Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier XXX

Towards Optimal LSTM Neural Networks for Detecting Algorithmically Generated Domain Names

Jose Selvi[1], Ricardo J. Rodríguez[2], (Member, IEEE), and Emilio Soria-Olivas[1]
[1]IDAL, Intelligent Data Analysis, Dept. of Electronic Engineering, ETSE, University of Valencia, Spain
[2]Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain

Corresponding author: Ricardo J. Rodríguez (e-mail: rjrodriguez@unizar.es).

The research by Ricardo J. Rodríguez was supported in part by the Spanish Ministry of Science, Innovation and Universities under grant MEDRESE-RTI2018-098543-B-I00, by the University, Industry and Innovation Department of the Aragonese Government under Programa de Proyectos Estratégicos de Grupos de Investigación (DisCo research group, ref. T21-20R), and by the University of Zaragoza and the Fundación Ibercaja under grant JIUZ-2020-TIC-08.

ABSTRACT Malware detection is a problem that has become particularly challenging over the last decade. A common strategy for detecting malware is to scan network traffic for malicious connections between infected devices and their command and control (C&C) servers. However, malware developers are aware of this detection method and begin to incorporate new strategies to go unnoticed. In particular, they generate domain names instead of using static Internet Protocol addresses or regular domain names pointing to their C&C servers. By using a domain generation algorithm, the effectiveness of the blacklisting of domains is reduced, as the large number of domain names that must be blocked greatly increases the size of the blacklist. In this paper, we study different Long Short-Term Memory neural network hyperparameters to find the best network configuration for algorithmically generated domain name detection. In particular, we focus on determining whether the (complex) feature engineering efforts required when using other deep learning techniques, such as Random Forest, can be avoided. In this regard, we have conducted a comparative analysis to study the effect of using different network sizes and configurations on network performance metrics. Our results show an accuracy of 97.62% and an area under the receiver operating characteristic curve of 0.9956 in the test dataset, indicating that it is possible to obtain good classification results despite avoiding the feature engineering process and additional readjustments required in other machine learning techniques.

INDEX TERMS deep learning, LSTM, malware, domain generation algorithms

I. INTRODUCTION

THE malware industry has become one of the most profitable illegal businesses in the world. According to Europol, the impact of cybercrime reached close to US $3 Trillion in 2013, that is, it is more profitable than the global trade in marijuana, cocaine, and heroin combined [1].

Banking trojans, ransomware, and other malicious software are examples of malware used by cybercriminals for damage and profit. Malware samples are analyzed to understand their behavior and implement the appropriate protection mechanisms. However, manual reverse engineering is a time-consuming task that has become practically impossible given the increase in the number of malware samples found in the last decade [2]. For instance, VirusTotal reports that, on average, more than 550,000 new samples are analyzed per day [3]. As a result, interest in automated analysis and classification methods has increased in recent years, with numerous articles published on this topic [4].

A common approach to detecting malware is to monitor its behavior from the network, since today most malware samples communicate with a Command & Control (C&C) server [5], usually using the Hypertext Transfer Protocol (HTTP). Destination Internet Protocol (IP) addresses and Domain Name System (DNS) resolutions are a good source of information that can be obtained from the network. When a malware sample connects to a specific IP address or DNS hostname, actions are taken to block this communication, based on a previously defined list of malicious or disreputable IPs and hostnames, called blacklists. These blacklists are frequently updated and revised to include the C&C servers

of the most recently released malware samples.

To reduce the effectiveness of these blacklists, malware developers began to use a different approach. Instead of using static hostnames or multiple hostnames that belong to a specific domain, they use algorithms to deterministically generate domain names. These algorithms are called Domain Generation Algorithms (DGA[1]). Malware using these algorithms generates a domain name at runtime and attempts to resolve it, repeating the operation until it can connect to a valid C&C server. This technique became popular due to the massive infection of the Conficker worm [6], as it allowed botmasters to deploy a new C&C server when the old one has been seized and thus continue to grow their botnet.

A domain name, or more precisely named Fully Qualified Domain Name (FQDN) [7], is a sequence of strings separated by a period character that can be managed as a single string representing a host specific on the network. An FQDN can be divided into Top-Level Domain (TLD), domain name, subdomains, and hostname. For instance, in the FQDN `mail.google.com`, `com` is the TLD, `google.com` is the domain name, and `mail.google.com` is the host name. Normally, humans can read an FQDN as a sentence in English or any other language, since it is generally created in a readable and memorable way. This is the main motivation of companies when choosing their brand, products, and domain names. In reality, this way of choosing domain names generated disputes and has resulted in companies paying large amounts of money to use the domain name they want to use [8].

By contrast, algorithmically generated domain names serve a completely different purpose. They are not created to be easy to remember and write, but rather to generate thousands deterministically using a snippet of code so that both the malware and the creator of the malware know which domain names will be generated in the near future. As a result, the generated domain names are seemingly random or strange to the human eye. Specifically, there are four different DGA generation schemes with different levels of randomness [9]: hash-based and arithmetic DGA-based domain names look like random characters, while permutation-based and wordlist DGA-based domain names are generated based on a sequence of words from a list of words or based on permutations of an initial domain name.

If we think about how a human would identify these algorithmically generated domain names, we would see that they would start reading from the first character of the FQDN to the last. Even before finishing the reading process, they would realize that the sequence looks strange compared to other more legitimate FQDNs. As we can see, following this approach, the concept of sequence is very important for FQDN classification problems, since the same set of characters distributed in a different order can change the result of the classification. For example, a certain FQDN like `myclassificationproblem.foo` would contain the same characters and length as `msolpiflbitecycraamoisn.foo`, although the latter should probably be classified as malicious (or at least considered a candidate for manual review).

In this paper, we study how Long Short-Term Memory (LSTM) Neural Networks can be used to detect DGA-based domain names. In particular, our main research question is to explore whether we can obtain an LSTM model that still performs well despite using a small dataset, avoiding the complex feature extraction and feature engineering inherent in other machine learning techniques. The use of a small dataset is motivated to facilitate comparison with other approaches, as we explain in more detail below.

