Detecting DGA domains with recurrent neural networks and side information

Ryan R. Curtin
Center for Advanced Machine Learning
Symantec Corporation
Atlanta, Georgia, USA

Andrew B. Gardner
Center for Advanced Machine Learning
Symantec Corporation
Atlanta, Georgia, USA

Slawomir Grzonkowski
Targeted Attack Analytics
Symantec Corporation
Dublin, Ireland

Alexey Kleymenov
Targeted Attack Analytics
Symantec Corporation
Dublin, Ireland

Alejandro Mosquera
Targeted Attack Analytics
Symantec Corporation
Reading, Berkshire, UK

ABSTRACT

Modern malware typically makes use of a domain generation algorithm (DGA) to avoid command and control domains or IPs being seized or sinkholed. This means that an infected system may attempt to access many domains in an attempt to contact the command and control server. Therefore, the automatic detection of DGA domains is an important task, both for the sake of blocking malicious domains and identifying compromised hosts. However, many DGAs use English wordlists to generate plausibly clean-looking domain names; this makes automatic detection difficult. In this work, we devise a notion of difficulty for DGA families called the *smashword score*; this measures how much a DGA family looks like English words. We find that this measure accurately reflects how much a DGA family's domains look like they are made from natural English words. We then describe our new modeling approach, which is a combination of a novel recurrent neural network architecture with domain registration side information. Our experiments show the model is capable of effectively identifying domains generated by difficult DGA families. Our experiments also show that our model outperforms existing approaches, and is able to reliably detect difficult DGA families such as `matsnu`, `suppobox`, `rovnix`, and others. The model's performance compared to the state of the art is best for DGA families that resemble English words. We believe that this model could either be used in a standalone DGA domain detector—such as an endpoint security application—or alternately the model could be used as a part of a larger malware detection system.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; *Artificial immune systems*; Web protocol security; • **Computing methodologies →** *Neural networks*.

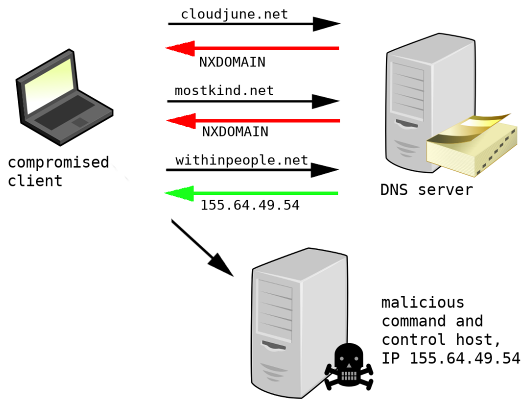


Figure 1: Typical example of malware using a DGA to find its command-and-control (C&C) server. The infected host attempts to resolve a number of DGA-generated domains, and connects to the first one that resolves successfully.

KEYWORDS

DGA, neural networks, C&C

ACM Reference Format:
Ryan R. Curtin, Andrew B. Gardner, Slawomir Grzonkowski, Alexey Kleymenov, and Alejandro Mosquera. 2019. Detecting DGA domains with recurrent neural networks and side information. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3339252.3339259

1 INTRODUCTION

Many modern malware families communicate with a centralized command and control (C&C) server. In order to do this, the malware must know the location of the C&C server to connect to—simple approaches might hardcode an IP or a domain name. But, these are easy to mitigate: the traffic to a specific IP can be trivially blocked, and domain names can be easily seized. Therefore, modern malware authors use *domain generation algorithms* (DGAs) in order to generate a large set of possible domain names where the C&C server may exist.

Typically, an infected machine will use the DGA to serially generate domain names. Each of these domain names will be resolved, and if the DNS resolution does not result in an NXDOMAIN response (i.e. if the domain name is registered), then the machine will attempt to connect to the resolved IP as if it is the C&C server. If any step of that process is not successful, then the machine will generate another domain name with the DGA and try again, until it is successful. Some DGA families generate random-looking domain names such as `xobxceagb[.]biz`; others generate difficult-to-distinguish domain names like `dutykind[.]net`.

This DGA-based approach to finding the C&C server is robust to IP blocking and domain name seizure; the C&C server operator can use any IP they have access to (and they may use different IPs at different times), and typically the number of unique domain names a DGA can generate is quite large, and sometimes the DGA itself may be hard to reverse engineer. Therefore, it is not generally feasible to pre-emptively seize all domain names that a DGA could generate. In fact, DGAs may even generate domain names that are not malicious or compromised, and this does not affect the malware's ability to reach the C&C server eventually.

As a consequence, the task of determining whether or not a given domain name is produced by a DGA is an integral part of modern malware defenses. Simultaneously, as DGA authors create DGAs that generate domain names that do not look randomly-generated, the challenge of detecting these domain names increases.

A large body of related work seeks to use machine learning techniques directly to classify domains as generated by DGAs or not. Our contribution adds to this lineage of work; here, our machine learning detector is one component of an effective malware detection system. To this end, we describe a machine learning system that is able to accurately classify a domain name as DGA-generated or clean using only the domain name itself and some simple additional features derived from WHOIS data. This system is especially effective on DGA families that generate domain names based on English word lists (i.e., domains that look benign to a human observer). Compared to previous approaches, our system performs better on difficult-to-detect DGA families that resemble English words (such as the `matsnu` and `suppobox` families), and the system is not difficult to deploy in a real-world environment—either as a standalone detector or as part of a larger malware detection system.

Overall, this paper makes the following contributions:

- We provide a novel machine learning system built partially on recurrent neural networks that is capable of classifying DGA-generated domain names even from families traditionally understood as difficult. To achieve this degree of performance, our model takes advantage of side information such as WHOIS.
- This model is robust: although it is trained with WHOIS information, predictions can still be made if WHOIS or other network level information is not available. This is crucial for real-time detection and prevention of malware outbreaks.
- We devise a new measure that we term the *smashword score*. We rank 41 DGAs in terms of detection difficulty using this measure, giving an intuitive measure of difficulty related to how closely the domain resembles English words. Our approach can be re-used for new DGA families, and we believe our measure is useful for other DGA detection works in the future.
- We successfully classify difficult DGA-generated domains using our model that other state-of-the-art approaches could not conclusively label; this includes domains with high smashword scores (e.g., those that are composed of combinations of English words). Note that these domains can even be difficult for humans to classify correctly.

2 RELATED WORK

The problem of distinguishing legitimate domain names from algorithmically generated is certainly not new, and has been studied for a number of years. DGAs first became widely known to the community with the introduction of Kraken [24] and Conficker [35] in 2008. Since that time, DGAs in malware have proliferated.

