IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 19, 2018, accepted November 27, 2018, date of publication January 31, 2019, date of current version March 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891588

A Machine Learning Framework for Domain Generation Algorithm-Based Malware Detection

YI LI[1], KAIQI XIONG[1], (Senior Member, IEEE), TOMMY CHIN[2], (Member, IEEE), AND CHENGBIN HU[1]

[1]Intelligent Computer Networking and Security Lab, Florida Center for Cybersecurity, University of South Florida, Tampa, FL 33620, USA
[2]Department of Computing Security, Rochester Institute of Technology, Rochester, NY 14623, USA

Corresponding author: Kaiqi Xiong (xiongk@usf.edu)

This work was supported in part by the Florida Center for Cybersecurity, University of South Florida, State University System of Florida, in part by the National Science Foundation (NSF) under Grant 1633978, Grant 1620871, Grant 1636622, and Grant 1620862, and in part by the BBN/GPO Project 1936 through an NSF/CNS Grant.

ABSTRACT Attackers usually use a command and control (C2) server to manipulate the communication. In order to perform an attack, threat actors often employ a domain generation algorithm (DGA), which can allow malware to communicate with C2 by generating a variety of network locations. Traditional malware control methods, such as blacklisting, are insufficient to handle DGA threats. In this paper, we propose a machine learning framework for identifying and detecting DGA domains to alleviate the threat. We collect real-time threat data from the real-life traffic over a one-year period. We also propose a deep learning model to classify a large number of DGA domains. The proposed machine learning framework consists of a two-level model and a prediction model. In the two-level model, we first classify the DGA domains apart from normal domains and then use the clustering method to identify the algorithms that generate those DGA domains. In the prediction model, a time-series model is constructed to predict incoming domain features based on the hidden Markov model (HMM). Furthermore, we build a deep neural network (DNN) model to enhance the proposed machine learning framework by handling the huge dataset we gradually collected. Our extensive experimental results demonstrate the accuracy of the proposed framework and the DNN model. To be precise, we achieve an accuracy of 95.89% for the classification in the framework and 97.79% in the DNN model, 92.45% for the second-level clustering, and 95.21% for the HMM prediction in the framework.

INDEX TERMS Malware, domain generation algorithm, machine learning, security, networking.

I. INTRODUCTION

Malware attackers attempt to infiltrate layers of protection and defensive solutions, resulting in threats on a computer network and its assets [1]–[3]. Anti-malware softwares have been widely used in enterprises for a long time since they can provide some level of security on computer networks and systems to detect and mitigate malware attacks. However, many anti-malware solutions typically utilize static string matching approaches, hashing schemes, or network communication whitelisting [4]. These solutions are too simple to resolve sophisticate malware attacks, which can hide communication channels to bypass most detection schemes by purposely integrating evasive techniques. The issue has posed a serious threat to the security of an enterprise and it is also a grand challenge that needs to be addressed.

Some of the sophisticate malware attackers use either a static or dynamic method to communicate with a centralized server to service a Command and Control (C2) [5]. In a static method, everything is fixed. For example, the malware has both a fixed IP address and a fixed domain name permanently (i.e., its domain name will not change throughout its lifespan). Thus, as long this malware has been identified as a threat, a simple rule can be applied to resolve this malware threat issue. In a dynamic method, Domain Generation Algorithm (DGA) [6] has been commonly used to communicate back to a variety of servers. The DGA is a sequencing algorithm that is used to periodically generate a large number of domain names, which are often used by malware to evade domain-based firewall controls. The generated domain names give malicious actors the opportunity to hide their C2 servers so that it is hard for the enterprise to identify the DGA. The domains generated by DGAs are short-lived registered domains and they are easier for human to identify but harder for machines to detect automatically.

The dynamics of a DGA commonly utilizes a seeded function. That is, given an input such as a timestamp, a deterministic output will follow as pre-defined by the DGA. The challenge behind deterring a DGA approach is that an administrator would have to identify the malware, the DGA, and the seed value to filter out past malicious networks and future servers in the sequence. The DGA increases the difficulty to control malicious communications as a sophisticated threat actor has the ability to change the server or location periodically the malware communicates back (callback) to the C2 in an automated fashion. There is a grant challenge in the detection of a DGA.

This study evaluates known DGA algorithms and malicious domains generated by those DGAs. In this research, we investigate machine learning approaches including multiple feature extractions, classification, clustering, and prediction techniques to understand those DGA domains. We also design a Deep Neural Network (DNN) model to classify a large dataset. Everyday, there are vastly running applications and services in the networking and their computer systems may frequently query domain names through DNS [7]. Security appliances that monitor and evaluate each DNS query need to determine whether a particular domain has some level of maliciousness, whether or not a specific query originates from a DGA and which DGA is originated from. Moreover, this study utilizes a real-time threat intelligence feed that has been collected over a one-year period on a daily basis while leveraging high-performance nodes [8], [9] from the Global Environment for Network Innovation (GENI) [10] to conduct extensive data processing.

In this paper, we first propose a machine learning framework to classify and detect DGA malware and develop a DNN model to classify the large datasets of DGA domains that we gradually collected. We then experimentally evaluate the proposed framework through a comparison of various machine learning approaches and a deep learning model. Specifically, our machine learning framework consists of the following four main components: (1) A dynamic blacklist consists of a pattern filter. The pattern filter is used to filter the incoming DNS queries in order to obtain the domains from them. Those filtered domains are stored in the blacklist. (2) A feature extractor. It extracts features from the incoming domains that are not in the blacklist. Those domains will be processed in the next component. (3) A two-level machine learning model: the first-level classification and the second-level clustering. To identify DGA domains, we first use various classification models to classify DGA domains and normal domains. Then, we apply the clustering method to group domains sequenced by the DGA. (4) A time-series prediction model: we propose a Hidden Markov Model (HMM) to predict incoming DGA domain features in order to better identify the DGA domains. The general goal of our machine learning framework is to determine which algorithm is employed so that our proposed framework can prevent future communications from the C2.

Furthermore, we have gradually collected the data for over one year and have obtained a large amount of datasets from real traffic. To analyze these data, we also propose a deep learning approach for large dataset classification. We first build a DNN model and then compare it with our machine learning models. The comparison results provide us an useful guideline for our future study in DGA detection and prediction. In our future research, we will also apply deep learning in clustering and prediction that are out of the scope of this paper.

In this research, our evaluation results show that we can achieve the accuracy of 95.89% for the first-level classification and the accuracy 92.45% for the second-level clustering, respectively. For our HMM model prediction, we can further achieve the accuracy of 95.21%. Our deep learning model for classification can reach the accuracy of 97.79%.

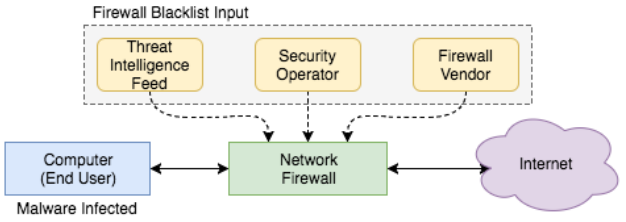
The main contributions of this paper are summarized here.

1) We propose a machine learning framework to perform DGA detection and prediction. We first distinguish the DGA domains from normal domains and then identify their generation algorithm. We also apply a time series model to predict DGA domains.
2) We propose a two-level model consisting of classification and clustering to first classify DGA domain names and then cluster the DGAs to groups of different DGAs. The first-level model (also referred to the classification model) in the framework can provide a high accurate classification. In the second model, we cluster DGA domain names into different groups using an unsupervised DBSCAN algorithm [11], [12], where DBSCAN represents density-based spatial clustering of applications with noise. By using the proposed two-level classification and clustering, we improve the accuracy of identifying certain domain names from a DGA. This is a fundamental requirement for future prediction of malicious attacks.
3) We design an HMM time-series predictor, which can be used to predict features to match the current features of domain names. The prediction results give the network a quick reference for blocking DGA domains. Because we do not check a DNS query for a feature match, with the predictor, we eliminate the risk of a communication with C2 servers when conducting an inspection.
4) We propose a deep neural network model to classify large DGA datasets. Different optimization algorithms are applied in our DNN model to obtain better accuracy. We separate training data from validation data in this research to prevent overfitting.

The rest of the paper is organized as follows. Section II gives the problem statement. Section III discusses related work. Section IV presents data collection, the proposed machine learning framework, and the deep learning model. Then, Section V discusses the evaluation of the machine learning framework and compares it with the results via DNN. Lastly, Section VI concludes our studies and presents future work.

II. PROBLEM STATEMENT

Firewall blacklisting constantly expands as the multiple sources of inputs expand filtering rules. However, sequences in a DGA may not be known to these inputs promptly. Moreover, for the malware that communicates with an appropriate domain correctly, a threat actor must register each respective domain name in the sequence to maintain the C2 or risk the loss of a node in the distribution. Figure 1 give a scenario for such a case.



DGA Sequence	Firewall Blacklist	Internet Domain	Filtered
swgqwmnpxuorh.com	swgqwmnpxuorh.com	swgqwmnpxuorh.com	Yes
ghluburrkboax.net	ghluburrkboax.net	ghluburrkboax.net	Yes
veivkqvvamlgx.biz	veivkqvvamlgx.biz	veivkqvvamlgx.biz	Yes
wykwohntbxrlh.org		wykwohntbxrlh.org	No
kjpbsprvnertx.co.uk			
ahmsbkdcffkst.info		ahmsbkdcffkst.info	No

FIGURE 1. Threat models: Multiple conditions for a DGA to function in a network environment where filtering results in a firewall that protects the communication and an empty cell in an Internet domain that results in an NXDOMAIN error. Note that the domains listed in the figure belong to existing live threats [13].

Our research problem is to accurately identify and cluster domains that originate from known DGA-based techniques where we target to develop a security approach that autonomously mitigates network communications to unknown threats in a sequence.

A. ASSUMPTIONS AND THREAT MODELS

Threat actors need a method to control and maintain the malware in a C2 environment while operating in an unnoticeable manner from network security systems. The successfulness of the malware does not require a domain to be registered or valid and a DGA may iterate a sequence that results in an NXDOMAIN situation (unregistered). Blacklisting, establishing a DNS sinkhole, and implementing a firewall rule are standard techniques to prevent a malicious network activity from malware and the signatures to implement these mitigation techniques are often provided by threat intelligence feeds. However, this research does not utilize any pre-defined blacklisting that contains pre-known malicious domains to block traffic derived from a DGA in the initial stages of our analysis and that such features are built over our observations. The main reason behind our implementation is that many threat intelligence feeds and heuristic data often provide signatures to malware that has plagued a network or public Internet. A sophisticated threat actor would implement or utilize a 0-day style malware (a malicious code that has never been seen or known to the public) and therefore, blacklisting would be inappropriate for our analysis. Our proposed machine learning framework aims to solve the problem of detecting DGA sequences using machine learning techniques derived from observations in a network.