The contribution of this paper is twofold. First, we have studied the impact of different hyperparameters of the LSTM neural networks on their performance, specially focused on finding the best network configuration for the detection of DGA-based domain names. To date, and to the best of our knowledge, this kind of study has not been explored in depth before. Second, we have compared the results of the best network configuration found with the results obtained in our previous work [10], which was based on masked N-Grams and Random Forest. This previous approach required us to spend a significant amount of time on the feature engineering process. This comparison allows us to decide whether or not the feature-engineering process can be skipped and to quantify the impact of this decision in the detection results. To this extent, we need to use the same (small) dataset in both approaches to compare them fairly. At the same time, the fact of using such a small dataset makes the neural network that we obtain is simpler in terms of size and complexity than other approaches, as otherwise the network would exhibit stability issues (i.e., it would not converge). We elaborate further on this issue when discussing the experimental results.

This paper is organized as follows. Section II provides some background on LSTM neural networks to help the reader understand the rest of this paper. Section III presents related work. Section IV describes the dataset, the encoding strategies that we follow to represent a sequence of characters, the deep learning models used, and the design of experiments. Section V discusses our findings. Finally, Section VI concludes the paper.

II. BACKGROUND: LONG SHORT-TERM MEMORY NEURAL NETWORKS

LSTM [11] is a specific design of Recurrent Neural Networks (RNN) [12] where neurons are distributed in four different pieces. Each of these pieces works as an independent neural network designed for a specific purpose, and some of them control the behavior of the input, output and forget gates. Together, they build an LSTM cell. When we refer to an LSTM with 128 neurons, we mean an LSTM network where each of these pieces within each LSTM cell has 128 neurons.

A gate is a product operation that controls whether a given value is passed on to the next operation or not, and how that

[1]In this paper, we use DGA interchangeably as a singular and plural acronym

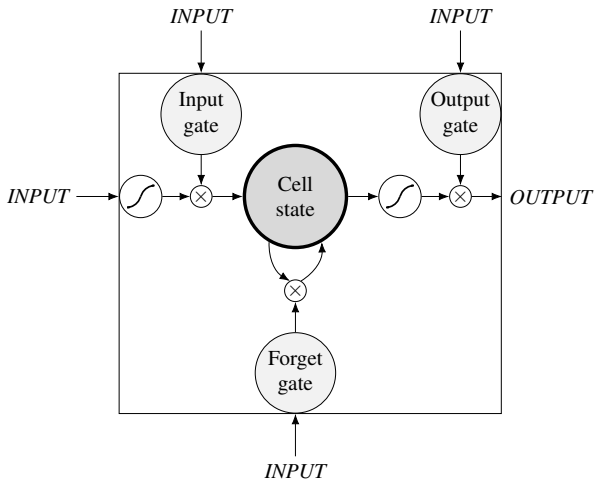

Figure 1. LSTM design (simplified).

value is affected by the gate. For instance, when a value of 2 is received as input, the input gate with value 0 results in that input value not being passed on to the next operation (i.e., the gate is closed), whereas a value 1 causes the input to be transmitted (i.e., the gate is opened). As shown in Figure 1, the gates are controlled by a layer of neurons that use a sigmoid activation function in the output layer. This sigmoid function makes the output value in the range of 0 to 1, which is a desirable range of values to set the opening level of that gate.

The input gate controls how the network takes current input into account. The output gate controls how the internal state affects the prediction or classification. Finally, the forget gate controls when some of the previously learned information should no longer be considered in subsequent iterations. In other words, the gates are designed to allow the network to choose what information to learn and what information to forget [11].

In addition to the NNs that control the gates, there is another NN responsible for generating the classification results, given an input and the information learned by the network in previous iterations. In Figure 1, this NN includes a hyperbolic tangent activation function, although other activation functions (such as softsign) can also be used.

LSTMs have been found to be very successful in time series processing due to their ability to learn long-short term dependencies, including DGA detection [13]. Well-known examples of products that use LSTM networks are Google Translator, Apple Siri, or Amazon Alexa, to name a few [14].

III. RELATED WORK

Neural Networks (NN) have been widely used to deal with the DGA detection problem, with good results. The advantage of these models is their ability to process raw data with minimum pre-processing. In our previous work, a significant amount of time was spent analyzing the data and extracting characteristics from it. This process requires a deep knowledge of the problem and the data being analyzed, and it is prone to errors such as not finding the most representative characteristics or not normalizing and preparing them properly, which results in unpredictable behavior and, in general, less accuracy and worse performance of the model.

Likewise, LSTM networks have been widely used to solve natural language processing, speech recognition [15], and other problems where the concept of sequence is important. Woodbridge et al. [13] propose a NN composed of an embedding layer, a LSTM layer of 128 neurons, and a dense layer. The first two are used for feature extraction, while the last is in charge of classification. They compare the results obtained with three different models: a hidden Markov model without distinctive features, logistic regression on character bigrams, and a random forest using manually-crafted domain features (similar to the lexical features proposed by da Luz and other authors). Their results show that the proposed model can classify 98% of DGAs with a FPR of 0.001.

Tran et al. [16] introduce an improvement over Woodbridge's proposal by including the use of an LSTM model with multiclass imbalance (LSTM.MI). In this model, the LSTM network includes cost items in its back-propagation learning mechanism, so the impact of a classification error differs depending on the type of error. Their model follows a two-stage approach: in the first stage, an LSTM.MI classifies domain names into DGA-based and non-DGA-based domains. At this stage, costs are set in such a way that a higher cost is set when classifying a legitimate domain as a DGA-based domain than the other way around. This setting prevents legitimate domains from being blocked as malicious by mistake, which is a desirable setting in protection mechanisms with automatic blocking capabilities. In the second stage, if the domain name was previously classified as DGA-based domain, a second multi-class LSTM.MI classifies the domain name in the specific family that generated it, giving the final result of the classification.