The early efforts to stop this threat were dealing with lack of sufficient training data to apply machine learning approaches [4]. One decade later it continues to be a problem but to a smaller extent. Thus early proposed approaches and techniques were rather statistical. For example Yadav et al. [44] applied such technique to show differences between valid domain names and algorithmically generated ones. The limitation of such approach would be that it often does not transfer to a different DGA family.

Another milestone in detection techniques was credited to more extensive usage of DNS data. For example Zhou et al. [47] gathered DNS NXDOMAIN data from RDNSs and then used it to assemble a set of suspicious algorithmically generated domain names.

A different approach was proposed by Jian et al. [20]. It relied on DNS traffic analysis but only for failed lookups. In this technique interactions between hosts and failed domain names would be extracted. Then a graph decomposition algorithm using nonnegative matrix tri-factorization technique to iteratively extract coherent co-clusters would be applied. The obtained sub-graphs would be further analyzed by exploring temporal properties of generated clusters. The authors claim that their anomaly based approach can detect new and previously undiscovered threats.

Further research efforts evolved towards more and more extensive usage of machine learning techniques. At a large scale, it was pioneered by Phoenix [36] that was able to use both the URLs and other side information to detect DGA botnets. The list of parameters observed by this system includes some handcrafted features like pronounceability, blacklist information, DNS query information. This approach does not use any recurrent neural network (RNN) or powerful modeling technique for the domains themselves leaving a room for improvement. Tong and Nguyen [42] have already proposed extensions to the Phoenix system. They included additional measures such as entropy, n-grams and modified distance metric for domain classification.

Further progress in DGA detection was reported when using machine learning techniques. For example, Zhao et al. [46] addressed the problem in the context of detecting APT malware. The authors proposed 14 features based on their big data research to characterize different properties of malware-related DNS and the ways that they are queried as well as defined network traffic features that can identify the traffic of compromised clients that have remotely been controlled. The features are comprised of signature-based engine, anomaly-based engine and so-called dynamic DNS features. The data was filtered by using Alexa[1] popularity and prevalence based on the number of hosts connecting to domains. As the outcome, an engine was built and it was used to compute reputation scores for IP addresses using extracted features vectors. The results are produced by using the J48 decision tree algorithm.

A comparable approach was presented by Luo et al. [27] who described a system using lexical patterns that were extracted from clean domains listed in the Alexa top 100k domains as well as confirmed malicious DGA cases. The proposed approach is machine learning-based and achieves 93% accuracy for detecting malicious domains on the test dataset.

Additional improvements for state of the art results were reported by Woodbridge et al. [43]. Despite a relatively simple Long Short-Term Memory (LSTM) network used to classify DGA domains, the approach was reported to have a high level of effectiveness. The presented results still have certain shortcomings, especially for difficult DGA classes that resemble English words.

The same problem was approached from a different angle by Anderson et al. [2]. The authors use a Generative Adversarial Network (GAN) to generate adversarial DGA domain names to try and deceive a classifier. The authors were able to achieve this goal. Then the GAN-generated domain names were added to the training set, which resulted in improved DGA detection performance. However, the authors did not test on any DGA families that look like they are made up of English words.

Shibahara et al. [38] proposed a slightly different algorithm that is using RNN on changes in network communication with a goal of reducing malware analysis time. This approach is not DGA-only specific but rather generic and attempts to cover other types of malware. However, it could successfully be used against DGA-type of threats based on their communication patterns. Thus this technique requires additional run-time data that is not required in many of the other approaches as it requires malware sandboxing. The authors claim that without their optimization the analysis time takes over 15 min. and their approach reduces this time by 67%, preserving the detection rate of malicious URLs at 97.9%.

Overall, though the task of DGA detection is certainly not new, there has not been much focus on directly detecting DGA families made from English words using the domain itself as a feature. This task has been described as 'extremely difficult' in some previous works [43]. Here, our focus is specifically on those DGA families.

3 MEASURING THE DIFFICULTY OF DETECTION OF A DGA FAMILY

Since data-based approaches for the detection of malicious domains have been a recurrent trend during recent years, it is inevitable that malware authors would shift to generation algorithms that overlap with lexical patterns commonly found in clean datasets to avoid being detected. Taking into account this adversarial environment, we need to be able to measure how our DGA detection models will perform not only overall, but also against the most difficult samples. In this context, 'difficult' samples can be understood to be those that trick existing detectors—the most relevant example is those DGA families that combine English words, like the `matsnu` family [39], which was one of the first of many families to build domain names from English word lists. These generate domains like the natural-looking domains `songneckspiritprintmetal[.]com` and `westassociatereplacerisk[.]com`, which present a much harder challenge to the many detection systems that depend on lexical features [2, 36, 43, 45].

An exploratory data analysis of our dataset shows that DGA families have characteristics that can affect the performance of classification approaches. From an information theory point of view, both the average length $\bar{l}(\cdot)$ and the average character entropy [37] $\bar{c}(\cdot)$ of the domain names seem likely to be interesting features to compare. The entropy of a single domain x is calculated as below:

$$\hat{c}(x) = -\sum_{x_i \in x} p(x_i) \log_2 p(x_i) \tag{1}$$

where $p(x_i)$ is the empirical probability of the character x_i in the string x. However, in our experiments, we found no serious correlation between the average character entropy $\bar{c}(\cdot)$ of a DGA family and if that family was made up of difficult English-like words. Thus, we cannot use $\bar{c}(\cdot)$ as a proxy for the difficulty of detecting a family.

Therefore, we have developed the *smashword score* $\hat{s}(\cdot)$, which is the the average n-gram overlap (with n ranging from 3-5) with words from an English dictionary. The computation of the smashword score amounts to calculating term-frequency inverse-document-frequency (TF-IDF) [40] scores for a domain name using an English list of words as a reference document set. Specifically:

$$\hat{s}(x) = \frac{1}{|\mathcal{N}_{i,j}(x)|} \sum_{n_i \in \mathcal{N}_{i,j}(x) \cap \mathcal{N}_{i,j}(D)} \log\left(|\{d \in D : n_i \in d\}|\right). \tag{2}$$

In this equation, $\mathcal{N}_{i,j}(x)$ refers to the set of character n-grams in the domain x of length i or j, D refers to the English word list, and $\mathcal{N}_{i,j}(D)$ refers to the set of character n-grams in the entire word list D of length i or j. The log term is the count of times an n-gram appears in the entire word list D. If there is no overlap in any n-grams between the domain and the word list, the score is has a lower bound of 0 and an upper bound depends on the word list. The score is normalized to the number of n-grams in the domain.