III. RELATED WORK

As the Internet has become widely distributed, it is very vulnerable to malware hazards [14], [15]. Malware attackers can choose different targets or cyber-physical devices and attack them like mobile devices and connected vehicles. Many of the targets the threat actor attack are susceptible to malware attacks due to mismanagement issues, poor patching behaviors, and dangerous 0-day attacks [16].

To differentiate DGA domain names from normal domain names, researchers have discovered that DGA-generated domain names contain significant features [17]. Therefore, many studies aim to target blocking those DGA domain names as an defense approach [18], [19]. The DGA that generates the domain fluxing botnet needs to be known so that we can take countermeasures. Several studies have looked at understanding and reverse engineering the inner workings of botnets [20]–[25]. Barabosch *et al.* [26] proposed an automatic method to extract DGA from current malware. Their study focused on domain fluxing malware and relied on the binary extraction for DGA. Their approach is only effective for certain types of malware [27]. Besides blocking and extracting DGAs from normal domains, a further study has been explored based on the features of DGA domain names [28].

Since the DGA domain names are usually randomly generated, the lengths of DGA domains are very long. Such a feature can be used to detect DGA domains. That is, shorter DGA domain names are more difficult to be detected. This is because most normal domains are tend to be short. Ahluwalia *et al.* [29] proposed a detection model that can dynamically detect DGA domains. They apply information theoretic features based on a domain length threshold. Their approach can dynamically detect the DGA domains with any length. Many other studies have been done on DGA detection based on the DGA domain features.

Ma *et al.* [30] proposed a lightweight approach to detect DGA domains based on URLs using both lexical and host-based features. They consider the lexical features of the URL such as length, number of dots, and special characters in the URL path. Antonakakis *et al.* [31] proposed a novel detection system, called Pleiades. They extracted a number of statistical features related to the NXDOMAIN strings, including distribution of n-grams. Wang and Shirley [32] proposed using word segmentation to derive tokens from domain names to detect malicious domains. The proposed feature space includes the number of characters, digits, and hyphens. Similar to Ma *et al.* [30], McGrath and Gupta [33] also took a close look at phishing URLs and found that the phishing URLs and DGA domains had different characteristics when compared with normal domains and URLs. Therefore, they proposed a model for detecting DGA domains based on domain length

comparison and character frequencies of English language alphabets. The similar approach based on DGA features can be found in [18] and [34]–[36].

In order to classify DGA domain names, Schiavoni *et al.* [5] proposed a feasible approach for characterizing and clustering DGA-generated domains according to both linguistic and DNS features. In the study, they have assumed that DGA domains have groups of very significant characters from normal domains. By grouping domains according to their features, the authors applied a machine learning classifier to distinguish DGA domains from normal domains easily. Several machine-learning techniques have been studied to classify malicious codes. They include neural networks, support vector machines (SVM) and boosted classifiers [37]. There are also several studies aiming to predict DGA domain names from historical DGA domains [38]. Woodbridge *et al.* [39] used DNS queries to find the pattern of different families of DGAs. Their approach does not need a feature extraction step. Instead, it leverages long short-term memory (LSTM) networks for real-time DGA prediction. Their approach can be easily implemented by using the open source tools. Similar to Woodbridge *et al.* [39], Xu *et al.* [40] checked DNS similarity and pattern to predict future DGA domains. Their approach is effective for some DGAs. Recently, researchers have proposed deep learning techniques for detecting DGAs and learning features automatically, i.e., no effort from human is needed for feature analysis [41].

With the increase of computational power and data storage capability, big data has become a popular research topic [42]. Deep learning in neural networks is another popular research topic in machine learning field [43]. It not only achieves great successes in a broad area of applications such as computer vision [44] and speech recognition [45], but also is very powerful for processing large datasets [46]. Deep learning uses a machine learning technique to build deep architectures for classification through either supervised or unsupervised approaches [47]–[49]. In the Internet of Things (IoT) and networking area, enormous data can be collected over time. This will result in handling huge datasets that we have to process and analyze [50]. Therefore, applying deep learning in networking and IoT areas has become very popular. Qiu *et al.* [51] discussed deep learning techniques in processing big data and Perera *et al.* [52] have conducted a survey about the potentials of applying deep learning to IoT systems. Fadlullah *et al.* [53] have presented how deep learning can be applied to network traffic control systems.

Since DGA algorithms can generate large amount of DGA domains, a deep learning approach can also be used in detecting DGA domains. Yu *et al.* [54] has proposed an inline DGA detection using deep networks. They argued that most of the DGA detection methods are used in training on small datasets rather than large datasets collected from real traffic. They first collected the data from real DNS traffic by using simple filtering steps. Then, they built a convolutional neural network (CNN) model to train on the large amount of datasets.

In our previous paper [13], we proposed a machine learning framework that contains a blacklist and a two-level classification and clustering model to detect DGA domains. In this paper, we added a HMM model to further predict the DGA domains. We also applied a deep learning model for handling large datasets.

IV. DESIGN

The important components in this research are: (1) domains extracted from DGAs; (2) a machine learning framework that encompasses multiple feature extraction techniques and the models to classify the DGA domains from normal domains, cluster the DGA domains, and predict a DGA domain. (3) a deep learning model to handle large datasets. Multiple online sources from simple Google searching provide example codes for a DGA construction. However, a majority of these techniques are trivial and fundamental at best. Online threat intelligence feeds give an approach to examining current and live threats in real-world environment. This section describes the approach for data collection and proposes a machine learning framework for DGA malware analysis.

A. THREAT INTELLIGENCE FEED AND ONGOING THREAT DATA

DGAs are plentiful through multiple online examples that are found from Google searching and Github repositories. However, sophisticated threat actors purposely create tailored DGA to evaluate current detection systems. Using real-time active malicious domains derived from DGAs on the public Internet measures the accuracy of the proposed approach. Specifically, threat intelligence feeds collected from Bambenek Consulting [55] over a period of one year were obtained through daily manual querying demonstrated trends of ongoing threats. The structure of the data is presented in a CSV format of domain names, originating malware, and DGA membership with the daily file size of approximate 110MB, we have collected 64GB in total. Figure 2 demonstrates an example feed from the collected data.

```
sjthgkvsnpkfq.net,Domain used by Cryptolocker - Flashback DGA for 16 Jan 2018,2018-01-16
tfysaplgjhsnp.biz,Domain used by Cryptolocker - Flashback DGA for 16 Jan 2018,2018-01-16
ubvjhbndmsjoj.ru,Domain used by Cryptolocker - Flashback DGA for 16 Jan 2018,2018-01-16
vwbubgdqikrwr.org,Domain used by Cryptolocker - Flashback DGA for 16 Jan 2018,2018-01-16
wlxnxfvwqsnyu.co.uk,Domain used by Cryptolocker - Flashback DGA for 16 Jan 2018,2018-01-16
```

FIGURE 2. An example sample dataset from Bambenek Consulting gives domain names, malware origins, DGA schema, and date collected.

B. THE MACHINE LEARNING FRAMEWORK

We propose a machine learning framework that consists of three important steps, as shown in Figure 3. We first have the DNS queries with the payload as the input. Then, the DNS queries will be passed to our process step, which consists of 4 important components: (1) We first use a domain-request packet filter to get domain names and then store them in a dynamic blacklist. If the input is a known domain, we will

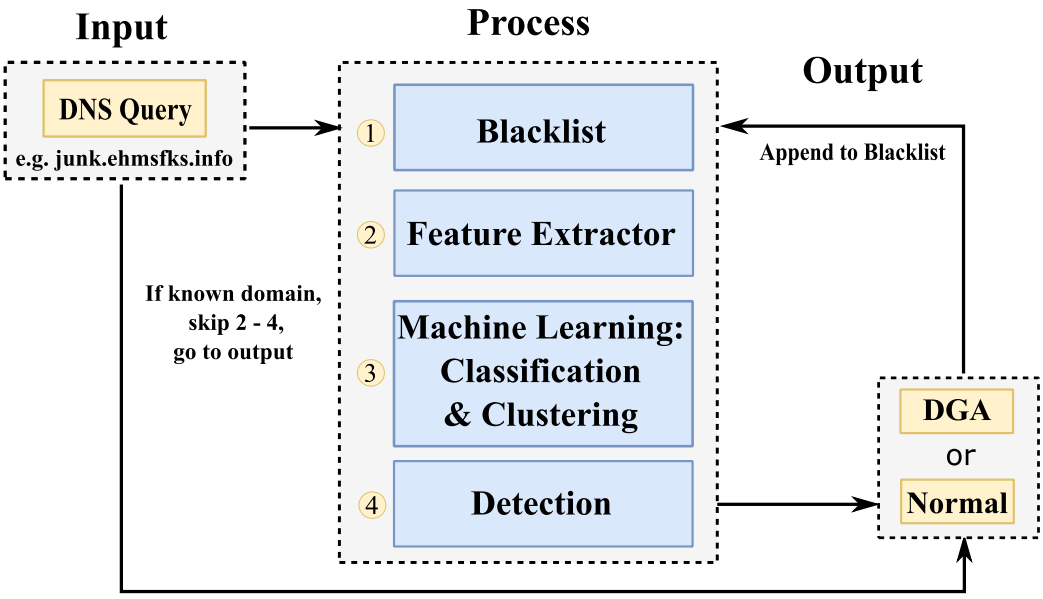


FIGURE 3. The proposed machine learning framework.

skip (2) - (4) and directly go to the output; otherwise, we will proceed to the next component. (2) Then, a feature extractor is used to extract domain features. (3) Next, we apply the first-level classification to distinguish DGA domains from non-DGA domains and the second-level clustering to group similar DGA domains. (4) Finally, we use a time-series model to predict the features of a domain. After the domain name goes through the process step, we will append this domain to the dynamic blacklist. The rest of this section discusses the four components of the process step in details.

1) DYNAMIC BLACKLIST

The domain names are the only information we need to perform classification and prediction in the following steps. We apply a domain-request packet filter to filter out the trivial information, which is useless in our experiment, collected from the raw data. After filtering, we obtain only domain names. In this process, we use the Gruber Regex pattern filter [56]. All the network traffic undergoes this filtering process. The filtered domain names are stored in the dynamic blacklist [57], which is initially empty and will be updated dynamically and then sent to the feature extractor in next step. The dynamic blacklist can help us to reduce unnecessary calculation, if a domain can be found in the dynamic blacklist, we can directly go to the output step.

2) FEATURE EXTRACTOR

The feature extractor is used to extract features from the domain names filtered in the first component. Each domain name is considered as a string. To efficiently classify domains, we use two types of features: linguistic features and DNS features. We start with the discussions of linguistic features and then the DNS features.