Convolutional Neural Networks (CNN) have been used successfully to solve the DGA classification problem. Catania et al. [17] propose a NN composed of an embedding layer, a one-dimension convolutional layer (1D-CNN), and dense layers. Using a CNN instead of a LSTM resulted in an increase in performance, since convolutions can be executed concurrently, while LSTM requires sequential execution, reducing the opportunity for parallel computation. They compare their results with the model proposed by Woodbridge et al. [13], although the evaluated model is not exactly the same as the model proposed by Woodbridge et al. (the greatest difference is in the activation function used in the LSTM layer). Both models are trained for 10 epochs with 70% of the dataset, and tested with the remaining 30%. The LSTM model shows a True Positive Rate (TPR) value around 94% and a False Positive Rate (FPR) value close to 3%, while the CNN model shows a TPR value around 97% and a FPR value of 0.7%.

As described above, CNNs have been used successfully to classify algorithmically generated domain names. However,

they are discarded from our approach as using a 1D-CNN with size N is, in essence, a similar approach to the one we followed in our previous work, where we used N-Grams to detect chunks of the string that can result in a classification decision. For instance, using a convolution layer of size 3 in the string "facebook" results in that string being split into "fac", "ace", "ceb", "ebo", "boo", "ook". Each one of these pieces would be convoluted to result in a number that would be the input of a dense layer of neurons. The weights in the convolution matrix would provide an abstraction of each specific N-Gram, which is similar to the manual masking of N-Grams that we perform in [10], but letting the network learn the best representation, in a similar way as embedding layers do.

Liu et al. [18] propose an improvement over the Recurrent Convolutional Neural Network (RCNN) initially proposed by Lai et al. [19], which is a combination of a Bidirectional LSTM (Bi-LSTM) layer and Convolutional layers. The Recurrent Convolutional Neural Network with Spatial Pyramid Pooling (RCNN-SPP) involves a modification in the pooling algorithm used in the convolutional layers to improve the representation of domain name features. In this modification, multiple filters with different sizes are used to obtain feature representations which are then combined to generate the final representation. This proposal achieves an accuracy of 92% for both binary and multi-class classification.

Yang et al. [20] propose a Heterogeneous Deep Neural Network (HDNN), composed of two different models. First, an Improved Parallel CNN (IPCNN) architecture that uses multiple CNNs with different kernel sizes combined together to extract local features at different scales. Second, a Self-Attention based Bi-LSTM (SA-Bi-LSTM) that extracts global features. Finally, IPCNN and SA-Bi-LSTM layers are combined together to provide a final classification result. The experiments show that HDNN performs better than other previous approaches with the dataset under evaluation.

Table 1 compares the methodologies of the related works listed above that are focused on detecting algorithmically generated domain names. This table also compares their performance metrics, which are discussed in more detail in Section V-A.

Compared to the related work, this paper has two main contributions. First, we conducted an in-depth study of the simplest and fastest LSTM network design that can achieve results similar to those obtained in [13, 17], but with less complexity of the network model. This in-depth study was not considered by Woodbridge et al. [13], who proposed their models with a specific and static configuration. Trainable parameters (also known as learnable parameters) are the layer weights that are updated by the back-propagation mechanism and that contain the information learned by the network from exposure to the training data [21]. Therefore, the complexity of the network (i.e., its size) increases with regard to the number of training parameters, since the number of parameters to be adjusted in the training process also increases. This complexity increases substantially when different models are combined, as the authors in [18, 20] do, because the use of bidirectional networks and convolutional layers using different kernel sizes considerably increases the number of trainable parameters. Second, we also compare our results with the results obtained from our previous work to verify whether models with automatic feature extraction, such as neural networks, can perform as well as models that include a previous process of manual feature engineering, saving the amount of time required to perform such a task.

Finally, let us recall that using complex approaches that are based on large datasets is not usually seen as a problem, as long as they provide better results overall. However, in this work we have deliberately imposed a strong constraint: the use of a small dataset rather than a large dataset. This decision is motivated because our goal is to compare our results in this work with our previous work [10], to show if the time-consuming task of the feature engineering process can be skipped and still get a model with good classification results.

IV. DESCRIPTION OF EXPERIMENTS

This section first describes the dataset that we built for experimentation and then the features that are extracted from the domain names. Finally we present the design of the network model and the experiments.

A. DATASETS

Using pure lexical features has several advantages over other approaches [22, 23, 24, 25, 26]. In a pure lexical approach, the list of malicious domain names is sufficient to train the model. This list can be generated from the algorithms used by different malware families. These algorithms are generally published in threat intelligence reports, where the algorithms are obtained by malware analysts using reverse engineering techniques.

However, the features used in the detection of algorithmically generated domain names depend to a large extent on information that can change over time. For instance, the information obtained from a DNS response today may differ significantly from the information obtained a few weeks ago. Therefore, it is difficult to get a good dataset of algorithmically generated domain names, especially for those malware families that are currently inactive or cannot be artificially generated, since no real DNS responses can be obtained.

In our previous work we proposed a new feature engineering scheme using the lexical features proposed by da Luz [26] and other authors, and adding a new set of features based on the occurrence of masked N-Grams [10]. Using Boruta's feature selection algorithm [27], we concluded that it was possible to get an accuracy of up to 98.73% using 15 features based on pure lexical information, which is similar to the accuracy obtained when non-lexical features are also used. However, the Random Forest approach requires a prior feature engineering process, which is complex, very time consuming, and has a strong impact on model results.

Table 1. Methodologies and metrics of our proposal and the related work (the values of other works were taken from their results).