Computing the smashword score $\hat{s}(x)$ for a string x can be done in $\min(|\mathcal{N}_{3,4,5}(x)|, |\mathcal{N}_{3,4,5}(D)|)$ operations; but since $|\mathcal{N}_{3,4,5}(x)|$ will generally be much smaller than $|\mathcal{N}_{3,4,5}(D)|$, we can say that the computation will generally take time linear in the length of the string x, since in a string with length $|x|$, there are $|x| - 2$ 3-grams, $|x| - 3$ 4-grams, and $|x| - 4$ 5-grams.

The average smashword score $\bar{s}(\cdot)$ of a DGA family is then calculated by simply taking the average smashword score $\hat{s}(\cdot)$ of all of the domains in that family that are present in the data.

We can expect that domains with a high smashword score will resemble English words, and thus we expect that $\bar{s}(\cdot)$ is a good indicator of the difficulty of detecting a DGA domain. Indeed, in the following section we find that our data bears out this expectation.

4 DGA FAMILIES

Before introducing our proposed classifier and experiments, we introduce our dataset of DGA families and clean domains in order to perform some exploratory analyses. In this section we establish

[1] See https://www.alexa.com/topsites.

DGA family	n	$\bar{l}(x)$	$\bar{c}(x)$	$\bar{s}(x)$	**sample 1**	**sample 2**
banjori[3]	435385	22.165	3.767	**173.909**	iivnleasuredehydratorysagp[.]com	yanzvinskycattederifg[.]com
beebone[3]	210	13.223	3.466	66.159	backdates10[.]com	dnsfor3[.]net
chinad[6]	256	19.843	3.882	19.565	evybt5gtf2tprvbi[.]info	m5j42r6uiqov2dgm[.]biz
conficker[5]	100000	11.754	3.208	19.988	jemmmpo[.]ws	xobxceagb[.]biz
gozi[3]	1212	23.262	3.550	**222.240**	questionibus[.]com	chrisredemptisviros[.]com
locky[5]	8	14.125	3.226	20.800	pccibcjncnhjn[.]yt	qtysmobytagnrv[.]it
matsnu[6]	99995	30.527	3.757	**332.581**	dutytillboxpossessprogress[.]com	dropbridgeexplorecraftgive[.]com
murofet[6]	102020	41.619	4.488	38.683	hyernzfvd10k57lyozotazkrp52gzfyp22eu[.]org	m19e41hydyfxgtcxjyn10nynukulxdvhub18[.]com
necurs[6]	2048	17.029	3.541	31.256	gmwyfuhwqveqcbasvtj[.]in	pysetkbwbryxbegmwg[.]eu
newgoz[6]	1000	29.885	4.214	23.476	afkv141b7q87du27t2i1b91gfp[.]biz	krx32h1jjusuatm2aetjkn6jn[.]org
others_dga_b[7]	2775	19.233	3.485	90.369	qktpxl[.]info	meatopen[.]net
proslikefan[6]	130	11.584	3.225	20.768	rxxeqcoy[.]cc	avhpdzz[.]com
pykspa[6]	5010	14.470	3.375	40.551	tdxiogn[.]org	dwaejwfox[.]cc
qakbot[6]	5000	20.719	3.740	43.164	hfbtlwqlqvoywaknjksaaeeus[.]net	wlwcrapplotshymcia[.]org
ramdo[6]	100000	20.000	3.393	51.347	aaooekcoyysuouaa[.]org	skmaogeyiwqgeyym[.]org
ramnit[6]	100	17.340	3.585	35.981	ckyioylutybvcxv[.]com	ibvtknxochoyjidm[.]com
ranbyus[6]	80	18.750	3.684	35.726	cyedjumagsrrav[.]cc	jxbdxeyxttdmcjagi[.]me
rovnix[6]	99764	26.797	3.685	**284.222**	coloniesgovernmentsthe[.]com	tohavetheontheofassent[.]com
shiotob[6]	2001	16.566	3.655	20.998	cwitdw951w1n9cm[.]net	3pttjmaw2g[.]com
suppobox[6]	258	17.298	3.341	**152.536**	bartholomewalbertson[.]net	dutykind[.]net
tinba[6]	101001	16.006	3.457	31.592	fuvfpkpwgjqj[.]com	hosvsbvbveee[.]com
volatile[3]	352	19.000	3.745	112.362	hpyersgtdobfla[.]info	ergtydobflashp[.]info

Table 1: Information about a representative set of DGA families. We include the number of samples n, average length $\bar{l}(\cdot)$, the average entropy $\bar{c}(\cdot)$, the average smash-word score $\bar{s}(\cdot)$, and two examples of the domains found in the family. Families with high smash-word score are given in bold.

the Ground Truth (GT) datasets for both confirmed DGA and non-DGA domains. Each of our sources are taken from public locations, making our dataset straightforward to reproduce.

4.1 DGA Ground Truth Set

The GT for DGA domains consists of domains generated using Python implementations of real-world malware families using various seeds if necessary as an input, as well as domains collected from the wild. In order to have sufficiently diverse coverage, the following entities were selected in order to represent many wide-spread patterns of DGA domains seen in-the-wild in 2017/2018:

- random-looking 2nd level domain names
- random-looking 3rd level domain names with generic 2nd level domain (usually dynamic DNS provider)
- domain names comprised of random words (generally English)

The last type of domains was of our particular interest as lexically they are virtually indistinguishable from legal domains which means that some extra techniques are required for detection.

In almost all cases DGAs are using some sort of input seeds in order to either randomize the output and don't generated the same domains twice, or make it unpredictable for researchers to avoid blocking or sinkholing. Here are some of the popular seeds:

- current date and/or time
- value embedded into a sample/group of samples by campaign (usually one DWORD)
- string(s) available online either on a malware authors or public server
- 3rd party public online document (for example, The US Declaration of Independence, the Apple license, etc)

In addition, some work has been done to make sure that there are diverse top level domains (TLDs) represented as malware authors tend to use only some particular ones which may introduce substantial skew to our dataset. Overall, we have collected 41 DGA families. Information on each family is given in Table 1, including the average entropy $\bar{c}(\cdot)$ and average smashword score $\bar{s}(\cdot)$. The families are collected from multiple sources and denoted in the table: DGArchive[2], an implementation for the `locky` family found on Github[3], Andrey Abakumov's DGA repository on Github[4], and Johannes Bader's DGA implementations[5]. Smaller or unknown families were grouped as `others_dga` and `others_dga_b` [6]. All of this data is publicly available; our set of DGA domains is reproducible.

