indicators are shown below:

$$PPV = \frac{TP}{TP + FP} \cdot 100\% , \quad (1)$$

$$TPR = \frac{TP}{TP + FN} \cdot 100\% , \quad (2)$$

$$FPR = \frac{FP}{TN + FP} \cdot 100\% , \quad (3)$$

$$FNR = \frac{FN}{TP + FN} \cdot 100\% , \quad (4)$$

$$F1 = \frac{2TP}{2TP + FP + FN} \cdot 100\% , \quad (5)$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \cdot 100\% , \quad (6)$$

$$AUC = \int_0^1 TPR(FPR) dFPR . \quad (7)$$

Where true positives (TP) is the number of DGA domains that were classified correctly, true negatives (TN) is the number of benign domains that were classified correctly, false positives (FP) is the number of benign domains that were misclassified as DGA, and false negatives (FN) is the number of DGA domains that were classified as benign domains.

The specific values of these performance indicators for each model for detecting both character-based and word-based DGA are summarized at the bottom of the Tab. 4.

Tab. 6. Accuracy for each word-based DGA detection model relating to different DGA families and benign domains.

Origin of domains	LSTM [%]	CNN [%]
Majestic	94.22	97.81
Matsnu	99.76	99.88
Nymaim v2	89.64	96.46
Suppobox	97.69	99.77
Bigviktor	98.22	99.82
Rovnix	99.84	99.96
Gozi	93.56	96.81
Banjori	100.00	100.00
Emotet	9.76	17.07

7. Conclusions

In the case of detecting domains generated by character-based DGAs, five of the seven classifiers proposed in this work achieved ACC greater than 90%. The random forest-based model was the best classifier, achieving an ACC of 97.03% with fairly low FPR and FNR. Random forest has already been successfully used in other works, such as [14], [19] and [23]. The worst performing classifiers were those based on the gaussian naive Bayes model and the nearest centroid model, achieving ACCs of 86.47% and 85.83%, respectively. Table 5 also shows that these two models were unable to detect any of the 851 domains belonging to the mixed DGA used by the *Banjori* malware.

All models performed equally poorly in detecting DGA domains belonging to *Vawtrak v2* and *Vawtrak v3* malware, achieving an accuracy of less than 50%. If we look at the domains generated by these two variants, we can see that they are quite pronounceable character-based DGA domains. Examples of domains belonging to *Vawtrak v2* include “alohgufda.com”, “usornatda.com”, or “fosornom.com”, examples of *Vawtrak v3* domains include “sumiwgecoll.com”, “aldemegnehi.com”, and “garidsemogn.com”.

As for the task of detecting word-based DGA domains, both proposed classifiers achieved ACC scores greater than 95%. The CNN layer-based model was the best, achieving an ACC of 98.30% with low FPR and FNR values of 2.19% and 1.22% respectively. Table 6 shows that the model additionally achieved accuracy close to 100% for all DGAs except one belonging to the *Emotet* malware. The test dataset included 82 domains belonging to this DGA, and both the LSTM and CNN models performed poorly in detecting them, failing to exceed an accuracy of 20%.

This result may have been influenced by the fact that many of the domains generated by the *Emotet* DGA, such as “www.69po.com”, “ceylonsri.com”, or “senteum.com”, are quite short in contrast to other DGAs and more closely resemble domains generated by character-based DGAs.