Authors	Method	Accuracy	TPR	FPR	AUC	Precision	Recall	F1-score
Woodbridge et al. [13]	LSTM	–	0.9800	0.0010	0.9993	0.9942	0.9937	0.9906
Tran et al. [16]	LSTM.MI	–	–	–	–	0.9842	0.9842	0.9842
Catania et al. [17]	CNN	–	0.9700	0.0070	–	–	–	–
Liu et al. [18]	RCNN	0.9236	–	–	0.9539	0.9236	0.8955	0.9046
Yang et al. [20]	HDNN	0.9773	–	–	–	0.9793	0.9751	0.9772
Selvi et al. [10]	RF	0.9873	0.9859	–	–	0.9859	0.9888	0.9873
Selvi et al. (this paper)	LSTM	0.9762	0.9757	0.0233	0.9955	0.9770	0.9762	0.9762

To avoid this prior process, we explore in this paper whether a neural network approach is suitable for detecting algorithmically generated domain names. To evaluate it, we need two datasets that represent a good example of legitimate domain names (such as "google.com" or "facebook.com") and algorithmically generated domain names used by malware samples from different families. Unfortunately, the datasets used by other authors such as Woodbridge et al. [13] or Catania et al. [17] are not public, so we need to create our own dataset. In particular, we use the datasets generated from our previous work [28], made up of $32,000$ legitimate and $32,000$ DGA-based domain names. Furthermore, this allows us to compare the results obtained here with those obtained in our previous research.

B. FEATURES

One of the main goals of this research is to assess whether the intensive time-consuming task of the feature engineering process (initial feature extraction, normalization, and feature readjustment) can be reduced by using a deep learning approach, which is a NN with multiple parameters and training with a big amount of data. Each of the layers in a deep learning architecture transforms the input data into a more abstract representation. A well-known example of this characteristic is the face detection problem, where neurons in the first layer are specialized in detecting vertical and horizontal lines. The next layer uses those lines detected in the previous layer to detect simple shapes such as squares or circles, and the following layers increase the complexity of the information they are able to identify. As the information is processed by additional layers, elements such as eyes, nose, etc., are detected. Finally, the last layer detects the presence of a face, based on the multiple levels of abstraction that the network applied to each of the layers. Due to this characteristic, NNs are particularly good at managing unstructured data and have been used successfully in other fields such as image processing, natural language processing, etc [12], as the most important features are automatically learned by the network directly from the input as a part of the training process. Therefore, the use of NN simplifies the feature engineering process, as extracting as many features as possible from each sample is not a requirement. Thus, our feature array contains the character sequence in the FQDN string.

In the field of computer science, strings are generally represented as an array of numerical values between 0 and 255 (that is, one byte) that represents one character, according to the well-known American Standard Code for Information Interchange (ASCII) standard [29]. However, this representation does not work particularly well in many Machine Learning (ML) models, since it creates a similarity between characters with contiguous ASCII codes, when in reality the sequence of ASCII characters does not follow any sense of continuity between them. For instance, given the strings "AAAAA," "BCDFG," and "UUUUU", a model that uses an ASCII representation will probably conclude that strings "AAAAA" and "BCDFG" are more similar to each other than "UUUUU", because "A" and "B" are represented by similar numerical values in their ASCII representation, whereas a human will probably say that "AAAAA" and "UUUUU" are more similar in that they are made up of a sequence of vowels.

To solve this problem, the concept of embedding was introduced. Embedding is a technique used to represent data (such as a word or a character) as a series of features that define that particular data. Most NN models include embedding as an optional layer that can be configured in the network design. When this layer is included in the network, it is affected by the training process in the same way as other layers in the network. As a consequence, the training process will find the optimal representation for each piece of information (i.e., a word, a character, etc.) [12].

Although this was the approach followed by Woodbridge et al. [13], under certain circumstances a static representation has been proven to provide good results. For instance, `Word2vec` [30] is a representation that maps each word in natural language space to a vector of hundreds of features that represents the meaning and context of the word. This representation is widely used by computer scientists to deal with natural language processing problems.

However, FQDNs are not necessarily made up of words, so we cannot use `Word2vec` for our experiments. For this reason, we follow a simple static approach called One-Hot Encoding [12]. One-Hot Encoding is a widely used technique for representing categorical features in classification problems, where each element is represented by a matrix of the same size as the number of possible categories. In our specific

Table 2. Examples of one-hot encoding.

foobar	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	...
f	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	...
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	...
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	...
b	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	...
a	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

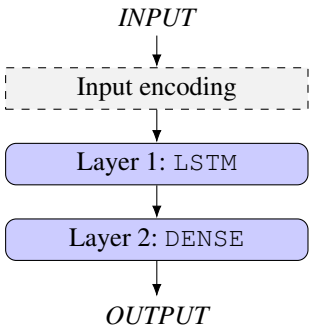

Figure 2. Network configuration for DGA detection.

problem, our representation is an array of size 38 (the English alphabet plus digits, the dash, and the dot symbols), where the first position set to 1 represents that the element is a letter "a", the second position represents that the element is a letter "b", and so on. Obviously, arrays with more than one position set to 1 are not valid, since an element cannot be represented by two different characters at the same time. Some examples of one-hot encoding representations can be found in Table 2.

Finally, LSTM requires all the samples included in a training batch be the same length. According to Request For Comments 1035 [7], the length of an FQDN is limited to 255 characters and each subpart of the FQDN is limited to 63 characters. To avoid this problem, we extend all the FQDN names in our datasets to the longest length sample by adding null characters, represented by a null vector of characteristics, to the end of the FQDN.

C. EXPERIMENTAL MODEL AND SCENARIOS

For our experiments, we design a network composed of two layers, as shown in Figure 2. We first encode the input appropriately for the LSTM model. In particular, as explained above we use a one-hot encoding strategy [12]. The first layer (`LSTM`) is made up of a LSTM cell, while the second layer (`DENSE`) is made up of a dense layer that interconnects each output of the previous layer to obtain a single output.

Regarding the design of experiments, several parameters of the network have been modified during the experiment, among them: (a) network size; (b) activation function; and (c) batch size in the training phase. We detail these changes in the following paragraphs.

First, different sizes of LSTM (5, 15, 25 and 50 neurons) are used in our experiments. The size of the network has a significant impact on the results, as smaller networks often require a smaller training dataset, train much faster, and classify with better generalization, as it is more difficult to find overfitting. However, larger networks can accommodate more complex datasets and answer more difficult classification problems. Therefore, finding a balance between the two is one of our goals in this document.