There are six linguistic features: Length, Meaningful Word Ratio, Percentage of Numerical Characters, Pronounceability Score, Percentage of the Length of the Longest Meaningful String (LMS), and Levenshtein Edit Distance. The detailed description and calculation of each linguistic feature are given as follow:

Length: We use $|d|$ to represent the length of a domain name.

Meaningful Word Ratio: This feature measures the ratio of meaningful words in a string (domain name). The ratio is calculated as follows:

$$f_1 = \sum_{i=1}^{n} \frac{|w_i|}{|d|} \tag{1}$$

where w_i is the i-th meaningful substring of this string, $|w_i|$ is the length of i-th meaningful substring. Since DGA domain names usually contain meaningless words; therefore, a small value of a ratio usually means that the domain could be a DGA domain name and a higher ratio implies a safer domain name. We strict the length of each meaningful substring $|w_i|$ in the string to be at least 4 letters because most legitimate domain names have meaningful substrings with more than 3 letters.

For example, for a domain name of *iylvword*, we have $f_1 = (|word|)/8 = 4/8 = 0.5$. If a domain name is *myproject*, we have $f_1 = (|my| + |project|)/9 = (2 + 7)/9 = 1$ because the domain is fully composed of two meaningful words.

Pronounceability Score: In the linguistic sense, the pronounceable words usually consists of many viable combinations of the phonemes. The pronounceability score characterizes the number of words that can be pronounced. The more pronounceable words, the higher the pronounceability score. Because DGA domains usually contain less viable combinations of phonemes, they usually tend to have a

TABLE 1. DGA classification features [13].

Features	Description	Feature Class	(+/-)
Meaningful words	Ratio of meaningful words	Linguistic	+
Prounceability	How easy can it be pounced	Linguistic	+
% of numerical characters	# of numbers	Linguistic	-
% of the length of the LMS	Ratio of LMS in the string	Linguistic	+
length of the Domain Name	How long is the Domain Name	Linguistic	-
Levenshtein edit distance	Min # of edits from last domain	Linguistic	+
Expiration date	If longer than 1 year	DNS	+
Creation date	If longer than 1 year	DNS	+
DNS record	If DNS record is documented	DNS	+
Distinct IP addresses	#. IP addresses related to this domain	DNS	+
Number of distinct countries	#. countries related this domain	DNS	+
IP shared by domains	#. domains are shared by the IP	DNS	-
Reverse DNS query results	If DN in top 3 reverse query results	DNS	+
Sub-domain	If domain is related to other sub-ones	DNS	+
Average TTL	DNS data time cached by DNS servers	DNS	+
SD of TTL	Distribution SD of TTL	DNS	-
% usage of the TTL ranges	Distribution range of TTL	DNS	+
# of distinct TTL values	Different value of TTL on server	DNS	-
# of TTL change	How frequently TTL changes	DNS	+
Client delete permission	If Client has delete permission	DNS	-
Client update permission	If Client has update permission	DNS	-
Client transfer permission	If Client has transfer permission	DNS	-
Server delete permission	If Server has delete permission	DNS	-
Server update permission	If Client has update permission	DNS	-
Server transfer permission	If Client has transfer permission	DNS	-
Registrar	The domain name registrar	DNS	+
Whois Guard	If use Whois Guard to protect privacy	DNS	-
IP address same subnet	If IP address is in the same subnet	DNS	-
Business name	If domain has a corporation name	DNS	+
Geography location	If domain provides address	DNS	+
Phone number	If domain provides a phone number	DNS	+
Local hosting	If use local host machine	DNS	+
Popularity	If on the top 10000 domain list	DNS	+

Note: DN - Domain name. TTL - Time-To-Live. SD - Standard deviation. All the features used in our model. (+/-): means that the feature is positively/negatively related to normal domains.

lower pronounceability score. Therefore, the pronounceability score can be a useful feature. To compute the pronounceability score of a string, we utilize an n-gram lookup table. We calculated the pronounceability score by extracting the n-grams score of a domain d. We choose the substring length $l \in 2, 3$ in our computation and count their occurrences in the English n-gram frequency text. For a domain d, the the pronounceability score is calculated as follow:

$$f_2 = \frac{\sum n - gram(d)}{|d| - n + 1} \tag{2}$$

where n is the length of the matching word in the n-gram list.

Percentage of Numerical Characters: This feature measures the percentage of numerical characters in a string. It can be simply calculated by:

$$f_3 = \frac{|n|}{|d|} \tag{3}$$

where $|n|$ is the number of numerical characters.

Percentage of the Length of LMS: This feature is to measure the length of the longest meaningful string in a domain name. The calculation can be written as:

$$f_4 = \frac{|l|}{|d|} \tag{4}$$

where $|l|$ is the length of the longest meaningful string.

Levenshtein Edit Distance: It measures the minimum number of single-character edits between a current domain and its previous domain in a stream of DNS queries received by the server. The Levenshtein distance is calculated based on a domain and its predecessor. For example, given two strings "test" and "task," the Levenshtein Edit Distance between them is 2 because the characters that need to be edited are e to a and t to k. Another example is that for "word" and "world", the Levenshtein Edit Distance is 1 because we just need to add a l after the r.

Besides linguistic features, we also use 27 DNS features shown in Table 1. Because DGA domains are generated very recently and live very shortly, they usually contain less information compared to normal domains. For example, DGA domains tend to have the creation dates within one year and their expiration dates are very soon.

3) TWO-LEVEL MODEL: CLASSIFICATION AND CLUSTERING

To better understand DGA domains, we propose a two-level machine learning model consisting of the first-level classification and the second-level clustering. In the first-level classification, we use machine learning classifiers to classify

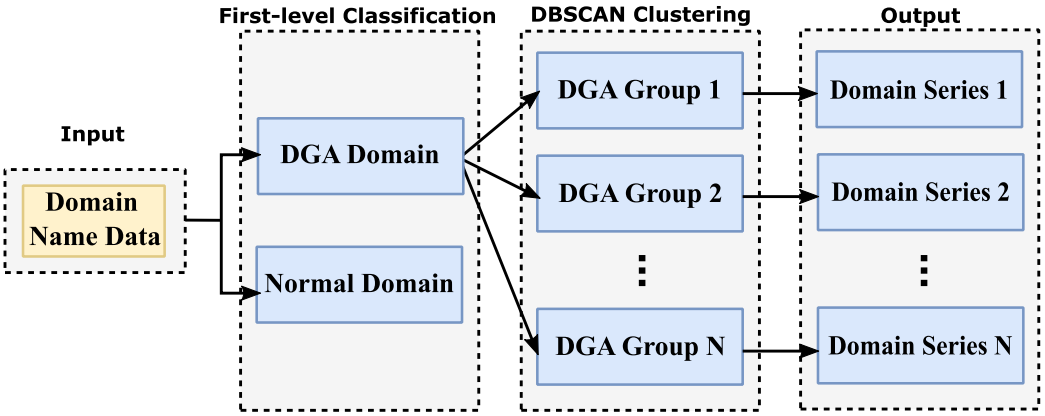


FIGURE 4. Two-level model of classification and clustering [13].

DGA domains and normal domains. Only the classified DGA domains will be sent to the second-level clustering. To divide the DGA domains into several groups based on their domain generation algorithm, we apply the DBSCAN clustering techniques. The workflow of the proposed two-level model is shown in Figure 4.

a: FIRST-LEVEL CLASSIFICATION

In the first-level classification, we use the features described above and test with seven different machine learning classifiers including Decision Tree-J48, Artificial Neural Network (ANN), Support Vector Machine (SVM), Logistic Regression, Naive Bayes (NB), Gradient Boosting Tree (GBT), and Random Forest (RF) to find the best classifier. Among those classifiers, we notice that J48 is the best to classify DGA domains (its detailed discussion is given in Section V), so J48 is chosen as the classifier in our first-level classification in this research.

b: SECOND-LEVEL CLUSTERING

In the second-level clustering, we apply the DBSCAN algorithm. Only the DGA domains obtained from the first-level classification will be used for clustering. In our DBSCAN algorithm, we use the features described above to calculate the domain distance and to group the domains that are generated by the same DGA together according to their domain feature difference. Let d_i and d_j be two different domain names, where $i \neq j$. We first set $i = 0$ representing the first domain and then calculate the overall distance between d_i and all other domains. The overall distance contains two parts: linguistic distance and DNS similarity.

The linguistic distance is computed based on the six linguistic features followed by the following equation:

$$D_l(d_i, d_j) = \sqrt{\sum_{k=1}^{6} distance_k(d_i, d_j)}, \tag{5}$$

where $distance_k(d_i, d_j)$ is the distance of each linguistic features between two domains d_i and d_j.

To get the DNS similarity, we first construct a weight matrix $\mathbf{M} \in \mathbb{R}^{K \times L}$, where K and L are the number of DNS features and that of linguistic features of all the DGA domains $\mathbb{D}$ that obtained from the first-level classification, respectively. The relationship between K and L is represented by a bipartite graph that is represented in $\mathbf{M}$, where each component $\mathbf{M}_{k,l}$ holds the weight of an edge (l, k). For each DNS record, weight $\mathbf{M}_{k,l}$ is computed by:

$$\mathbf{M}_{k,l} = \frac{1}{|\mathbb{D}(k)|}, \quad \text{for any } l = 1, \ldots, L, \tag{6}$$

where $|\mathbb{D}(k)|$ is the cardinality of the subset of domains that are pointed to the DNS record. We then use a matrix $\mathbf{S} \in \mathbb{R}^{L \times L}$ to store DNS similarity information, where for each component, $\mathbf{S}_{d_i,d_j}$ is the similarity value of domains d_i and d_j. Our intuition is that when two domains point to the same DNS record k, they should have high similarity. Therefore, we can calculate the similarity matrix based on the weight matrix $\mathbf{M}$. Let $\mathbf{N}$ be a column-normalized matrix of M. That is,

$$\mathbf{N}_{k,l} = \frac{\mathbf{M}_{k,l}}{\sum_{k=1}^{K} \mathbf{M}_{k,l}}, \quad \forall l = 1, 2, \ldots, L. \tag{7}$$

Furthermore, the final similarity matrix is calculated by:

$$S = N^T \odot N \in \mathbb{R}^{L \times L}. \tag{8}$$

The overall distance is a combination of the linguistic distance and DNS similarity and it is calculated by:

$$D(d_i, d_j) = S_{d_i,d_j} + log(\frac{1}{D_l(d_i, d_j)}) \tag{9}$$

After we have calculated the overall distance, we can get all points density-reachable from d_i based on the threshold distance, ϵ. If $D(d_i, d_j) > \epsilon$, we add those points d_j to a cluster C. The minimal cluster points, *MinPts*, is used to represent a core point. In our experiment, we set $MinPts = 3$ because it can achieve better performance of clustering. Let d_i be a core point. If the number of point in $C > MinPts$, then a cluster is formed. If d_i is a border point, implying that no points are density-reachable from d_i, then our DBSCAN model visits