References

- [1] “Botnet Threat Update Q3 2023”, Spamhaus, [Online]. Available: <https://info.spamhaus.com/botnet-threat-updates>.
- [2] “What is a Botnet?”, Palo Alto Networks, [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/what-is-botnet>.
- [3] A. Randall *et al.*, “The Challenges of Blockchain-based Naming Systems for Malware Defenders”, *2022 APWG Symposium on Electronic Crime Research (eCrime)*, Boston, USA, 2022 (<https://doi.org/10.1109/eCrime57793.2022.10142131>).
- [4] X.H. Vu, X.D. Hoang, and T.H.H. Chu, “A Novel Model Based on Ensemble Learning for Detecting DGA Botnets”, *2022 14th International Conference on Knowledge and Systems Engineering (KSE)*, Nha Trang, Vietnam, 2022 (<https://doi.org/10.1109/KSE56063.2022.9953792>).
- [5] E. Durmaz, “DGA Classification and Detection for Automated Malware Analysis”, Cyber.WTF, 2017 [Online]. Available: <https://cyber.wtf/2017/08/30/dga-classification-and-detection-for-automated-malware-analysis>.
- [6] “Kaspersky Security Bulletin 2023”, Kaspersky, [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2023/11/28102415/KSB_statistics_2023_en.pdf.
- [7] L. Asher-Dotan, “What is Domain Generation Algorithm: 8 Real World DGA Variants”, Cybereason, [Online]. Available: <https://www.cybereason.com/blog/what-are-domain-generation-algorithms-dga>.
- [8] R. Sivaguru *et al.*, “Inline Detection of DGA Domains Using Side Information”, *IEEE Access*, vol. 8, pp. 141910–141922, 2020 (<https://doi.org/10.1109/access.2020.3013494>).
- [9] X.D. Hoang and X.H. Vu, “A Novel Machine Learning-based Approach for Detecting Word-based DGA Botnets”, *Journal of Theoretical and Applied Information Technology*, vol. 99, no. 24, 2021.
- [10] D. Plohmann *et al.*, “A comprehensive measurement study of domain generating malware”, *Proc. of 25th USENIX Security Symposium*, Austin, USA, pp. 263–278, 2016.
- [11] J. Woodbridge, H.S. Anderson, A. Ahuja, and D. Grant, “Predicting Domain Generation Algorithms with Long Short-Term Memory Networks”, *arXiv*, 2016 (<https://doi.org/10.48550/arXiv.1611.00791>).
- [12] M. Pereira *et al.*, “Dictionary Extraction and Detection of Algorithmically Generated Domain Names in Passive DNS Traffic”, *International Symposium on Research in Attacks, Intrusions, and Defenses*, Heraklion, Greece, 2018 (https://doi.org/10.1007/978-3-030-00470-5_14).
- [13] R.R. Curtin *et al.*, “Detecting DGA Domains with Recurrent Neural Networks and Side Information”, *Proc. of the 14th International Conference on Availability, Reliability and Security – ARES’19*, pp. 1–10, 2019 (<https://doi.org/10.1145/3339252.3339258>).
- [14] X.D. Hoang and X.H. Vu, “An Improved Model for Detecting DGA Botnets Using Random Forest Algorithm”, *Information Security Journal: A Global Perspective*, vol. 31, no. 4, pp. 441–450, 2021 (<https://doi.org/10.1080/19393555.2021.1934198>).
- [15] A. Cucchiarelli, C. Morbidoni, L. Spalazzi, and M. Baldi, “Algorithmically Generated Malicious Domain Names Detection Based on n-Grams Features”, *Expert Systems with Applications*, vol. 170, art. no. 114551, 2021 (<https://doi.org/10.1016/j.eswa.2020.114551>).
- [16] B. Yu *et al.*, “Inline DGA Detection with Deep Networks”, *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, New Orleans, USA, 2017 (<https://doi.org/10.1109/ICDMW.2017.96>).
- [17] K. Highnam, D. Puzio, S. Luo, and N.R. Jennings, “Real-time Detection of Dictionary DGA Network Traffic Using Deep Learning”, *SN Computer Science*, vol. 2, art. no. 110, 2021 (<https://doi.org/10.1007/s42979-021-00507-w>).
- [18] D. Tran *et al.*, “A LSTM Based Framework for Handling Multiclass Imbalance in DGA Botnet Detection”, *Neurocomputing*, vol. 275, pp. 2401–2413, 2018 (<https://doi.org/10.1016/j.neucom.2017.11.018>).
- [19] X.D. Hoang and Q.C. Nguyen, “Botnet Detection Based on Machine Learning Techniques Using DNS Query Data”, *Future Internet*, vol. 10, art. no. 43, 2018 (<https://doi.org/10.3390/fi10050043>).
- [20] S. Yadav, A.K.K. Reddy, A.L.N. Reddy, and S. Ranjan, “Detecting Algorithmically Generated Malicious Domain Names”, *Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement*, pp. 48–61, 2010 (<https://doi.org/10.1145/1879141.1879148>).