Second, all the activation functions supported by Keras [31], a well-known library for building NNs, are used in each of the NNs inside the LSTM cell. However, most of them are discarded from our study since they make the model unstable, as it does not converge. In particular, for the NNs that manage the gates, any activation function other than the default (sigmoid function) results in the model having unpredictable behavior. Similarly, several activation functions are also tested for the NNs that generate the classification. In this case, only the hyperbolic tangent (the default) and the softsign function result in the model being stable. Therefore, only these functions are included in our final analysis.

Third, two different batch sizes (10 and 100 items) are used in the training process. During the training phase, the NNs process each item in the dataset and the result is compared with the expected result. The difference between the expected and the actual result is used in the back-propagation mechanism, which recalculates the weight of each neuron to improve classification. This mechanism can be run for each input or for each batch of inputs. The batch size defines the number of inputs that are provided to the network before running the back-propagation mechanism.

With regard to model convergence, our preliminary experiments revealed that a maximum number of 1000 epochs was sufficient to allow the model to converge. In addition, since the best performance is not necessarily achieved in the last epoch, we keep the network configuration that achieves the best performance. Finally, to prevent the learning procedure from getting stuck at a local minimum until it reaches the 1000 epoch limit, a callback was introduced into the learning process to produce an early stop if the training does not get an improvement in accuracy during the last 100 epochs.

This model is finally verified using a Hold-Out strategy [32]. First, we randomly divide the dataset into two smaller datasets (50% each). One of these resulting datasets is used as a Test dataset, whereas the other is randomly divided again: 80% is the Training dataset, while the other 20% is the Validation dataset. The Training dataset is the dataset used in the training process and the validation dataset is the dataset used to evaluate the performance of the network for each epoch. The validation dataset is also used to choose the network configuration that provided the best results for our classification problem. Finally, the Test dataset is used to measure the precision of the model.

Batch size	Number of neurons			
	5	15	25	50
10	0.9595	0.9655	0.9648	0.5951
100	0.9592	0.9655	0.9669	0.9678

Table 3. Best accuracy achieved with hyperbolic tangent activation function.

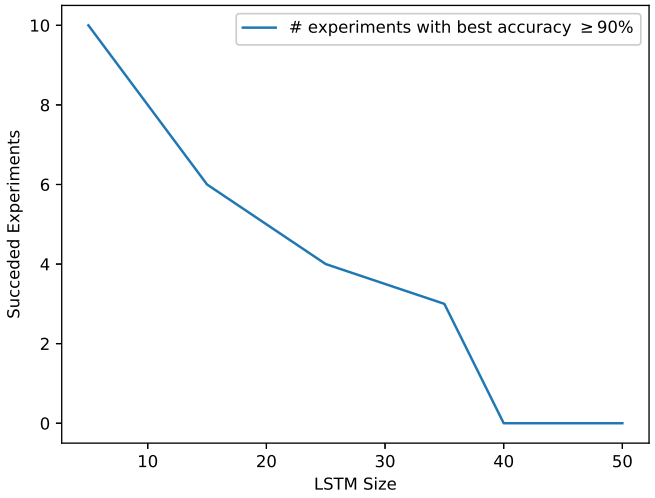

Figure 3. Experiments with best accuracy above 90%.

V. EXPERIMENTAL RESULTS

This section first discusses the results of our experiments and then illustrates them with some running examples.

A. DISCUSSION OF RESULTS

First, we analyze how different batch sizes impact network performance. Table 3 shows that a smaller batch size has a greater impact on accuracy as the network size increases. Using smaller batch sizes means that the network weights are updated more frequently, based on the error obtained from a small subset of the training dataset. In contrast, using larger batch sizes produces a smoother update in network weights. In particular, we observe that when using a network size of 50 neurons and a batch size of 10 elements, the performance of the network drops dramatically. Figure 4 shows how smaller batch sizes result in a less consistent increase in accuracy across epochs, while larger batch sizes produce a more stable increase, even when both achieve a good accuracy. This lack of stability when using small batch sizes with a larger LSTM size meant that more experiments did not converge on good accuracy results. This fact is observed in Figure 3, where the percentage of successful experiments decreases as the network size increases, reaching a value of zero for the LSTMs of 50 neurons.

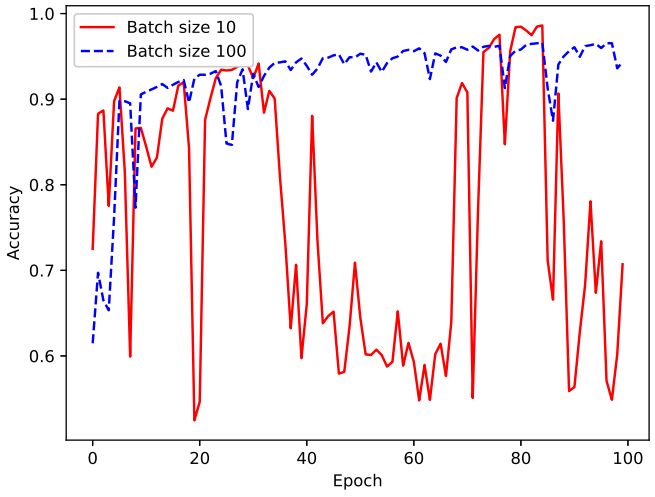

Figure 4. Between-epoch accuracy for LSTM of 25 neurons. Batch size 10 (red) and 100 (blue).

Table 4. Best accuracy using a batch size of 100.

Activation function	Number of neurons			
	5	15	25	50
htan	0.9592	0.9655	0.9669	0.9678
softsign	0.9594	0.9652	0.9652	0.9681

As mentioned in the previous section, most of the activation functions resulted in an unstable network, except for the hyperbolic tangent (htan) and softsign functions. To evaluate network performance using both activation functions, we use the same network configuration. We tested both options for each size of network, using a batch size of 100, and measured the best accuracy obtained for the validation dataset over ten consecutive experiments.

The results obtained in Table 4 show that the accuracy for most network configurations is almost the same, regardless of the use of htan or softsign as activation functions. However, the softsign activation function worked better for larger network sizes when using a smaller batch size, as shown in Table 3. Based on these results, we have used softsign as the activation function for the rest of the experiments.