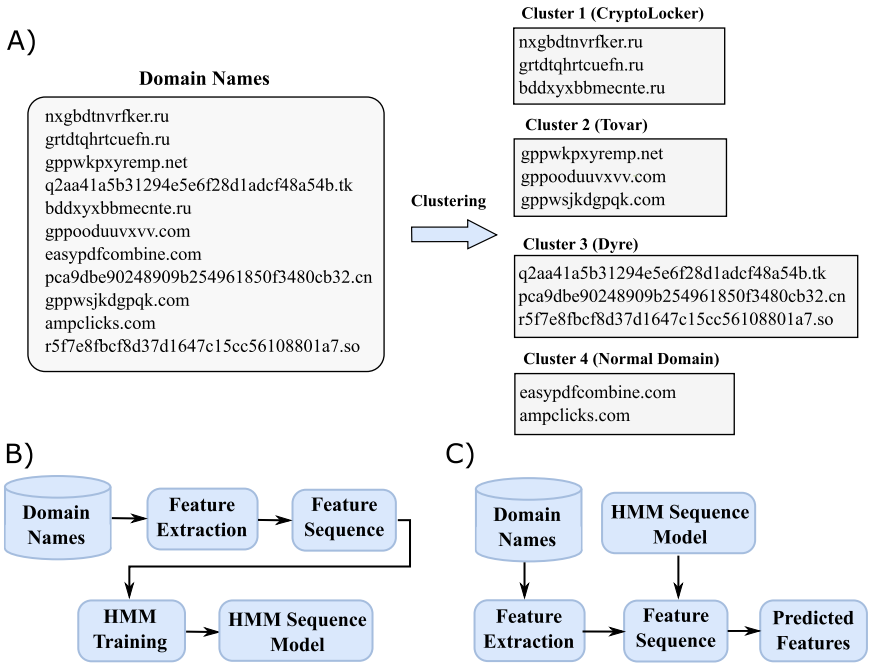


FIGURE 5. (A) A representative example of a clustering (B) HMM training procedure (C) Workflow of the HMM model prediction.

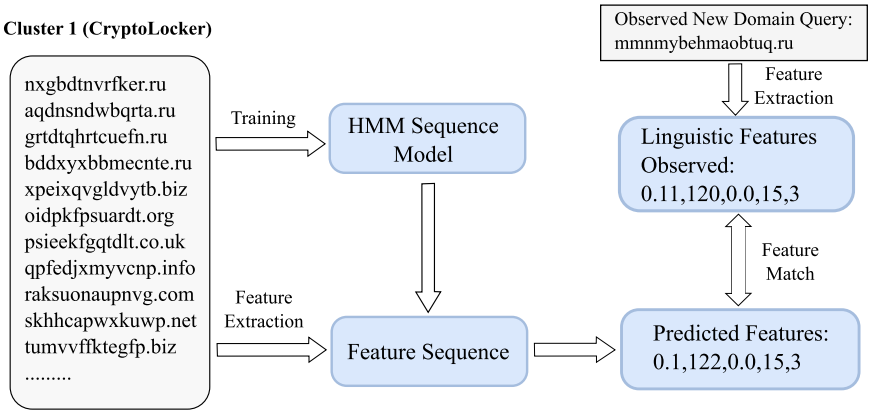


FIGURE 6. An example of the HMM model prediction.

the next domain. The above steps will be repeated until all of the domains have been processed.

4) A TIME-SERIES PREDICTOR

To analyze the clustering result, we build an HMM-based time-series prediction model to predict incoming DGA domain features. Figure 5 (A) shows an example dataset derived from clustering. We use every domain cluster to train a separate HMM model. We distinguish the model from training and prediction stages. Figure 5 (B) shows the HMM training step. Each HMM data record represents a series of domain observations. First, a sequence of domain names are processed by a feature extractor and each of these feature vectors is used as a training record. Then, similar sequences are clustered as a group of DGA domain names with certain outcomes. After the training process, if a sequence does not have an HMM sequence representation (or it is not presented in the training data but the test data), the HMM model then generates the future predicted results. Otherwise, we will use an existing HMM sequence representation. Figure 5 (C) shows the HMM prediction workflow. Once the model has been trained, a set of features is formed by a series of DGA domains. Then, we go to the prediction stage. In this stage, we produce a complete time-series list of domain features from a domain name to be synthesized. For the input of real-time domains, we compare the predicted features with the features extracted from the observed new domain query, as shown in Figure 6.

The detail of our HMM model is described as follows. We assume that each domain name of a DGA cluster at time t

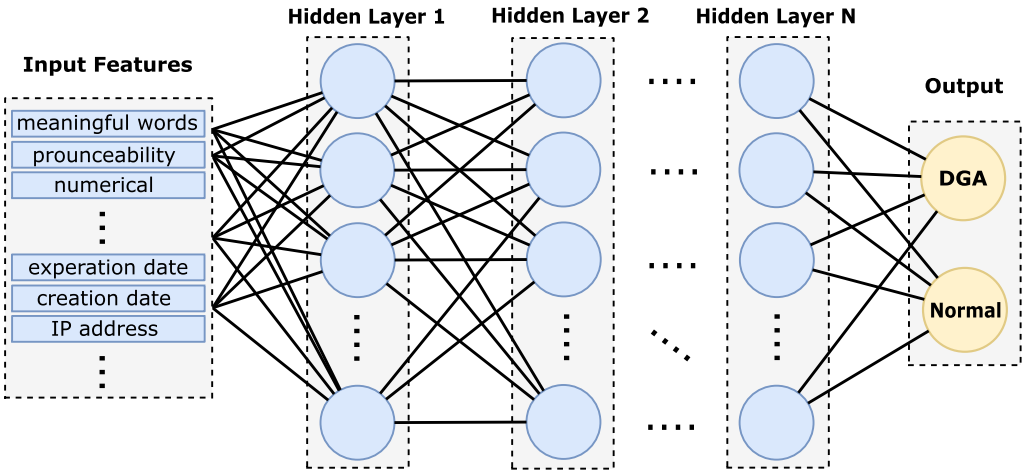


FIGURE 7. The proposed deep learning model.

is generated by an unknown DGA, S_t, which is *hidden* from the system. Moreover, the hidden state satisfies the n-th order *Markov property*, where S_t given $S_{t-1} \ldots S_{t-n}$ is independent of S_i for $i < t-n$. The n is the HMM sequence length used in our model. Thus, a sufficient length sequence encapsulates all we need to know about the history to predict future features. Thus, the joint distribution of domain features and observed features can be factored as shown here:

$$P(S_{1::T}, Y_{1::T}) = P(S_1)P(Y_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(Y_t|S_t)$$

Notation $S_{1::T}$ means $S_1, \ldots, S_T$. S is the hidden domain name state that is of a type of DGA. Y is an observed feature. Thus, we can infer the probability of S_t from the sequence of features $Y_{1::T}$ and previous state $S_{1::t-1}$, which is the foundation of the HMM model. *Length of the HMM model sequence:* The HMM predictor is affected by the sequence length of the given data.

C. DEEP LEARNING FOR CLASSIFICATION

A deep neural network (DNN) model is one of the most popular models in the deep learning area. DNNs have achieved a remarkable success in areas such as computer vision and natural language processing. It is also good for processing big datasets. In this section, we discuss the detailed DNN model that is used in our deep learning framework. DNNs can be viewed as a deeper version of artificial neural networks (ANNs) with many hidden layers, which include multiple nodes, also called neurons, in each hidden layer. The nodes in each layer are fully connected to the nodes in the next layer and each connected line has a weight. DNNs can model complex non-linear relationships by using activation functions. The optimization algorithms are used in DNNs to reduce loss. To process large dataset, we build a deep learning model to classify the DGA domains and normal domains and compare our deep learning model with our machine learning methods introduced in Section IV-B. Our deep learning model is shown in Figure 7.

1) ACTIVATION FUNCTION

A nonlinear function is applied to each hidden layer to introduce a nonlinearity. This nonlinear function is also called the activation function, which transforms the value of each node in the previous hidden layer before being passed onto the weighted sum of the next layer. The Rectified Linear Unit (ReLU) and the sigmoid function are the most common activation functions that are used in many DNNs. The sigmoid activation function converts the weighted sum into a value between 0 and 1 and it is written as below:

$$f_{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{10}$$

The ReLU activation function sets the value smaller than zero to be zero and the value greater than zero stays the same; it is expressed as below:

$$f_{ReLU}(x) = max\{0, x\} \tag{11}$$

The ReLU activation function often works a little better than a smooth function like the sigmoid function and it is easier to compute. Thus, we apply ReLU in our deep learning model.

2) LEARNING RATE

A learning rate is an important parameter in optimization algorithms. It is a hyper-parameter that tells us how much we need to update a vector parameter, θ, whose elements refer to weights in this paper. When we have a lower learning rate, the steps along the downward slope is slower. In this case, we will not miss any local minima, but it will take a long time to converge. The general equation is given as follows.

$$w_{new} = w_{old} - l \cdot g, \tag{12}$$

where w denotes the weight, l is the learning rate, and g represents the gradient of the loss function. Typically, the learning

rate is randomly given by a user. Getting a better learning rate is important. In the section V, we will analyze the results using different learning rates.

3) OPTIMIZATION ALGORITHMS

One of the important steps in the deep learning is to reduce a loss function and update those DNN model parameters that can be learned, such as weights and bias values. The optimization algorithms can help us minimize the loss function. Because our DNN model is used for classifying DGA domains and normal domains, the prediction input can be turned into a probability value between 0 and 1. Log loss can be used to measure the performance of our DNN model. The equation of calculating log loss is given as follows:

$$LogLoss = -(y\log(p) + (1-y)\log(1-p)), \tag{13}$$

where y is the labeled output. The output is either labeled as a DGA domain expressed as a value of 1, or a normal domain expressed as a value of 0. p is the predicted output probability between 0 and 1.