4.2 Non-DGA Ground Truth Set

For non-DGA domains the GT is comprised of domains found in the Alexa top 1 million sites and the OpenDNS public domain lists[7], giving 1.02M clean domains for the clean GT set. There were multiple major problems that had to be addressed:

[2]See https://dgarchive.caad.fkie.fraunhofer.de/.
[3]See https://github.com/sourcekris/locky.
[4]See https://github.com/andrewaeva/DGA.
[5]See https://johannesbader.ch and https://github.com/baderj/domain_generation_algorithms.
[6]Sinkholed domains collected from public WHOIS registration information containing `jgou.veia@gmail.com` as the contact email.
[7]These lists can be found at https://github.com/opendns/public-domain-lists.

(1) some prevalent DGA domains can manage to get into the list of world top popular domains,

(2) 3rd level domains should be covered separately,

(3) some DGA domains are so short that they collided with the non-DGA domains, and

(4) some DGA domains use combinations of English words, so the chances that they collide with non-DGA domains are quite high.

In the last two cases, malware authors have no problem when collisions take place, as in any case malware will be waiting for a valid response from the C&C before following up. Just the opposite, such cases can make the work of security engineers more complicated as they cannot simply ban all domains generated by the DGA—since some domains are known to be clean. In our dataset, we only found 12 such domains that existed in both the non-DGA and DGA sets. It presents no modeling problem to leave these points in both sets.

5 SIDE INFORMATION

DGA families with a high average smashword score $\hat{s}(\cdot)$ are very hard to classify based on the domain names alone. In fact, human analysts may even have a difficult time differentiating—for instance, it is plausible at first glance that `darkhope.net` might be a personal website for a 1990s-era teenaged computer enthusiast. In reality, that domain name is generated by the `suppobox` DGA. Thus, we cannot hope to build an effective classification system using single domain names alone.

Therefore, we augment our domain names with *side information*, which we collect from the WHOIS database [11]. Specifically, given a domain name, we perform a WHOIS lookup, and extract the following numeric or Boolean features:

- `has_registrarname`: Boolean, indicates whether a registrar name is available.
- `has_contactemail`: Boolean, indicates if any contact email is available.
- `days_since_created`, `days_since_updated`, `days_until_expiration`: numeric, the number of days since the domain was created, updated, or until expiration
- `status_length`: numeric, length of the "status" field
- `has_registrant_info`, `has_admincontact_info`, `has_billingcontact_info`, `has_techcontact_info`, `has_zonecontact_info`: Boolean, indicates whether contact information is available.
- `has_registrar_iana_id`: indicates whether a registrar IANA ID is given.

Note that for a non-registered domain name (NXDOMAIN), the boolean features will all be `false`, and the numeric features will all be taken as 0.

We do not perform any semantic analysis on the content of the WHOIS record; instead, we focus on those features most likely to give us information relevant to DGAs and C&C servers: temporal information about the registration, and whether the domain itself is registered. The features we are using roughly match the type of features used by Ma et al. [28].

For our dataset, we used a snapshot of collected WHOIS data with 245M records. For our clean domain names, we matched 927k domains (91.7%) to WHOIS data, and for the DGA domains, we matched only 2.3k domains (0.18%) to WHOIS data. This is expected, given that most DGA domains are never registered. In practice, either a snapshot of WHOIS data (potentially updated nightly) or on-demand access of the WHOIS data could be used, depending on the scalability needs of the deployment.

In our dataset, DGA families have an average of 3.5% of their domains matched to WHOIS data; with the `ramnit` family matching the highest percentage at 84%, and the `pandex` family matching the lowest nonzero percentage at 0.008% (only 7 out of 91758 domains registered). 19 families, totaling 321k domains, have no domains matched to any WHOIS data.

Although having matching WHOIS data for a domain is strongly correlated with whether or not the domain arises from a DGA, note that a detector built to classify a domain as malicious simply if there is no WHOIS data would not be very effective: with our data, it would achieve a true positive rate (TPR) of 96.5%, but with an unacceptably high false positive rate (FPR) of 8.3%. Though WHOIS data gives us good information, it is not sufficient for prediction.

5.1 WHOIS and GDPR

After the passing of the European privacy bill GDPR [10], it is unclear how WHOIS lookups will be affected [19]. At the time of our experiments, WHOIS data was still publicly available. However, if this is not the case in the future, it would be easy to find alternatives. Given that the important features we extract depend more on the temporal registration information than the contact details of the registrant, we could replace our WHOIS features with DNS tracking systems like Active DNS [23] or the Alembic system [25].

At the time of this writing, it is not clear what the long-term solution for WHOIS data will be. But, since WHOIS data is widely used for security applications [5, 6, 28], it seems unlikely that the types of features we are using for our system will become unavailable.

6 MODEL ARCHITECTURE

Given the effectiveness of deep learning classifiers for character-level DGA modeling [43, 45], we have designed our DGA detector on character-level recurrent neural networks (RNNs) [21]. A character-level RNN sequentially receives characters from a string, updating internal state as each character is passed in. Instead of training the RNNs to predict the class of the domain, we instead train two RNNs to predict the next character in the domain and combine these predictions via a generalized likelihood ratio test (GLRT). In addition, our model also incorporates the WHOIS side information discussed in the previous section via model stacking. This allows us to achieve significantly better performance on more difficult DGA families.

Overall, our model is a logistic regression classifier built on the output of four different models:

(1) A character-level RNN GLRT model built only on the *subdomains* in the training set.

(2) A character-level RNN GLRT model built only on the *domains* in the training set.
(3) One-hot encoded top-level domain features (for the most popular 250 TLDs).
(4) Extracted features from the WHOIS information.

The overall architecture of the model can be seen in Figure 2. In the following subsections we describe the details of the full model.

6.1 Character-level RNN GLRT

The core of the model is the character-level RNN that uses the generalized likelihood ratio test to classify a domain or a subdomain as DGA or non-DGA. Previous approaches and other uses of RNNs often predict the class of the output directly [16, 43]; however, this only allows backpropagation of the error signal at the end of the entire sequence, which can slow the learning process.

Therefore, we build one RNN on each class in the input dataset (in our case, there are only two classes: *DGA* and *non-DGA*). Each input sequence is converted to a one-hot character encoding, and the label or expected output of the RNN for each time step is the one-hot encoding of the *next* character in the sequence. This means the RNN is trained to predict the next character in the sequence. Thus, backpropagation can be done at every timestep, instead of waiting until the end of the sequence to compare the output of the RNN with the desired label. Our model's architecture is a single LSTM layer [18] followed by a single dense layer, pictured in Figure 3. We use LSTMs to help avoid the vanishing and exploding gradient phenomenons [33]. Although it is possible to build a more complex network, we found that this provides a good balance between training time and the accuracy of the model.