When analyzing the results considering all the network sizes included in our experiment, we observe that the larger the network size, the better the accuracy achieved. A network size of 50 neurons provided better results in the validation dataset than the results obtained with smaller network sizes. However, we observed only a slight improvement in the results when doubling the size of the network from 25 to 50 neurons.

Furthermore, we observed that, although the best result in a set of experiments improves when the network size is increased, the average performance of all experiments decreases as the network size increases, in particular with small batch sizes. This behavior is probably motivated by the fact that a small batch size increases the probability of large changes in network weights. This generally lowers the average performance, in a similar way to that described by [33] when the learning rate is too large.

Table 5. Confusion matrix.

	Classification	
	MALWARE	**CLEAN**
Real Malware	15731	391
Real Clean	369	15501

The configuration that provided the best results for our classification problem was obtained with a LSTM network of 50 neurons, the softsign activation function, and a batch size of 10 elements. This network configuration was tested with the Test dataset, which was not used in the previous analysis and design, resulting in the confusion matrix shown in Table 5 in which the rows show the dataset the samples belonged to and the columns represent the results of the classification.

Note that our experiments only include the analysis of hyperparameters such as network size, employed activation function, and batch size. Other hyperparameters were set to their default values. Although our model achieves good results, further experiments are needed to investigate how optimizing other hyperparameters affects the performance of the proposed model.

The model obtained an accuracy of 0.9762, and an Area Under the Curve (AUC) of 0.9955 in the testing dataset. The AUC represents a way to measure the performance of the approach, since it represents the number of false positives combined with false negatives. Since the AUC is close to one, this means that the proposed model provides a good approach to the classification problem.

In particular, our true positive rate (TPR) was 97.57%, which represents DGA-generated domain names that were correctly classified by our model. On the other hand, our false positive rate (FPR) was 2.33%, which represents legitimate domain names that were mistakenly classified as algorithmically generated. Taking into account the Alexa Top 1000 domains, this FPR was reduced to 1.7% and further reduced to 0% versus the Alexa Top 100 domains (we further extend this discussion in Section V-B). Finally, we have also calculated the Matthews Correlation Coefficient (MCC), a specific indicator of the quality of two-class classifications. MCC has scored 0.9525, indicating a correlation close to a perfect agreement. Table 6 shows all the performance metrics of our approach, including also Cohen's Kappa coefficient (which compares an observed accuracy with an expected accuracy). Figure 5 shows the Receiver Operating Characteristic (ROC) curve of our model.

It is worth noting that we did not reproduce the experiments of Woodbridge et al. [13] or other authors with our own dataset for two main reasons. First, they did not share their code and datasets, so it was very difficult for us to properly compare their performance against our proposed model using the same dataset, since the results are highly dependent on the composition of the dataset. For example,

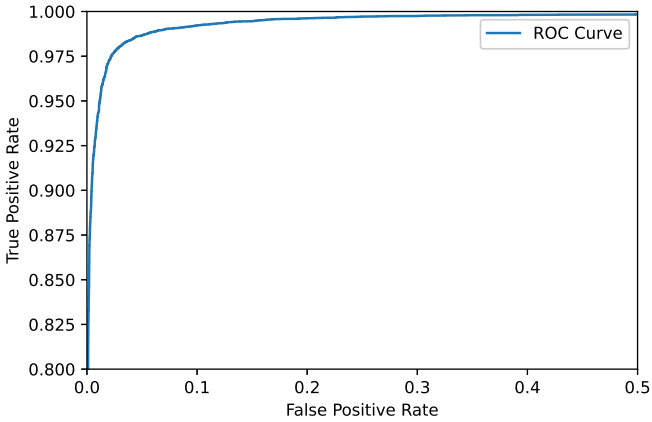

Figure 5. ROC curve of the proposed model.

Table 6. Performance metrics of our approach.

Accuracy	**AUC**	**TPR**	**FPR**
0.9762	0.9955	0.9757	0.0233
Recall	**F1-score**	**Kappa**	**MCC**
0.9762	0.9762	0.9524	0.9525

using a dataset where the presence of wordlist-based DGA is lower may result in better accuracy in models that detect strange combinations of characters, which is something that can happen when using an approach of CNN. Second, our dataset is much smaller than the datasets used by other authors such as Woodbridge et al. [13]. The size of our dataset is intentionally smaller so that the results are comparable with other techniques that require a prior feature engineering process, as we did in our previous work [10]. Unlike our case, the models designed by others have many trainable parameters that need to be adequately trained with larger datasets. Due to the lack of details to satisfactorily reproduce other works, we decided to provide our results as well as the results published by other authors, for the sake of completeness.

Regarding the results of the related work, the authors in [13] also used an LSTM approach with a dataset of almost 2 million elements and got 98% of DGA-based domain names detected with an FPR of 0.1%, with a total AUC of 0.9993. Their approach was later re-evaluated for comparison in [17], resulting in a TPR of 94% and an FPR of 3%. The model proposed in [17], which relies on a 1D-CNN, obtained a TPR of 94% and an FPR of 3%. We have summarized in Table 1 the metrics of the related work and the metrics of our approach. Let us remark that the values of the related work were taken from their results and therefore not directly comparable with ours. As we have already mentioned, our results are only directly comparable to the results of our previous approach given in [10] as we are using the same dataset. We have highlighted both comparable results in Table 1. More information is provided below. Following best practices [34] and for the sake of reproducibility, we have

released the models used in this paper [35]. The dataset that we are using in this work is also publicly available in [28].

The results of both approaches given in [17] show that our LSTM approach outperforms the LSTM approach in [13], and achieves a TPR rate similar to that of the 1D-CNN approach introduced by Catania et al. [17]. However, the FPR of their approach is much better than ours. This is because the decision boundary threshold of their model was changed to 0.90, instead of the default value of 0.50. This modification causes the network to classify a domain name as algorithmically generated only if the probability of this classification is greater than 90%, which obviously results in a smaller FPR and a higher true negative rate (TNR), but also in a smaller TPR and a higher false negative rate (FNR). This kind of approach makes sense when domain names detected as positives are automatically blocked, as it is preferable never to block legitimate domain names, even if multiple malicious domain names are not blocked. However, the opposite approach is preferred when the model is used for passive detection. In passive detection situations, it is preferable to identify all malicious domain names, even when some of them result in false positives. In our approach, we prefer to keep the model balanced for performance measurement, as our model can be used interchangeably for active or for passive detection.