There are many optimization algorithms that can be used. In this paper, we introduce three optimization algorithms, Gradient Descent [58], Adagrad [59] and Adam [60], and make a comparison among them.

a: STOCHASTIC GRADIENT DESCENT (SGD)

It is one of the most popular algorithms to perform optimization in the DNNs. It is a stochastic variant of the gradient descent algorithm to minimize the loss function. SGD updates parameter θ, which is for each training instance x^i, and label y^i. The equation of updating the parameter is given as follows:

$$\theta = \theta - l \cdot g(\theta, x^i, y^i) \tag{14}$$

where l is the learning rate, $g(\theta, x^i, y^i)$ is the gradient of the log loss function for each training instance, x^i, and label, y^i.

b: ADAPTIVE GRADIENT ALGORITHM (ADAGARD)

It is an algorithm for gradient-based optimization. Unlike gradient decent, Adagard can adaptively update the learning rate. If its parameters are associated with frequent features, the learning rates will be updated to low learning rates and if the parameters are associated with infrequent features, then the learning rates will be updated to large learning rates. Therefore, it is more suitable for dealing with sparse data. At first, we update every parameter $\theta(i)$ using the same learning rate l. Then, at every time step t, we use a different learning rate for every parameter $\theta(i)$. The general equation is presented as follows:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{l}{\sqrt{G(t,i)+\epsilon}} g(t,i), \tag{15}$$

where $g(t, i)$ is the gradient of the loss function with respect to parameter $\theta(i)$ at time step t. l is a given learning rate. It is updated for every parameter $\theta(i)$ based on the past gradients $G(t, i)$.

c: ADAPTIVE MOMENT ESTIMATION (ADAM)

Similar to Adgard, Adam can also adaptively update the learning rates for every parameter. Adam exploits the idea of calculating the momentum. The general equation is written as follows:

$$\theta_{t+1} = \theta_t - \frac{l}{\sqrt{V(t)+\epsilon}} M(t), \tag{16}$$

where $V(t)$ is the first moment, the mean of the gradients of the loss function with respect to parameter θ at time step t. $M(t)$ is the second moment that is the variance of the gradients.

4) TRAINING AND VALIDATION

In our DNN model, we separate a training dataset from a validation dataset for overfitting prevention. We only train on the training dataset and test with the validation dataset. Other parameters used in the DNN model are periods, a batch size and steps. Periods control the granularity of displaying the results. The batch size is the number of examples for a single step. It is usually chosen randomly, but for SGD it is usually set to be 1. Steps is the total number of training iterations. Thus, the number of training example T_{period} in each period is calculated as follow:

$$T_{period} = \frac{B \cdot S}{P} \tag{17}$$

where B is the batch size, S is the steps and P denotes the periods.

V. EVALUATION

A. EXPERIMENTAL SETUP

Our machine learning methods and the DNN model are implemented using some open source tools, such as, Tensorflow. GENI is a Natinoal Science Foundation (NSF) funded heterogenous testbed solution. Leveraging high-performance nodes aids in the ability to process large volumes of real-time data feeds in a timely manner. The nodes selected for the evaluation consisting of systems running: Intel(R) Xeon(R) CPU E5-2450 @ 2.10GHz, 16 GB of hard drive space, and 1GB of memory, where the size can be manipulated based on reservation. Our DNN model is trained on GAIVI[1], a computer cluster with largescale parallel computing capabilities. Each node has the hardware: GeForce GTX TITAN X major: 5 minor: 2 memoryClockRate(GHz): 1.076.

To evaluate our model thoroughly, we use five datasets of DGA domain data: CryptoLocker, Tovar, Dyre, Nymaim, and Locky from the latest DGA-feed [61]–[63]. We collected the DGA domain names over a period of one year since2017. To provide a list of the normal control group domain names, we choose the top 1 million most popular Internet domains listed in domain punch [64]. 160,000 domain names were tested in our machine learning framework. We mix the control domain names and the DGA domains names with a 1:1 ratio

[1]GAIVI is funded in part by the National Science Foundation under Grant 1513126.

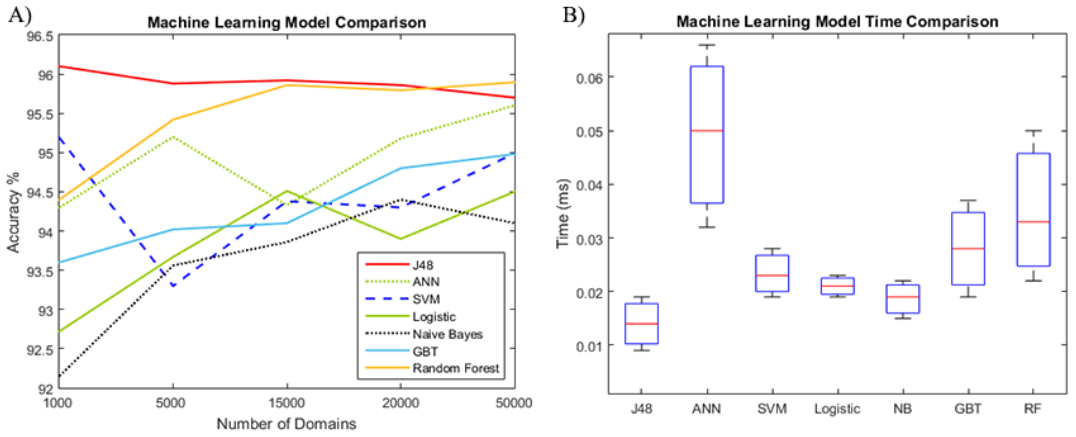


FIGURE 8. (A) Accuracy of different machine learning algorithms (B) Classification time of different machine learning algorithms.

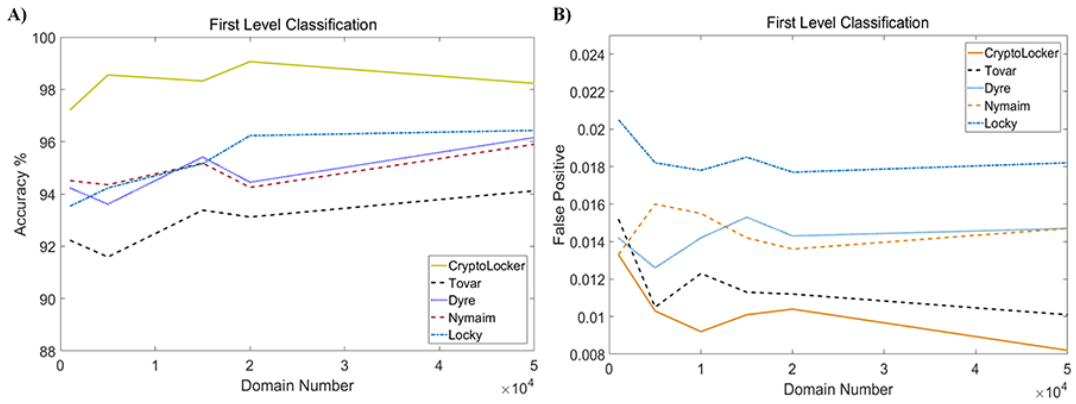


FIGURE 9. (A) J48 classification accuracy for each DGA domain with different data sizes (B) J48 classification false positive rate [13].

for the first-level classification. In the second-level clustering, we use classified DGA domain names from the first-level classification to cluster them into different groups of DGA domains. In the HMM prediction model, we use each group of clustering result as an input. Thus, we build one HMM prediction model for each clustering group of a DGA, respectively. To handle the large amount of data we collected over time, we build a DNN model for classification. We evaluate our DNN model and compare it with our first-level machine learning classification. In our DNN model, we have trained and tested on more than 1 million domain names.

B. EXPERIMENTAL RESULTS

1) THE PROPOSED MACHINE LEARNING FRAMEWORK

To find the best model for the first-level classification, we test seven different machine learning models, J48, ANN, SVM, Logistic Regression, Naive Bayes, Gradient Boosted Tree, and Random Forest. We perform the 10-fold cross-validation on these machine learning models. In each fold, 80% are used for training and 20% are used for validation. Figures 8 (A) and (B) show the performance of different algorithms on the classification of the DGA domains. We find that J48 has the highest average accuracy, 95.89%, compared to other machine learning algorithms. We can see that Random Forest has similar accuracy with J48. The average accuracy of Random Forest is 95.47%. Figure 8 (B) also shows that J48 is the fastest one with an average of 0.0144 ms to classify the domain names. To see the accuracy of J48 associated with scalability, we test five groups of samples for each DGA generated domain with a total number of 1000, 5000, 15000, 20000, 50000 domain names. We find that J48 performs the best for CryptoLocker domain names.

Figure 9 (A) shows that the average accuracy of Cryptolocker is 95.89% and its highest accuracy reaches 98.27%, while other DGA domains have accuracies ranging from 92% to 95%. As shown in Figure 9 (B), we also compute the false positive rate because it is also an important metric, where the lower the better. The false positive rates are 0.010, 0.012, 0.014, 0.015, 0.018 for CryptoLocker, Tovar, Dyre, Nymaim, and Locky, respectively.

Figure 10 (A) shows how the second-level clustering algorithm performs on different DGAs. When we use both linguistic distance and DNS similarity as the overall distance, its average accuracy is 87.64%, whereas if we only use DNS similarity as the overall distance, the average accuracy is 89.02%. This is because most of DGAs have very similar

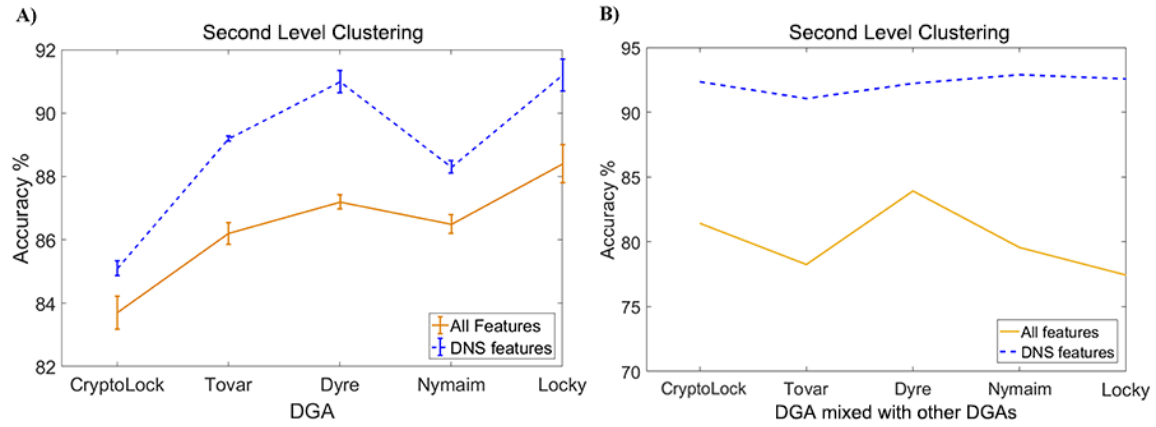


FIGURE 10. (A) Clustering accuracy for each DGA (B) Clustering accuracy for each two DGA groups [13].