In order to perform the one-hot encoding, we first build a dictionary D on the entire training set, including the 'unknown' character `'?'`. If a character is encountered at prediction time that is not in D, then it is encoded as `'?'`.

We use the categorical cross-entropy [14] for the loss function. Then, during prediction, at each time step i for the input x, the output of the model θ is a probability $p(x_i|(x_0, \ldots, x_{i-1}), \theta)$ and with this we can construct an estimate of the likelihood of the point x arising from the model θ:

$$p(x|\theta) = \prod_i p(x_i|(x_0, \ldots, x_{i-1}), \theta). \qquad (3)$$

For the generalized likelihood ratio test [31], if we calculated both likelihood estimates $p(x|\theta_{\text{non-dga}})$ and $p(x|\theta_{\text{dga}})$, we could then set a threshold η and compute

$$\Lambda(x) = \frac{p(x|\theta_{\text{dga}})}{p(x|\theta_{\text{non-dga}})}, \qquad (4)$$

and if $\eta > \Lambda(x)$, we classify the point as a DGA domain; otherwise, we classify the point as non-DGA. The value of η can be swept in order to control the false positive and true positive rate. η is directly related to the typical posterior probability of a classifier; in fact, if we normalize the likelihood estimates we can produce a posterior probability of x being a DGA domain:

$$p(\theta_{\text{dga}}|x) = \frac{p(x|\theta_{\text{dga}})}{p(x|\theta_{\text{non-dga}}) + p(x|\theta_{\text{dga}})}. \qquad (5)$$

Then, setting a threshold for $p(\theta_{\text{dga}})$ is reducible to setting a GLRT threshold η.

For our DGA classifier, we build two separate RNN-GLRT models as described above: one on the *subdomains* of our training set, and one on the *domains*. Each of these two models, in turn, contains a separately-trained LSTM RNN, whose outputs are combined to perform the GLRT as shown above.

As input to the logistic regression model, we extract six features from each RNN-GLRT model, giving a total of twelve features. The features are listed below.

- A boolean feature indicating whether a domain or subdomain could be extracted from the input domain x.
- The likelihood estimate $p(x|\theta_{\text{non-dga}})$.
- The likelihood estimate $p(x|\theta_{\text{dga}})$.
- The posterior probability $p(\theta_{\text{non-dga}}|x)$.

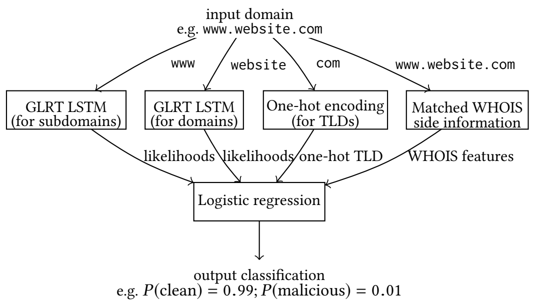


Figure 2: Overall architecture of our proposed DGA detection model that incorporates side information. Input points are split into subdomain, domain, and TLD. The subdomain and domain are run through individual RNN+GLRT models, then the output is combined with the WHOIS data and the one-hot encoded TLD into the final logistic regression model.

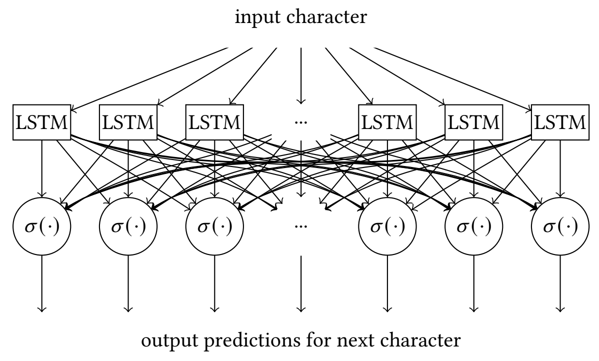


Figure 3: The individual RNN model architecture. The model takes a one-hot encoded character as input (along with its current hidden state) in order to predict the next character in the sequence.

- The posterior probability $p(\theta_{\text{dga}}|x)$.
- The likelihood ratio $\Lambda(x)$.

Since we are extracting the likelihood estimates and posterior probabilities into a logistic regression model, we actually have no need to select a threshold η—that is only needed for a standalone GLRT LSTM model. Instead, in our combined model, the logistic regression will learn directly from the probabilities and likelihoods.

6.2 Top-level domain features

Since TLDs are so short (usually two or three characters), it is excessive to train an RNN on them. Therefore, we use a one-hot encoding of the TLD, matching against the 249 most frequent TLDs in our training dataset; if there is no match, the TLD is encoded as 'other', giving a total of 250 binary features out of the TLD.

In order to perform the conversion, we used the TLD list available from http://publicsuffix.org. The most common TLDs in our dataset were `.com`, `.org`, `.ru`, `.net`, and `.info`. We found that the `.ru`, `.info`, `.biz`, and `.cc` TLDs contained significantly higher concentrations of DGA domains, with each of those TLDs containing at least 3 times as many DGA as non-DGA domains. Since we have split these into separate features, we can expect our model to learn which TLDs domain generation algorithms are more likely to use.

6.3 WHOIS side information

The WHOIS data makes up the rest of the input to the logistic regression model. It is concatenated with the RNN-GLRT features for the domain, the RNN-GLRT features for the subdomain, and the one-hot encoded TLD features.

Before all of these features are fed into the logistic regression model, we perform whitening via PCA for decorrelation and scaling [22]. This step can improve the performance of the model, although it generally also makes interpretability more difficult.

6.4 Computational concerns

Recurrent neural networks, especially those with complex memory cells like LSTMs, are well-known to be time-consuming to train [12, 26]. Our model is not exempt from this; for large datasets, it may take many hours to train[8]. However, in practice this is not a concern—a single forward pass through the model for classification is comparatively very fast, and once our model is trained, there are no computational difficulties with deployment in a low-latency detection system. This means that the model can be, e.g., deployed into a consumer endpoint security product without problems.

7 ADVERSARIAL SAMPLES

In recent years, the phenomenon of *adversarial samples* has surfaced in the deep learning community [15, 41]. In essence, a malicious actor could take a sample that was correctly classified by the model, perturb the input slightly, and the perturbed sample would be misclassified. When images are used, these perturbations are often invisible to the eye. These adversarial attacks have been successfully applied to fields outside of images, including audio [8] and malware classification [17]. Though there are some defense mechanisms that have been developed [13, 32], many of these are later found to be circumventable [7].