Moreover, as described in [17], the authors used an imbalanced dataset, comprising a million clean domain names and nearly 2 million domain names algorithmically generated. This resulted in a higher TPR, since the class representing a positive classification is much larger in the Training dataset. Unlike [17], our dataset is balanced and much smaller ($64,000$ domain names). We opted to use such a small dataset as our main research goal was to find the simplest model that can be trained with a reduced dataset and yet obtain good accuracy results.

Comparing our current approach using LSTM to our previous approach using Random Forest (RF) [10], we got slightly better accuracy in our RF approach (98.73% vs. 97.62%). This means that the feature engineering performed in our previous work provided a good representation of the data. At the same time, it also means that we have created a model that obtained almost the same accuracy but without spending time in a complex and time-consuming feature engineering process. Avoiding the feature engineering process is particularly useful when it is required to be done frequently. This can happen in adversarial scenarios such as those described by [36]. Malware developers can modify the way DGA-based domain names are generated, so the features extracted from the previous feature engineering process are no longer useful. In such a scenario, LSTM-based approaches can be retrained with the new set of domain names and extract new features that continue to produce good classification results.

B. RUNNING EXAMPLES

Here we discuss some real examples and interesting results from our model. Focusing on a specific domain like `facebook.com`, the model classified it as `CLEAN`, which was the expected behavior. Similarly, the domain `dasqwij28ndnas812endmq83.com` was classified as `MALWARE`, which was also the expected behavior (this domain was algorithmically generated by one of the malware samples whose DGA is included in our datasets).

We did some additional experiments to better understand how classification works. For example, we chose the domain `somethingmalicious.com` (initially classified as CLEAN) and then we began to slightly modify the domain until our network detected it as MALWARE. We observed that the classification remains CLEAN when adding suffixes like numbers, such as `somethingmalicious1990.com`. However, the domain was detected as MALWARE by inserting numbers randomly along the string, such as `som9ething7mal6ici2ous.com`.

We have also observed that the network detects as MALWARE those domain names that include unusual combinations of consonants and vowels. For example, inserting a "w" character between the words that make up the `somethingmalicious.com` domain name creates the unusual combination of "ngwm" characters. Our model detects the resulting domain name (`somethingwmalicious.com`) as MALWARE.

Finally, we verified that the top 100 sites referenced by Alexa [37] were correctly classified as CLEAN and out of the top 1000 sites, only 17 were erroneously detected as MALWARE (representing 1.7% of false positives). These incorrectly classified domain names are shown in Table 7.

Table 7. False positives in Alexa Top 1000.

doubleclick.net	slideshare.net
slickdeals.net	adplxmd.com
secureserver.net	themeforest.net
trackingclick.net	daikynguyenvn.com
prjcq.com	bookmyshow.com
seesaa.net	inquirer.net
uploaded.net	torcache.net
youjizz.com	fanfiction.net
commentcamarche.net	

VI. CONCLUSIONS

One of the most commonly used techniques to detect malicious software and infected devices is to analyze network communications. To maintain control over infected systems, malware developers began to use a more stealthy communication strategy using algorithmically generated domain names instead of fixed IP addresses or domain names that are quickly blacklisted.

In this paper, we looked for the simplest possible neural network model that can be trained with a small dataset and still get good accuracy results. As a base model, the neural

network is composed of a LSTM layer and a dense layer. We used a one-hot encoding strategy to represent the input data from the network. In addition, we performed sensitivity analysis with the network parameters (such as LSTM layer size, activation functions, and batch size) to find the best performing settings.

Our experiments revealed that larger networks provide better classification accuracy. However, we observed only a slight improvement in the results when doubling the size of the network (in particular, from 25 to 50 neurons). This means that increasing the network size does not provide significant performance improvements. Additionally, larger batch sizes improve network stability, while smaller batch sizes increase the likelihood of obtaining the optimal configuration to provide the best accuracy. Based on the knowledge gained from these experiments, we proposed a network configuration that achieves detection rate accuracy of 97.62% in the testing dataset, with a very low false positive rate on the Alexa Top 1000 list (namely, a 1.7% rate).

For the sake of reproducibility and open science, we are publicly releasing our network model and datasets. This network model is able to learn how to identify various aspects of domain names that can reveal that they were algorithmically generated. For instance, the combination of unusual characters or the distribution of numbers or other symbols was found to be a factor in the classification results.

Although our LSTM-based approach achieves slightly worse results than approaches that rely on other methods such as Random Forest, it does not require any up-front feature engineering process. In particular, the near 1% degradation in accuracy, TPR, recall, and F1-score is affordable, as the model achieves good classification results while avoiding the need for the feature engineering process. Using LSTM-based approaches allows retraining without any human intervention. For instance, an autonomous system can provide new datasets of DGA techniques to the model to improve its detection metrics. In contrast, approaches based on other methods such as Random Forest require human intervention when new features need to be extracted as a consequence of an advanced adversary with sufficient knowledge of the underlying machine learning models to deceive and circumvent malicious classification.

As future work, we aim to investigate the addition of new restrictions in the proposed model, such as setting limits on the dynamic response classification based on network congestion or the workload of the system. In this regard, we envision a model that is capable of adapting to a situation of high network load, dynamically reducing latency, even when this implies reducing the performance of the classification results. We also plan to investigate how the optimization of other hyperparameters, in addition to network size, activation function, and batch size, affects the performance of the proposed model.

References

[1] Europol, "EU Serious and Organised Crime Threat Assessment," [Online; https://www.europol.europa.eu/sites/default/files/documents/socta2013.pdf], European Union Agency for Law Enforcement Cooperation (Europol), techreport, 2013.

[2] AV Test, "Malware Statistics & Trends Report," [Online; https://www.av-test.org/en/statistics/malware/], 2020, accessed on April 13, 2020.