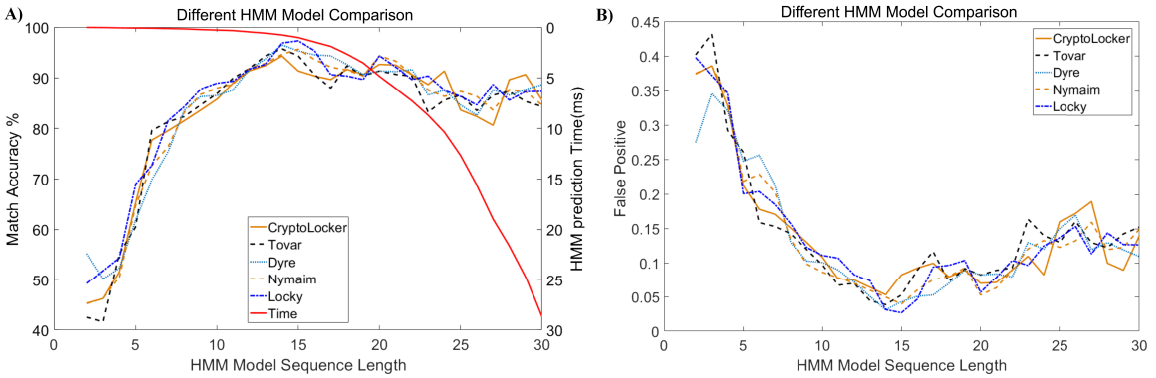


FIGURE 11. (A) The match accuracy for HMM models with different sequence lengths. (B) The false positive rate for HMM models with different sequence lengths.

string compositions and lengths. These features can not help the clustering algorithm to identify similar DGA domains from each other. Furthermore, we test the accuracy of clustering when more groups are mixed together.

As shown in Figure 10 (B), we test all the two group combinations for all the five DGAs. When we mix Cryptolocker with other DGAs, the average accuracy for clustering is 81.43% for all the features. However, when we use only DNS features as the DBSCAN distance, its accuracy increases to 92.45%, which means that most Cryptolocker domains are clustered into one group. Similarly, when testing other groups, we find that the accuracies of clustering are 91.05%, 92.22%, 92.89%, and 92.57% for ovar, Dyre, Nymaim, and Locky, respectively. The result demonstrates that the clustering model is efficient to group the same DGA domains into one group for a further time series model. Our clustering model can provide high-quality training data for the HMM model. The higher accuracy of clustering can make the prediction of future domain names more accurately.

Furthermore, we have extensively studied how the time series model performs when doing a feature prediction. We train an HMM sequence model for each cluster. To find the best HMM model for each dataset, we test match accuracy and false positive for a model with the HMM sequence lengths ranging from 2 to 30. Figure 11 (A) shows that the peak accuracy is at sequence length ranging from 14 to 15, where the average accuracy is 95.21% among them. Moreover, from the average prediction time analysis for each sequence model shown in Figure 11 (A), we see a dramatic increase on prediction time when the sequence length is greater than 15. It takes only an average of 1.02 ms to predict a DGA domain when the model length is 15. But, when the model length is 20, the time will increase to 4.85 ms, which is a 3.75 times increase. On the other hand, the false positive rate for these models also reaches the lowest at either lengths 14 or 15 in Figure 11 (B). The average false positive rate we observed is 0.045 at length 15. The result demonstrates that the HMM sequence model with length 15 has the best accuracy and a very fast running time. The performance of an HMM sequence model grants a fast response and the capability to block DGA domains before it starts with a DNS query.

2) DEEP LEARNING ENHANCEMENT

In the DNN model, the parameters include number of neurons and hidden layers. We test with different values of neurons and hidden layers and manually change them. Then, we choose the best one based on the results of accuracy,

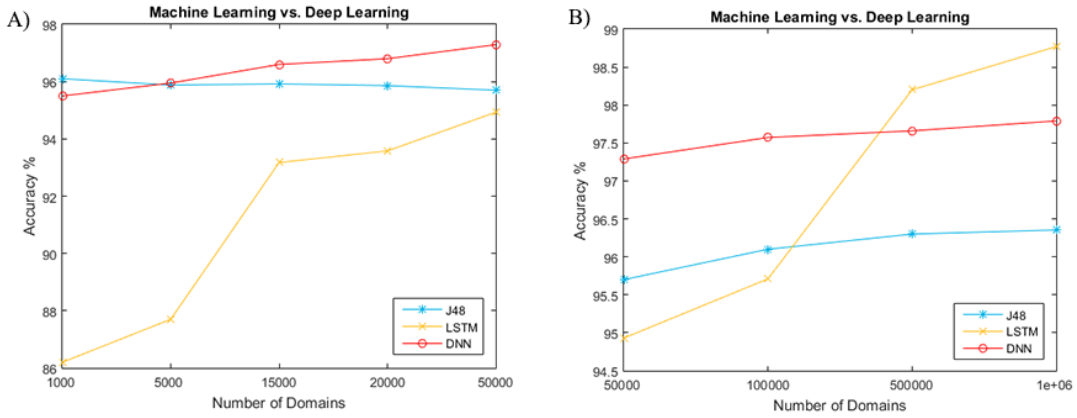


FIGURE 12. (A) J48, LSTM and DNN accuracy comparison with the number of domains ranging from 1000 to 50000. (B) J48, LSTM and DNN accuracy comparison with a large number of domains ranging from 50k to 1M.

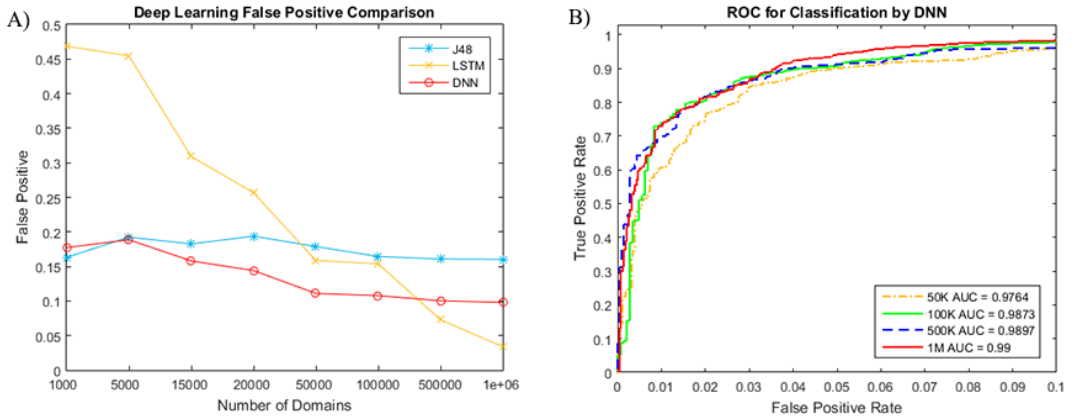


FIGURE 13. (A) Comparison of false positives among J48, LSTM and DNN. (B) ROC curves among the different number of domains, 50K, 100K, 500K, and 1M.

log loss, and AUC. We first compare our DNN classification model with the first-level classification in our machine learning framework and the LSTM model described in [39]. We use the LSTM model to train and test with our collected dataset. Since we find that J48 has the best accuracy among seven machine learning models, we will only compare J48 with our DNN model. Figure 12 shows the accuracy comparison results among J48, LSTM model and the proposed DNN model with a different number of domain names. Figure 12 (A) shows the accuracy testing with the total number of 1000, 5000, 15000, 20000, 50000 domain names. The average accuracy for J48 is 95.89%, the average accuracy for the LSTM is 91.12%, and the average accuracy for the DNN model is 96.43%. We can see that when the number of domain names is 1000, J48 has better accuracy than the DNN model, and at 5000 domain names, their accuracies are similar. Then, as the number of domain names increases, the accuracies of the DNN model has outdistanced the J48 model. The accuracy of the LSTM model is lower than J48 and the DNN model, but it increases very fast. In the Figure 12 (B), we continue to increase the number of domain names and the total number of domain names are 50k, 100k, 500k, and 1M. While the highest accuracy for J48 is 96.35% at 1M domain names, the highest accuracy for the LSTM model is 98.77% at 1M domain names, and the highest accuracy for the DNN model is 97.79%, also at 1M domain names. We can see from the figure, as the number of domain names continues to increase, the increase rate of accuracies starts to slow down. The LSTM model has better accuracy than the DNN model for the very large number of domain names, such as 500k and 1M domain names. However, the average accuracy of LSTM is 96.9%, which is lower than the average accuracy of the DNN model, 97.58%.

We also compare the false positive among J48, LSTM model and the DNN model since it is a common measurement in machine learning. Figure 13 (A) shows the false positives of J48, LSTM model and the DNN model testing with the total number of 1K, 5K, 15K, 20K, 50K, 100K, 500K, and 1M domain names, respectively. The false positives are normalized values among all positives. The smaller the value, the better the performance. We can see that the false positives of the LSTM model drop quickly as the number of domain

TABLE 2. Precision, recall, AUC and time comparison.

	DNN				LSTM			
No. of Domains	Precision	Recall	AUC	Time(s)	Precision	Recall	AUC	Time(s)
1K	0.7856	0.9926	0.9555	90.86	0.1333	0.9891	0.910	44.45
5K	0.7691	0.9954	0.9667	263.57	0.1684	0.9863	0.9520	120.24
15K	0.8136	0.9914	0.9798	202.47	0.5570	0.9827	0.9720	362.20
20K	0.8330	0.9902	0.9819	259.12	0.6610	0.9724	0.9710	442.52
50K	0.8764	0.9882	0.9864	2376.65	0.8167	0.9665	0.9770	1217.59
100K	0.8801	0.9907	0.9873	11496.49	0.8216	0.9770	0.9830	2499.69
500K	0.8894	0.9903	0.9897	18185.41	0.9218	0.9915	0.9970	12161.82
1M	0.8924	0.9914	0.9900	23154.35	0.9652	0.9913	0.9990	22184.56

names increases, but for the number of domain names smaller than 50K, it has a worse false positive than the DNN model and J48. From the evaluation results of J48, the LSTM model and the DNN model, we have observed that the DNN model has better performance in handling all sizes of datasets than J48 and LSTM.

Besides the comparison of the accuracy and the false positive, we also compare the Precision, Recall, and the area under the Receiver Operating Characteristic (ROC) curve (AUC) between the LSTM model and the DNN model. Precision measures the fraction of true positive instances among all predicted positive instances. Recall measures the fraction that are detected as positive among all the actual positive instances. The ROC curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The AUC measures the area under the ROC curve. The larger the AUC, the better the performance. Table 2 shows the comparison of Precision, Recall, AUC and Time used in seconds between the LSTM model and the DNN model.