Given that adversarial samples are not limited to images, it is reasonable to believe that neural network-based DGA detectors could also suffer from this vulnerability. In our situation a malicious actor would wish to take a domain that is detected as from a DGA and have it labeled as a non-DGA domain. It would be very straightforward to perform an attack like the Fast Gradient Sign Method [41] to modify the characters in a domain name. In fact, it is not (generally) important to DGA authors what the domain name looks like, so there is no cost to modify the letters of the domain.

Such a technique would likely prove effective against an approach that only incorporated the domain name itself. However, note that our model also incorporates domain registration side information from WHOIS. Although a malware author can change the domain name they are using at will and nearly arbitrarily, it is significantly more difficult to cause the WHOIS registration information (such as registration date) to have specific values. To do that, a malware author might need to register a domain perhaps months in advance and host a clean website on it, which is both expensive and time-consuming. Thus, it would be more difficult for a malware author to work around our proposed model.

8 EXPERIMENTS

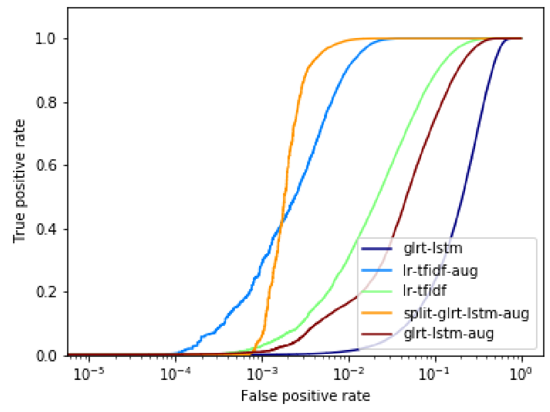
The most important situation for any DGA detection model is when it encounters an entirely new DGA family that it has never seen before. This is the situation that we focus on in our experiments, since it reflects the real-world 'zero-day' situation. We compare our model to several baselines that reflect the state-of-the-art for machine learning systems that do not use network traffic data.

Leave-one-out models. To simulate the situation where a DGA family has not been seen, we validate the performance of our DGA detection model by performing *leave-one-out* experiments, where we train the model on all DGA families except one, and then the test set consists entirely of the left-out DGA family combined with some never-before-seen non-DGA domains. This shows us how well the model is able to generalize to unseen DGA family types.

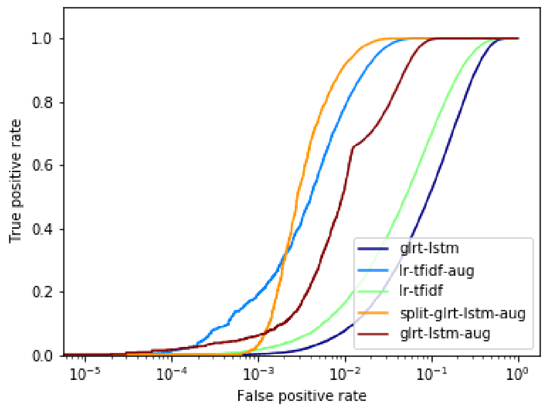
Dataset details. Our collected dataset, as described in Section 4, includes 41 DGA families plus non-DGA data, totaling 2.3 million domain names (1.01 million non-DGA, 1.28 million DGA). Of these 41 DGA families, many with high average smashword score have been specifically mentioned in related work as difficult. The LSTM model of Woodbridge et al. [43] is specifically shown to perform very poorly on the `matsnu`, `suppobox`, and `beebone` families, each of which have above average to very large average smashword scores. Mac et al. [29] claim that `matsnu` is not differentiable from non-DGA domains at all, and show very poor performance on all their surveyed algorithms for the `nymaim` DGA, which is very similar to the `gozi` DGA that we use here. Because our model has been specifically designed to focus on DGA families that are understood to be more difficult, we will focus on these families.

Baseline models. We wish to compare the performance of our proposed model with existing and baseline approaches. Therefore, we compare our model with four other models, which we now introduce. Two of these are simple baseline models, with and without WHOIS information, and the other two are based on LSTM architectures that represent the most closely related state-of-the-art work of Woodbridge et al. [43].

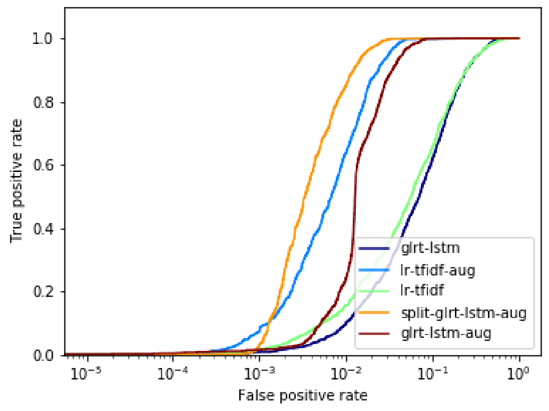
[8]Our training was conducted on a high-end consumer-grade system with a single GPU.



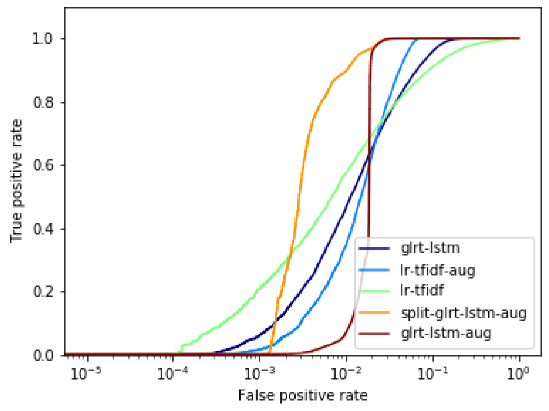
(a) ROC curves for `matsnu` family.



(b) ROC curves for `rovnix` family.



(c) ROC curves for `gozi` family.



(d) ROC curves for `banjori` family.

Figure 4: ROC curves for different families.

- **lr-tfidf**: logistic regression on TF-IDF features extracted from the domain name using 2-grams.[9] Any WHOIS side information is not used here, so this model presents a reasonable baseline using only the domain name.
- **lr-tfidf-aug**: logistic regression on TF-IDF features extracted from the domain name using 2-grams, and augmented with the WHOIS features. This is a reasonable baseline for classification using both the domain name and the side information (WHOIS features).
- **glrt-lstm**: a GLRT LSTM model built only on the full domain name (no side information). This can be considered to be a slight improvement over the model of Woodbridge et al. [43] due to the use of the GLRT.
- **glrt-lstm-aug**: a GLRT LSTM model built only on the full domain name, and then used as input to a logistic regression model, with the WHOIS features augmented.