[3] VirusTotal, "Statistics - VirusTotal," [Online; https://www.virustotal.com/en/statistics/], 2020, accessed on April 10, 2020.

[4] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

[5] J. Gardiner and S. Nagaraja, "On the Security of Machine Learning in Malware C&C Detection: A Survey," *ACM Comput. Surv.*, vol. 49, no. 3, Dec. 2016.

[6] P. Porras, H. Saïdi, and V. Yegneswaran, "A Foray into Conficker's Logic and Rendezvous Points," in *Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, ser. LEET'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 7–7.

[7] P. Mockapetris, "RFC 1035: Domain Names - Implementation and Specification," Internet Engineering Task Force, Tech. Rep., November 1987, available at http://www.rfc-editor.org/rfc/rfc1035.txt.

[8] J. Litman, "The DNS Wars: Trademarks and the Internet Domain Name System," *J. Small & Emerging Bus. L.*, vol. 4, p. 149, 2000.

[9] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A comprehensive measurement study of domain generating malware," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 263–278.

[10] J. Selvi, R. J. Rodríguez, and E. Soria-Olivas, "Detection of algorithmically generated malicious domain names using masked n-grams," *Expert Systems with Applications*, vol. 124, pp. 156–163, 2019.

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

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[13] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," *CoRR*, vol. abs/1611.00791, 2016.

[14] A. Vieira and B. Ribeiro, *Introduction to Deep Learning Business Applications for Developers: From Conversational Bots in Customer Service to Medical Image Processing*. Apress, Berkeley, CA, 2018.

[15] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal*

Processing (ICASSP). IEEE, 2015, pp. 4520–4524.

[16] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen, "A LSTM based framework for handling multiclass imbalance in DGA botnet detection," *Neurocomputing*, vol. 275, pp. 2401–2413, 2018.

[17] C. Catania, S. García, and P. Torres, "Deep convolutional neural networks for dga detection," in *Argentine Congress of Computer Science*. Springer, 2018, pp. 327–340.

[18] Z. Liu, Y. Zhang, Y. Chen, X. Fan, and C. Dong, "Detection of algorithmically generated domain names using the recurrent convolutional neural network with spatial pyramid pooling," *Entropy*, vol. 22, no. 9, p. 1058, 2020.

[19] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[20] L. Yang, G. Liu, Y. Dai, J. Wang, and J. Zhai, "Detecting stealthy domain generation algorithms using heterogeneous deep neural network framework," *IEEE Access*, vol. 8, pp. 82 876–82 889, 2020.

[21] S. Skansi, *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer, Cham, 2018.

[22] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *Proceedings of the Network and Distributed System Security Symposium, (NDSS 2011)*, 2011.

[23] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 4, pp. 14:1–14:28, Apr. 2014.

[24] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-Based Botnet Tracking and Intelligence," in *Proceedings of the 11th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Cham: Springer International Publishing, 2014, pp. 192–211.

[25] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, T. Kohno, Ed. USENIX Association, 2012, pp. 491–506.

[26] P. M. da Luz, "Botnet Detection Using Passive DNS," Master Thesis, Department of Computing Science, Radboud University Nijmegen, 2013.

[27] M. B. Kursa and W. R. Rudnicki, "Feature Selection with the Boruta Package," *Journal of Statistical Software*, vol. 36, no. 11, pp. 1–13, 2010.

[28] J. Selvi, R. J. Rodriguez, and E. Soria-Olivas, "Dga and legitimate dns domains," Feb. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.4643821

[29] S. Gorn, R. W. Bemer, and J. Green, "American standard code for information interchange," *Communications of the ACM*, vol. 6, no. 8, pp. 422–426, 1963.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013.

[31] Keras, "Keras Documentation: Use of activations," [Online; https://keras.io/activations/], 2020, accessed on April 27, 2020.

[32] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[33] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[34] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. v. Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 65–79.

[35] J. Selvi, R. J. Rodríguez, and E. Soria-Olivas, "Code and Dataset in a Docker Container," [Online; https://github.com/jselvi/docker-lstm-dga], 2021, accessed on March 29, 2021.

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

[37] Alexa, an Amazon Company, "Alexa Top Sites," [Online; http://www.alexa.com/topsites], 2020, accessed on April 27, 2020.

AUTHORS' BIOGRAPHIES

Jose Selvi is an Executive Principal Security Consultant in the cybersecurity firm NCC Group. In the last 15 years he has been delivering advanced security services and solutions in various industries, including penetration tests, incident handling, intrusion detection, forensic analysis, security assessments, and information security research in new technologies. Jose received Computer Engineering (BSc & MSc) and Telecommunication Engineering (BSc) degrees from the Universitat de València, Valencia (Spain). He is currently a PhD candidate, and his research is focused on exploring Machine Learning techniques to solve problems found in the cybersecurity industry.

In the professional field, Jose is a regular contributor to the infosec community. He has developed or contributed to several open source tools and he is a regular speaker at cybersecurity events such as DEFCON, BlackHat, Ekoparty, OWASP Appsec, etc.

Ricardo J. Rodríguez received MSc. and PhD. degrees in Informatics and System Engineering from the University of Zaragoza, Spain, in 2010 and 2013, respectively. He was a Visiting Researcher with the School of Computer Science and Informatics, Cardiff University, Cardiff, U.K., in 2011 and 2012, and the School of Innovation, Design and Engineering, Mälardalen University, Västeras, Sweden, in 2014. He was also a Visiting Professor at the University of Campania "Luigi Vanvitelli", Italy, during two three-month periods in 2016 and in 2018.

He is currently an Assistant Professor at the University of Zaragoza, Spain. His research interests include performability analysis, program binary analysis, and digital forensics. He is involved in reviewing tasks for international conferences and journals.

Emilio Soria-Olivas received a PhD. degree from the Universitat de València, Valencia, Spain, in 1997. He is a Full Professor at the University of València. His research is centered on the application of Advanced Machine Learning Methods to improve decision taking in different scenarios: economic, social, health, etc. Currently he is working on deep learning (representation learning) and automatic machine learning (machine learning without parameters).