To overcome overfitting, we separate our training data from validation data and calculate the log loss to measure the performance. The goal is to keep the validation log losses to match the training log losses as far as possible. Figure 14 shows the training and validation log losses over 10 periods. We choose the number of 50000 domain names as an example. We can see that the validation log losses are very similar

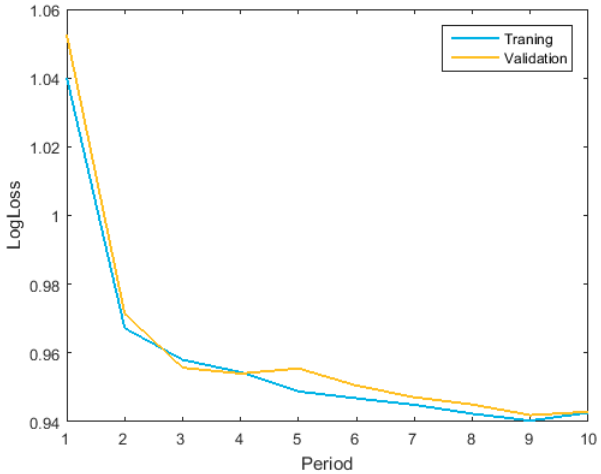


FIGURE 14. Training and validation log loss over periods.

to the training log losses and keep the peace with the training log losses as the period goes on.

To better build and evaluate our DNN model alone, we choose the number of 50K, 100K, 500K, and 1M domains to measure the ROC, AUC, learning rate, and optimization algorithms. Figure 13 (B) shows the ROC curves of different number of domains. The AUCs for the number of 50K, 100K, 500K, and 1M domains are 0.9764239, 0.9873341, 0.9897, and 0.99, receptively. The 1M domains result has the best AUC and the ROC curve is plotted in the red line. The ROC of 50K domains is plotted in the yellow line and it has the smallest AUC among them.

Figure 15 shows the evaluation of different learning rates. We choose the learning rate of 0.00001, 0.0001, 0.001, 0.01, 0.05, 0.1, and 0.5 to make a comparison among the four different number of domains, 50K, 100K, 500K, and 1M. We evaluate their accuracies and log losses. Figure 15 (A) shows the accuracy comparison of different learning rates. The purple line, yellow line, blue line and red line represent the number of 50K, 100K, 500K, and 1M domains, respectively. As the learning rate increases, the accuracy also increases generally. At the learning rates of 0.05 and 0.1, the average accuracies are higher than the average accuracies at other learning rates. Then, the accuracy decreases at the learning rate of 0.5. The average accuracies at the learning rates of 0.00001, 0.0001, 0.001, 0.01, 0.05, 0.1, and 0.5 are 91.95%, 95.44%, 96.77%, 97.25%, 97.58%, 97.54%, and 97.15%, respectively. The average accuracy at the learning rate of 0.05 is the highest, which is slightly higher than the average accuracy at 0.05. Figure 15 (B) shows the log loss comparison of different learning rates. The average log losses at the learning rate of 0.00001, 0.0001, 0.001, 0.01, 0.05, 0.1, and 0.5 are 2.75182, 1.54855, 1.12041, 0.94418, 0.83415, 0.84775, and 0.99205, receptively. The learning rate of 0.05 has the lowest log loss and the best accuracy. Therefore, in our DNN model, we choose 0.05 as the learning rate for all our other experiments.

In the Figure 16, we show the results of using different optimization algorithms, Gradient Descent, Adam, and Adagrad. Figure 16 (A) reports the accuracy comparison among three optimization algorithms. We also test with four different number of domains, 50K, 100K, 500K, and 1M. The average accuracies for Gradient Descent, Adam, and Adagrad are 96.997%, 97.079%, 97.578%, respectively. Among them, the Adagrad optimization algorithm has the

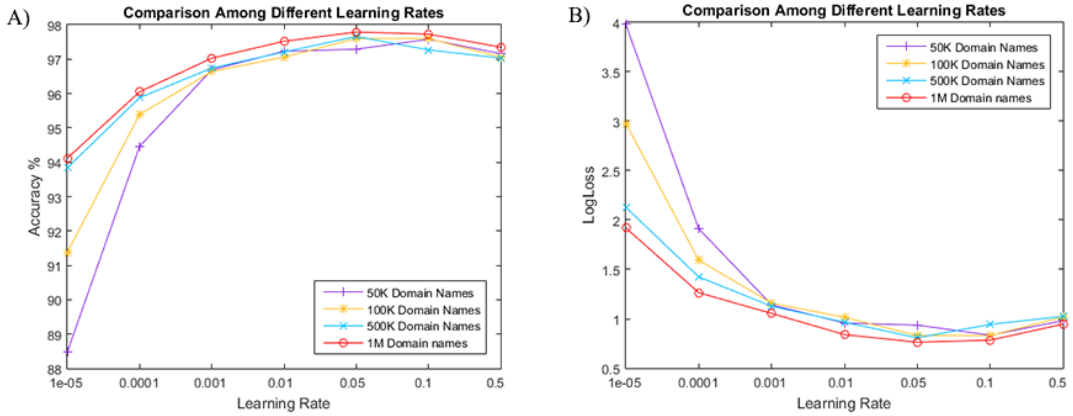


FIGURE 15. (A) Comparison of accuracies with different learning rates, 0.00001, 0.0001, 0.001,0.01,0.05, 0.1 and 0.5. (B) Comparison of log losses with different learning rate, 0.00001, 0.0001, 0.001,0.01,0.05, 0.1 and 0.5.

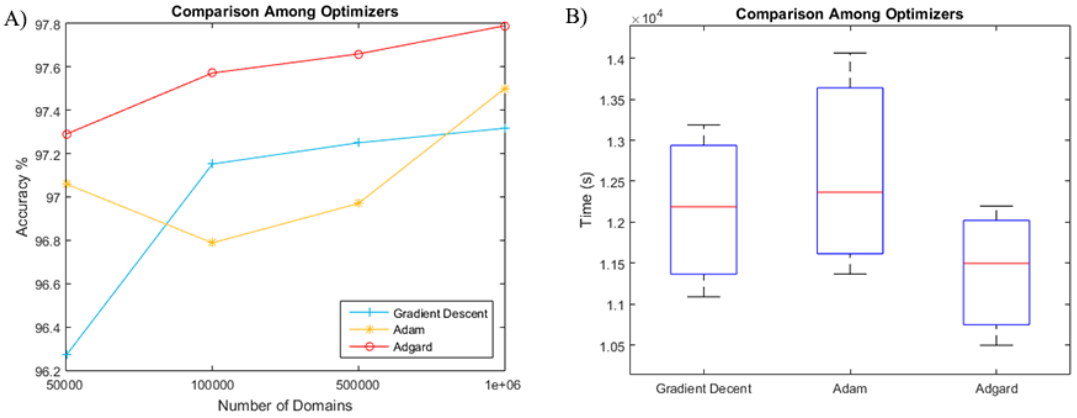


FIGURE 16. (A) Accuracy comparison among different optimization algorithms, Gradient Descent, Adam, and Adagrad. (B) Time comparison among these different optimization algorithms, where Adagrad has the least time used.

best accuracy. Figure 16 (B) shows the time used for each optimization algorithm. The average time are 12188.16041, 12364.92334, and 11496.49419 seconds, respectively, for building the DNN model. Adagrad use the least time in convergence, which is suitable for our DNN model. The average time for predicting one domain name is around 27.87 ms.

C. DISCUSSIONS

As seen in our experimental evaluation, the proposed machine learning framework has demonstrated the efficient way to predict a future DGA domain name. We have evaluated the proposed machine learning framework with the most latest DGA domain names from DGA-feed to cluster and predict DGA domains from these real-world data. Our evaluation has shown that with 33 features we proposed in our model, the J48 classification algorithm performed the most effectively and efficiently in comparison to ANN, SVM, Logistic, and Naive Bayes in terms of the minimal classification timeof 0.0144 ms, and the highest accuracy of 95.89%. We have also tested the clustering accuracy. Our result has shown that the DBSCAN clustering model we used can provide highly accurate clustered domains for a further time-series model with accuracy of 92.45%. We have noticed that the best accuracy we get from clustering is the one where only DNS query features are used. The experimental results have proved that a cluster of DGA domains usually points to several specific server IPs. DNS information of these domains are very similar and therefore clustering them with only DNS features is very accurate. We have further evaluated different HMM models in terms of accuracy and performance. We have found that the HMM model performs the best at length 15 with a fast running time of 1.02ms and a high match accuracy of 95.21%. We then evaluate our DNN model and compare it with the J48 classification algorithm. We find that the DNN model is better for classifying large datasets. The average accuracy of the DNN model is 96.43% and the highest accuracy is 97.79% testing with 1M domain names. We have also evaluated the different learning rates in our DNN model. We have noticed that the best learning rate for the DNN model is 0.05. The accuracy of DNN model increases as the learning rate increases. Then, after the learning rate of 0.1, the accuracy starts to decrease. This is because the learning rate is used to control the steps to update the weights; if the learning rate is too large or

too small, it will result in either missing the local minima or converging too slow. Therefore, choosing a proper learning rate is very important in the DNN model. At last, we compare the optimization algorithms used in the DNN model. We have found that the Adagrad optimization algorithm has the best performance compared to Adam and gradient descent. Adagrad converges fast because it is good for sparse data. In our dataset, some DNS features of the DGA domains are all zeros, so Adagrad is suitable in our DNN model.

VI. CONCLUSIONS AND FUTURE WORK

Detecting DGAs is a grand challenge in security areas. Blacklisting is good for handling static methods. However, DGAs are usually used by an attacker to communicate with variety of servers. They are dynamic, so simply using the blacklisting is not sufficient for detecting a DGA. In this research, we have proposed the machine learning framework with the development of a deep learning model to handle DGA threats. The proposed machine learning framework consists of a dynamic blacklist, a feature extractor, a two-level machine learning model for classification and clustering, and a prediction model. We have collected a real-time threat intelligence feed over a one-year period where all domains live threats on the Internet.

As the size of the data we collected becomes larger and larger, we have built a deep learning model to perform the classification, which has a better performance than the machine learning algorithms. Based on our extensive experiments on the real-world feed, we have shown that the proposed framework can effectively extract domain name features as well as classify, cluster and detect domain names. We have further used our DNN model to improve our classification.

In the future, we will further explore deep learning algorithms for domain name clustering and prediction for this research and evaluate them on a real-world testbed such as GENI [65]–[67].

ACKNOWLEDGMENT

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of NSF, FC2, and USF.

REFERENCES

[1] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Berlin, Germany: Springer, 2008, pp. 108–125.

[2] T. Chin, K. Xiong, and M. Rahouti, "SDN-based kernel modular countermeasure for intrusion detection," in *Proc. 13rd EAI Int. Conf. Secur. Privacy Commun. Netw.* Cham, Switzerland: Springer, 2017, pp. 270–290.

[3] U. Ghosh, P. Chatterjee, D. Tosh, S. Shetty, K. Xiong, and C. Kamhoua, "An SDN based framework for guaranteeing security and performance in information-centric cloud networks," in *Proc. 11th IEEE Int. Conf. Cloud Comput. (IEEE Cloud)*, Jun. 2017, pp. 749–752.