Our model. We refer to our model as the **split-glrt-lstm-aug** model; this is the model from Section 6.

[9]We did not use 3-grams, because the memory usage on our system was too large.

Training and implementation details. The **lr-tfidf** and **lr-tfidf-aug** models were implemented with `scikit-learn` [34], and the three LSTM-based models were implemented with Keras [9] using the TensorFlow backend [1]. Each LSTM model used 500 LSTM units and was trained for 100 epochs (passes over the dataset) with early stopping using the RMSprop optimizer, with dropout of 0.2. With our setup (one nVidia GeForce GTX TITAN X), each LSTM model took approximately 8-10 hours to train. In our experiments, we found that changing the optimizer made little difference to the resulting model, and we found that increasing or decreasing the number of LSTM units decreased performance slightly. Overall, our model seemed to be relatively robust to hyperparameter choice.

Figure 4 shows receiver operating characteristic curves (ROC curves) on the four datasets with highest smashword score. We can see in the figures that the **split-glrt-lstm-aug** model (our proposed model) outperforms each of the other models, providing better performance at lower false positive rates. For instance, on the difficult `matsnu` family, when the false positive rate is chosen to be 0.5%, the **split-glrt-lstm-aug** model operates at a true positive rate of 95%, whereas the next best model (**lr-tfidf-aug**) operates at a true positive rate of only 70%.

family	$\bar{s}(x)$	lr-tfidf	lr-tfidf-aug	glrt-lstm	glrt-lstm-aug	split-glrt-lstm-aug
matsnu	332.6	0.581	0.824	0.501	0.546	**0.891**
rovnix	284.2	0.540	0.758	0.515	0.635	**0.805**
gozi	222.2	0.540	0.684	0.520	0.548	**0.773**
banjori	173.9	0.701	0.585	0.634	0.508	**0.808**
suppobox	152.5	0.509	0.505	0.579	**0.798**	0.568
volatile	112.4	0.605	0.498	0.818	0.850	**0.958**
others_dga_b	90.4	0.649	0.502	**0.704**	0.604	0.677
beebone	66.2	0.498	0.498	0.498	0.498	**0.749**
ramdo	51.4	0.902	0.498	0.973	0.498	**0.980**
qakbot	43.2	0.940	0.498	**0.966**	0.914	0.951
pykspa	40.6	0.777	0.498	0.911	0.866	**0.957**
murofet	38.7	0.975	0.498	0.960	0.498	**0.994**
ranbyus	35.7	0.940	0.498	**0.963**	0.815	0.791
tinba	31.6	0.859	0.498	**0.968**	0.655	0.867
necurs	31.3	0.818	0.498	**0.813**	0.514	0.581
new_goz	23.3	0.863	0.498	**0.993**	0.498	0.990
shiotob	21.0	0.653	0.498	0.894	0.846	**0.902**
locky	20.8	**0.849**	0.498	0.751	0.498	0.557
proslikefan	20.8	0.647	0.498	**0.802**	0.600	0.665
conficker	19.9	0.609	0.498	**0.836**	0.619	0.649
chinad	19.6	0.786	0.498	0.952	0.498	**0.979**
dyre	6.5	0.548	0.498	0.779	0.498	**0.974**

Table 2: Partial AUC (FPR <= 0.01) performance numbers for selected different DGA families, with a focus on those with either high or low average smashword score. Our split-glrt-lstm-aug model performs best on those DGA families with high average smashword score.

In typical application scenarios, we typically care only about running our classifier at false positive rates less than or equal to 1% ($\text{FPR} \leq 0.01$). Therefore, we study the performance of the classifiers using the *partial AUC* [30] measure, which is the standard area-under-the-curve (AUC) measure specific to false positive rates less than a given threshold. In Table 2, we show the partial AUC of each model for each leave-one-out family experiment, sorted by decreasing $\bar{s}(\cdot)$.

On the most difficult families (with large $\bar{s}(\cdot)$), the proposed **split-glrt-lstm-aug** reliably and significantly outperforms all other compared models. This is the region of most interest in our work, as these families are difficult to detect—even with WHOIS data. Each of these difficult families generates domains that resemble English words; see Table 1. Note that the **lr-tfidf-aug** model and **glrt-lstm-aug** models both have access to the WHOIS features; however, only **split-glrt-lstm-aug** is able to take advantage of these to provide good performance for families with high $\bar{s}(\cdot)$.

For 'easier' families with lower $\bar{s}(\cdot)$, where the generated domains typically look more like random characters, classification can be performed more reliably with only the text of the domain itself; thus, the **glrt-lstm** model is dominant in this regime.

Overall, we see that our model is successful in detecting DGA-generated domains that resemble English words. The model appears to generalize well to different families, given the nature of our leave-one-out experiments.

9 CONCLUSION

In this paper we have considered the problem of DGA domain detection. We introduced a measure of complexity for DGA families called the *smashword score*, which reflects how closely a DGA's generated domains resemble English words. Because DGA families with higher smashword scores have typically posed greater difficulty for detection, we build a novel machine learning model consisting of recurrent neural networks (RNNs) using the generalized likelihood ratio test (GLRT), and augment these models with a logistic regression model that also includes side information such as WHOIS information.

This combined model notably outperforms existing state-of-the-art approaches on DGA families with high smashword score, such as the difficult `matsnu` and `suppobox` families. We believe that this model could be used as either a standalone model or as a part of a larger DGA detection system that could also incorporate network traffic, such as something more like the Pleiades system [3].

There is room for future improvement in our work. The model we have used is specialized for DGA families based on English words, and therefore can be less effective for those DGA families that do not look like natural domain names. Thus, in a production environment or in an improved system, our model could be ensembled with other techniques that are more effective for DGA families with lower smashword scores.

In a future work we would also like to explore multilingual approaches to tackle new families that may use non-English dictionaries and expand our side information features.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation.* USENIX Association, 265–283.

[2] Hyrum S. Anderson, Jonathan Woodbridge, and Bobby Filar. 2016. DeepDGA: Adversarially-Tuned Domain Generation and Detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (AISec '16).* ACM, New York, NY, USA, 13–21. https://doi.org/10.1145/2996758.2996767

[3] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From throw-away traffic to bots: detecting the rise of DGA-based malware. In *Proceedings of the 21st USENIX conference on Security symposium.* USENIX Association, 24–24.