- [21] H. Zhao, Z. Chang, G. Bao, and X. Zeng, “Malicious Domain Names Detection Algorithm Based on N-Gram”, *Journal of Computer Networks and Communications*, pp. 1–9, 2019 (<https://doi.org/10.1155/2019/4612474>).
- [22] Y. Qiao *et al.*, “DGA Domain Name Classification Method Based on Long Short-term Memory with Attention Mechanism”, *Applied Sciences*, vol. 9, no. 20, art. no. 4205, 2019 (<https://doi.org/10.3390/app9204205>).
- [23] L. Yang *et al.*, “A Novel Detection Method for Word-based DGA”, *Lecture Notes in Computer Science*, vol. 11064, pp. 472–483, 2018 (https://doi.org/10.1007/978-3-030-00009-7_43).
- [24] F. Ren, Z. Jiang, X. Wang, and J. Liu, “A DGA Domain Names Detection Modeling Method Based on Integrating an Attention Mechanism and Deep Neural Network”, *Cybersecurity*, vol. 3, art. no. 4, 2020 (<https://doi.org/10.1186/s42400-020-00046-6>).
- [25] Y. Li, K. Xiong, T. Chin, and C. Hu, “A Machine Learning Framework for Domain Generation Algorithm (DGA)-based Malware Detection”, *IEEE Access*, vol. 7, pp. 32765–32782, 2019 (<https://doi.org/10.1109/access.2019.2891588>).
- [26] A. Géron, *Hands-on Machine Learning with Scikit-learn, Keras, and TensorFlow*, O’Reilly Media, Inc, 2nd ed., 848 p., 2019 (ISBN: 9781492032649).
- [27] A. Smola and S.V.N. Vishwanathan, *Introduction to Machine Learning*, Cambridge University Press: Cambridge, UK, 2008.
- [28] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011 (<https://dl.acm.org/doi/10.5555/1953048.2078195>).
- [29] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, New York, 440 p., 2013 (<https://doi.org/10.1007/978-1-4614-7138-7>).
- [30] K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”, *Biological Cybernetics*, vol. 36, pp. 193–202, 1980 (<https://doi.org/10.1007/BF00344251>).
- [31] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks the ELI5 way”, Saturn Cloud, 2018 [Online]. Available: <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way>.
- [32] C. Olah, “Understanding LSTM Networks”, Colah, 2015 [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [33] J. Starmer, “Long Short-Term Memory (LSTM), Clearly Explained”, StatQuest with Josh Starmer, 2022 [Online]. Available: <https://www.youtube.com/watch?v=YCzL96nL7j0>.
- [34] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory”, *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997 (<https://doi.org/10.1162/neco.1997.9.8.1735>).
- [35] “The Majestic Million”, Majestic, [Online]. Available: <https://majestic.com/reports/majestic-million>.
- [36] J. Bader, “Binary Reverse Engineering Blog”, [Online]. Available: <https://bin.re/blog>.
- [37] J. Bader, “Domain Generation Algorithms”, GitHub repository, [Online]. Available: https://github.com/baderj/domain_generation_algorithms.
- [38] A. Abakumov, “DGA”, GitHub repository, [Online]. Available: <https://github.com/andrewaeva/DGA>.
- [39] F. Denis, “Dyre/Dyreza DGA”, GitHub repository, [Online]. Available: <https://gist.github.com/jedisct1/33ab6b4e81209dbf53a3>.
- [40] “DGA”, GitHub repository, [Online]. Available: <https://github.com/360netlab/DGA>.
- [41] P. Chaignon, “DGA-collection”, GitHub repository, [Online]. Available: <https://github.com/pchaigno/dga-collection>.
- [42] T.D. Truong and G. Cheng, “Detecting Domain-flux Botnet Based on DNS Traffic Features in Managed Network”, *Security and Communi-*

cation Networks, vol. 9, pp. 2338–2347, 2016 (<https://doi.org/10.1002/sec.1495>).

- [43] J. Brownlee, “How to Perform Feature Selection with Numerical Input Data”, *Machine Learning Mastery*, [Online], Available: <https://machinelearningmastery.com/feature-selection-with-numerical-input-data/>.
 - [44] D. Takahashi, “Emotet Domain”, GitHub repository, [Online], Available: <https://github.com/HASH1da1/emotet-domain>.
 - [45] Wordninja 2.0.0., Python Package Index, [Online], Available: <https://pypi.org/project/wordninja/>.
 - [46] R. Sivaguru *et al.*, “An Evaluation of DGA Classifiers”, *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, USA, 2018 (<https://doi.org/10.1109/BigData.2018.8621875>).
-


Hubert Biros

Independent Researcher, Kraków, Poland

E-mail: hubertbiros00@gmail.com

Mirosław Kantor, Ph.D.

Institute of Telecommunications

 <https://orcid.org/0000-0002-3160-6422>

E-mail: miroslaw.kantor@agh.edu.pl

AGH University of Krakow, Kraków, Poland

<https://www.agh.edu.pl>