[4] C. Khancome, V. Boonjing, and P. Chanvarasuth, "A two-hashing table multiple string pattern matching algorithm," in *Proc. IEEE 10th Int. Conf. Inf. Technol., New Generat. (ITNG)*, Apr. 2013, pp. 696–701.

[5] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-based botnet tracking and intelligence," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2014, pp. 192–211.

[6] A. K. Sood and S. Zeadally, "A taxonomy of domain-generation algorithms," *IEEE Secur. Privacy*, vol. 14, no. 4, pp. 46–53, Jul./Aug. 2016.

[7] K. Xiong, "Multiple priority customer service guarantees in cluster computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, May 2009, pp. 1–12.

[8] K. Xiong, *Resource Optimization and Security for Cloud Services*. Hoboken, NJ, USA: Wiley, 2014.

[9] K. Xiong. (2008). *Resource Optimization and Security for Distributed Computing*. [Online]. Available: https://repository.lib.ncsu.edu/handle/1840.16/3581

[10] M. Berman *et al.*, "GENI: A federated testbed for innovative network experiments," *Comput. Netw.*, vol. 61, pp. 5–23, Mar. 2014.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining (KDD)*. Portland, OR, USA: AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[12] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 19-1–19-21, Jul. 2017, doi: 10.1145/3068335.

[13] T. Chin, K. Xiong, C. Hu, and Y. Li, "A machine learning framework for studying domain generation algorithm (DGA)-based malware," in *Proc. SecureComm*, 2018, pp. 433–448.

[14] K. Xiong and X. Chen, "Ensuring cloud service guarantees via service level agreement (SLA)-based resource allocation," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst. Workshops, ICDCS Workshops*, Jun./Jul. 2015, pp. 35–41.

[15] T. Chin and K. Xiong, "Dynamic generation containment systems (DGCS): A moving target defense approach," in *Proc. 3rd Int. Workshop Emerg. Ideas Trends Eng. Cyber-Phys. Syst. (EITEC)*, Apr. 2016, pp. 11–16, doi: 10.1109/EITEC.2016.7503690.

[16] K. Sornalakshmi, "Detection of DoS attack and zero day threat with SIEM," in *Proc. IEEE Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2017, pp. 1–7.

[17] S. Yadav and A. L. N. Reddy, "Winning with DNS failures: Strategies for faster botnet detection," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Berlin, Germany: Springer, 2011, pp. 446–459.

[18] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with DNS traffic analysis," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1663–1677, Oct. 2012.

[19] F. Guo, P. Ferrie, and T.-C. Chiueh, "A study of the packer problem and its solutions," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2008, pp. 98–115.

[20] T. Holz *et al.*, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proc. LEET*, vol. 8, no. 1, 2008, pp. 1–9.

[21] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Secur. Privacy*, vol. 9, no. 3, pp. 49–51, May/Jun. 2011.

[22] J. Stewart, "Inside the storm: Protocols and encryption of the storm botnet," in *Proc. Black Hat Tech. Secur. Conf.*, New York, NY, USA, 2009.

[23] H. S. Phillip Porras and V. Yegneswaran, "Conficker C P2P protocol and implementation,"2009.

[24] B. Stone-Gross *et al.*, "Your botnet is my botnet: analysis of a botnet takeover," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 635–647.

[25] L. Zhang, S. Yu, D. Wu, and P. Watters, "A survey on latest botnet attack and defense," in *Proc. IEEE 10th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Nov. 2011, pp. 53–60.

[26] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla, "Automatic extraction of domain name generation algorithms from current malware," in *Proc. NATO Symp. IST Inf. Assurance Cyber Defense*, Koblenz, Germany, 2012, pp. 1–13.

[27] 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, pp. 59-1–59-39, Dec. 2016.

[28] M. Mowbray and J. Hagen, "Finding domain-generation algorithms by looking at length distribution," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Nov. 2014, pp. 395–400.

[29] A. Ahluwalia, I. Traore, K. Ganame, and N. Agarwal, "Detecting broad length algorithmically generated domains," in *Proc. Int. Conf. Intell., Secure, Dependable Syst. Distrib. Cloud Environ.* Cham, Switzerland: Springer, 2017, pp. 19–34.

[30] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 1245–1254.

[31] M. Antonakakis *et al.*, "From throw-away traffic to Bots: Detecting the rise of DGA-based malware," in *Proc. USENIX Secur. Symp.*, vol. 12, 2012, pp. 24–24.

[32] W. Wang and K. Shirley. (2015). "Breaking bad: Detecting malicious domains using word segmentation." [Online]. Available: https://arxiv.org/abs/1506.04111

[33] D. K. McGrath and M. Gupta, "Behind phishing: An examination of phisher modi operandi," in *Proc. LEET*, vol. 8, 2008, p. 4.

[34] Y. He, Z. Zhong, S. Krasser, and Y. Tang, "Mining DNS for malicious domain registrations," in *Proc. IEEE 6th Int. Conf. Collaborative Comput., Netw., Appl. Worksharing (CollaborateCom)*, Oct. 2010, pp. 1–6.

[35] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding malicious domains using passive DNS analysis," in *Proc. NDSS*, 2011, pp. 1–17.

[36] S. Srinivasan, S. Bhattacharya, and R. Chakraborty, "Segmenting Web-domains and hashtags using length specific models," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 1113–1122.

[37] A. Shabtai, R. Moskovitch, Y. Elovic, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Inf. Secur. Tech. Rep.*, vol. 14, no. 1, pp. 16–29, 2009.

[38] R. Sharifnya and M. Abadi, "A novel reputation system to detect DGA-based botnets," in *Proc. IEEE 3rd Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct./Nov. 2013, pp. 417–423.

[39] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. (2016). "Predicting domain generation algorithms with long short-term memory networks." [Online]. Available: https://arxiv.org/abs/1611.00791

[40] W. Xu, K. Sanders, and Y. Zhang, "We know it before you do: Predicting malicious domains," in *Proc. Virus Bull. Conf.*, 2014, pp. 1–5.

[41] J. Saxe and K. Berlin. (2017). "eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys." [Online]. Available: https://arxiv.org/abs/1702.08568

[42] J. Chen *et al.*, "Big data challenge: A data management perspective," *Frontiers Comput. Sci.*, vol. 7, no. 2, pp. 157–164, 2013.

[43] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[45] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[46] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A survey on deep learning in big data," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) Embedded Ubiquitous Comput. (EUC)*, vol. 2, Jul. 2017, pp. 173–180.

[47] X.-W. Chen and X. Lin "Big data deep learning: Challenges and perspectives," *IEEE Access*, vol. 2, pp. 514–525, 2014.

[48] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 513–520.

[49] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *Proc. IEEE Int. Conf. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 4959–4962.

[50] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 1st Quart., 2018.

[51] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP J. Adv. Signal Process.*, vol. 2016, no. 1, p. 67, 2016.

[52] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.

[53] Z. M. Fadlullah *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, 1st Quart., 2017.

[54] B. Yu, D. L. Gray, J. Pan, M. De Cock, and A. C. A. Nascimento, "Inline DGA detection with deep networks," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2017, pp. 683–692.

[55] Bambenek. *OSINT Feeds From Bambenek Consulting*. Accessed: Jan. 27, 2019. [Online]. Available: http://osint.bambenekconsulting.com/feeds/

[56] L. Yang, R. Karim, V. Ganapathy, and R. Smith, "Fast, memory-efficient regular expression matching with NFA-OBDDs," *Comput. Netw.*, vol. 55, no. 15, pp. 3376–3393, 2011.

[57] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2014, pp. 1–21.

[58] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*. Cham, Switzerland: Springer, 2010, pp. 177–186.

[59] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Feb. 2011.

[60] D. P. Kingma and J. Ba. (2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

[61] JBT Organization. (2017). *Domain Feed of Known DGA Domains*. [Online]. Available: http://osint.bambenekconsulting.com/feeds/

[62] K. Jarvis. *CryptoLocker Ransomware*. Accessed: Jan. 27, 2019. [Online]. Available: https://www.secureworks.com/research/cryptolocker-ransomware

[63] Pchaigno. *A Collection of Known Domain Generation Algorithms*. [Online]. Available: https://github.com/pchaigno/dga-collection

[64] S Technologies. (2016). *Top Million Websites & TLDs*. [Online]. Available: https://domainpunch.com/tlds/topm.php

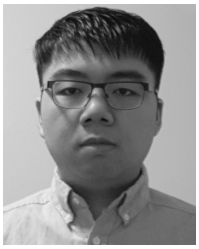
[65] T. Chin, X. Mountrouidou, X. Li, and K. Xiong, "An SDN-supported collaborative approach for DDoS flooding detection and containment," in *Proc. Mil. Commun. Conf.*, 2015, pp. 659–664.

[66] S. R. Lenkala, S. Shetty, and K. Xiong, "Security risk assessment of cloud carrier," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2013, pp. 442–449.

[67] K. Xiong and H. Perros, "SLA-based service composition in enterprise computing," in *Proc. 16th Int. Workshop Qual. Service (IWQoS)*, 2008, pp. 30–39.

YI LI received the B.S. and M.S. degrees in computer science from Henan Polytechnic University, China, in 2012 and 2015, respectively. She is currently pursuing the Ph.D. degree in computer science with the University of South Florida. Her current research interests include computer networking, software-defined networking, user behavior analysis in social media, machine learning, and deep learning.

KAIQI XIONG received the Ph.D. degree in computer science from North Carolina State University. Before returning to academia, he was with IT industry for several years. He is currently an Associate Professor with the University of South Florida, affiliated with the Florida Center for Cybersecurity, the Department of Mathematics and Statistics, and the Department of Electrical Engineering. His work was supported by the National Science Foundation (NSF), NSF/BBN, the Air Force Research Laboratory, Amazon AWS, Florida Center for Cybersecurity, and Office of Naval Research. His research interests include security, networking, and data analytics with applications cyber-physical systems, cloud computing, sensor networks, and the Internet of Things. He received the Best Demo Award at the 22nd GENI Engineering Conference (GEC22) and the U.S. Ignite Application Summit with his team in 2015 as well as the Best Paper Award at several conferences, such as the 2018 IEEE Power and Energy Society General Conference.



TOMMY CHIN received the B.S. degree in applied networking and system administration from the Rochester Institute of Technology, in 2013, where he is currently pursuing the M.S. degree in computing security. He holds numerous industry level certifications, awards, and achievements. His current research interests include software-defined networking and wireless sensor networks.

CHENGBIN HU received the M.S. degree in medical science from the University of South Florida, in 2016, where he is currently pursuing the Ph.D. degree in computer science and engineering. He has authored seven peer-reviewed papers in the areas of network security and medical science. His research interests include wireless network security, medical data analysis, and deep learning.