[4] Adam J. Aviv and Andreas Haeberlen. 2011. Challenges in Experimenting with Botnet Detection Systems. In *Proceedings of the 4th Conference on Cyber Security Experimentation and Test (CSET'11).* USENIX Association, Berkeley, CA, USA, 6. http://dl.acm.org/citation.cfm?id=2027999.2028005

[5] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. 2011. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Syposium (NDSS 2011).*

[6] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. 2011. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International World Wide Web Conference (WWW 2011).* ACM, 197–206.

[7] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security.* ACM, 3–14.

[8] Nicholas Carlini and David Wagner. 2018. Audio adversarial examples: Targeted attacks on speech-to-text. *arXiv preprint arXiv:1801.01944* (2018).

[9] François Chollet et al. 2015. Keras. https://keras.io.

[10] Council of European Union. 2016. Council regulation (EU) no. 2016/679 (General Data Protection Regulation). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG.

[11] L. Daigle. 2004. *WHOIS Protocol Specification.* RFC 3912. RFC Editor. 1–4 pages. https://www.rfc-editor.org/rfc/rfc3912.txt

[12] Patrick Doetsch, Michal Kozielski, and Hermann Ney. 2014. Fast and robust training of recurrent neural networks for offline handwriting recognition. In *Proceedings of the 2014 14th International Conference on Frontiers in Handwriting Recognition (ICFHR).* IEEE, 279–284.

[13] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning.* MIT Press Cambridge.

[15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572* (2014).

[16] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5-6 (2005), 602–610.

[17] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security.* Springer, 62–79.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] ICANN. 2018. Data Protection/Privacy Issues. https://www.icann.org/dataprotectionprivacy.

[20] Nan Jiang, Jin Cao, Yu Jin, Li Erran Li, and Zhi-Li Zhang. 2010. Identifying Suspicious Activities Through DNS Failure Graph Analysis. In *Proceedings of the The 18th IEEE International Conference on Network Protocols (ICNP '10).* IEEE Computer Society, Washington, DC, USA, 144–153. https://doi.org/10.1109/ICNP.2010.5762763

[21] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078* (2016).

[22] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. 2018. Optimal whitening and decorrelation. *The American Statistician* (2018), 1–6.

[23] Athanasios Kountouras, Panagiotis Kintis, Chaz Lever, Yizheng Chen, Yacin Nadji, David Dagon, Manos Antonakakis, and Rodney Joffe. 2016. Enabling network security through active DNS datasets. In *International Symposium on Research in Attacks, Intrusions, and Defenses.* Springer, 188–208.

[24] Felix S. Leder and Peter Martini. 2009. NGBPA Next Generation BotNet Protocol Analysis. In *Emerging Challenges for Security, Privacy and Trust*, Dimitris Gritzalis and Javier Lopez (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 307–317.

[25] Chaz Lever, Robert Walls, Yacin Nadji, David Dagon, Patrick McDaniel, and Manos Antonakakis. 2016. Domain-Z: 28 registrations later measuring the exploitation of residual trust in domains. In *2016 IEEE Symposium on Security and Privacy (S&P).* IEEE, 691–706.

[26] Sicheng Li, Chunpeng Wu, Hai Li, Boxun Li, Yu Wang, and Qinru Qiu. 2015. Fpga acceleration of recurrent neural network based language model. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 111–118.

[27] Xi Luo, Liming Wang, Zhen Xu, Jing Yang, Mo Sun, and Jing Wang. 2017. DGASensor: Fast Detection for DGA-Based Malwares. In *Proceedings of the 5th International Conference on Communications and Broadband Networking (ICCBN '17).* ACM, New York, NY, USA, 47–53. https://doi.org/10.1145/3057109.3057112

[28] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009).* ACM, 1245–1254.

[29] Hieu Mac, Duc Tran, Van Tong, Linh Giang Nguyen, and Hai Anh Tran. 2017. DGA Botnet Detection Using Supervised Learning Methods. In *Proceedings of the Eighth International Symposium on Information and Communication Technology (SoICT 2017).* ACM, New York, NY, USA, 211–218. https://doi.org/10.1145/3155133.3155166

[30] Donna K. McClish. 1989. Analyzing a portion of the ROC curve. *Medical Decision Making* 9, 3 (1989), 190–195.

[31] Jerzy Neyman and Egon S Pearson. 1933. IX. On the problem of the most efficient tests of statistical hypotheses. *Phil. Trans. R. Soc. Lond. A* 231, 694-706 (1933), 289–337.

[32] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on.* IEEE, 582–597.

[33] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML '13)*, Vol. 28. PMLR, Atlanta, Georgia, USA, 1310–1318.

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[35] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. 2009. 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 (LEET'09).* USENIX Association, Berkeley, CA, USA, 7–7. http://dl.acm.org/citation.cfm?id=1855676.1855683

[36] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. 2014. Phoenix: DGA-Based Botnet Tracking and Intelligence. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Sven Dietrich (Ed.). Springer International Publishing, Cham, 192–211.

[37] C. E. Shannon. 1948. A Mathematical Theory of Communication. *Bell Systems Technical Journal* 27 (1948), 623–656.

[38] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada. 2016. Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning. *2016 IEEE Global Communications Conference (GLOBECOM)* (2016), 1–7.

[39] Stanislav Skuratovich. 2015. *MATSNU.* Technical Report. Check Point Software Technologies Ltd.

[40] Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[42] Van Tong and Giang Nguyen. 2016. A method for detecting DGA botnet based on semantic and cluster analysis. In *Proceedings of the Seventh Symposium on Information and Communication Technology, SoICT 2016, Ho Chi Minh City, Vietnam, December 8-9, 2016.* 272–277. https://doi.org/10.1145/3011077.3011112

[43] Jonathan Woodbridge, Hyrum S Anderson, Anjum Ahuja, and Daniel Grant. 2016. Predicting domain generation algorithms with long short-term memory networks. *arXiv preprint arXiv:1611.00791* (2016).

[44] Sandeep Yadav, Ashwath Kumar, Krishna Reddy, A L. Narasimha Reddy, and Supranamaya Ranjan. 2012. Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis. 20 (10 2012).

[45] Bin Yu, Jie Pan, Jiaming Hu, Anderson Nascimento, and Martine De Cock. 2018. Character Level based Detection of DGA Domain Names. In *Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN '18).*

[46] Guodong Zhao, Ke Xu, Lei Xu, and Bo Wu. 2015. Detecting APT Malware Infections Based on Malicious DNS and Traffic Analysis. 3 (01 2015), 1132–1142.

[47] Yonglin Zhou, Qing-Shan Li, Qidi Miao, and Kangbin Yim. 2013. DGA-Based Botnet Detection Using DNS Traffic. *J. Internet Serv. Inf. Secur.* 3 (2013), 116–123.