Expert Systems With Applications 124 (2019) 156–163

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Detection of algorithmically generated malicious domain names using masked N-grams

Jose Selvi[a], Ricardo J. Rodríguez[b,*], Emilio Soria-Olivas[a]

[a] IDAL, Intelligent Data Analysis, Department of Electronic Engineering, ETSE, University of Valencia, Spain
[b] Centro Universitario de la Defensa, Academia General Militar, Zaragoza, Spain

ARTICLE INFO

Article history:
Received 28 February 2018
Revised 18 January 2019
Accepted 19 January 2019
Available online 24 January 2019

Keywords:
Random Forest
Malware
Domain-generated algorithms

ABSTRACT

Malware detection is a challenge that has increased in complexity in the last few years. A widely adopted strategy is to detect malware by means of analyzing network traffic, capturing the communications with their command and control (C&C) servers. However, some malware families have shifted to a stealthier communication strategy, since anti-malware companies maintain blacklists of known malicious locations. Instead of using static IP addresses or domain names, they algorithmically generate domain names that may host their C&C servers. Hence, blacklist approaches become ineffective since the number of domain names to block is large and varies from time to time. In this paper, we introduce a machine learning approach using Random Forest that relies on purely lexical features of the domain names to detect algorithmically generated domains. In particular, we propose using *masked N-grams*, together with other statistics obtained from the domain name. Furthermore, we provide a dataset built for experimentation that contains regular and algorithmically generated domain names, coming from different malware families. We also classify these families according to their type of domain generation algorithm. Our findings show that masked N-grams provide detection accuracy that is comparable to that of other existing techniques, but with much better performance.

1. Introduction

The malicious software (malware) scene has changed since the early 90s, when the main goal of malware research groups such as 29A (29A Labs, 1995) was to gain kudos among peers, to improve their knowledge of technology, and to expose security risks to the public. However, nowadays malware has become a highly profitable business. For instance, Europol stated in 2013 that the global impact of cybercrime rose close to US $3 Trillion, making it more profitable than the global trade in marijuana, cocaine, and heroin combined (Europol, 2013). This cybercrime profit strongly relies on malware, including banking trojans, ransomware, and other sophisticated types of malware.

According to AV Test (2017), the number of malware samples has exponentially increased since 2006, although the number of new malware programs has followed a more linear trend. To manually analyze and understand the behavior of a malware sample by means of reverse engineering is a very error-prone and time-consuming task, becoming infeasible for anti-malware companies which analyze large corpuses of malware samples daily. For instance, Kaspersky reported a daily analysis of 350,000 malware samples in 2013 (Kaspersky Lab, 2014). Therefore, many studies of automatic malware detection approaches have been published in the literature in the last few years.

Many of these works address network-based malware detection approaches, since most malware samples communicate with their Command & Control (C&C) servers using the Internet. Malware uses a large corpus of network communication protocols (such as IRC, HTTP, or even DNS, to name a few), depending on its design. In any event, it eventually connects to specific domain names or IP addresses from the Internet. When a malicious domain name (or IP address) is detected, several actions are taken in cooperation with the corresponding Internet service providers and domain registrars to take down the malicious server and add the domain name (or the IP address) to a blacklist of well-known malicious locations.

To avoid early detection of C&C servers, malware developers have started to use a stealthier communication strategy. Specifically, they algorithmically generate domain names that may host a C&C server. Then, a malware sample tries to reach such a domain server. When it connects, the malware sample starts to communicate. Otherwise, another C&C domain name is automatically computed and tested. These algorithmically generated domain names

* Corresponding author.
E-mail addresses: jselvi@pentester.es (J. Selvi), rjrodriguez@unizar.es (R.J. Rodríguez), emilio.soria@uv.es (E. Soria-Olivas).

https://doi.org/10.1016/j.eswa.2019.01.050
0957-4174/

are computed by a Domain Generation Algorithm (DGA), a technique which became popular in 2008 thanks to the Conficker worm (Porras, Saïdi, & Yegneswaran, 2009). Since these algorithms generate domain names in a dynamic way, anti-malware companies have more difficulties in discovering the malicious locations and in blacklisting them in a timely and efficient manner.

However, algorithmically generated (DGA-based) domain names are *far different* from regular domain names. Regular domain names are commonly based on a word (or a set of words) which is fairly representative of the service provided by the server (usually a name easy to remember and type). On the other hand, the outcomes of a DGA are always proportional to the domain name length and the randomness used by the algorithm. As a result, most DGA-based domain names seem like random,"weird" words to the human eye.

The contribution of this paper is two-fold. First, we build a dataset containing regular and DGA-based domain names and classify different malware families according to their type of DGA. We have also publicly released our dataset for the sake of reproducibility. Second, we introduce a machine learning approach to detect algorithmically generated domain names in a network communication. Obviously, although Domain Generation Algorithm is a technique that malware developers use, overcoming this technique does not mean overcoming malware itself. There are other approaches for C&C communications that they could use. The solution that we propose in this paper should be seen as a part of an in-depth defense strategy, and not as a solution to malware on its own.

In particular, we use Random Forest as a machine learning model and Boruta as the feature selection algorithm. Our approach relies on a set of characteristics (features) extracted from the domain names. Besides statistical features, in this paper we propose the use of what we have termed as "masked N-grams". Recall that a N-gram is a contiguous sequence of N items from a given sample of text (or speech). In masked N-grams, every character of the given sample is substituted by a character that represents its type (i.e., consonant, vowel, digit, or other symbol). We also evaluate the detection accuracy when masked N-grams are used. Our findings show that similar results of detection accuracy are achieved with fewer features when masked N-grams are used, thus improving the performance of the detection system.

This paper is organized as follows. Section 2 introduces related work. Section 3 describes the dataset and the classification of malware families according to their DGA, and also introduces the set of statistical features and masked N-grams that we propose. Section 4 then details the machine learning models used as a detection strategy. In Section 5 we present our experiments and discuss our findings. Finally, Section 6 concludes the paper.

2. Related work

In the last few years, a combination of network and lexical features has been used to detect DGA-based domain names. The Domain Name Server (DNS) protocol (Mockapetris, 1987) is responsible for locating Internet domain names and translating them into Internet IP addresses. The DNS protocol describes how request and response messages are built, including fields with information other than the requested domain name itself, such as the message type, destination IP addresses, or Time-To-Live (TTL), among others.

A fully qualified domain name (FQDN) is the complete domain name for a specific system on the Internet. The FQDN is divided into two parts, the hostname and the domain name. For instance, imagine an email server located at the FQDN myemail.foo.bar.com. The hostname is *myemail*, while the domain is foo.bar.com. The division of a domain name string is read from right to left, and is composed of the top-level domain (TLD), second level domain (often simply known as "domain"), and subdomains (third level domain, fourth level domain, and so on). Examples of how a FQDN is divided are given in Table 1. Note that a FQDN lacks the TCP/IP protocol name (e.g., http:// or ftp://).

Table 1
Examples of structure of a FQDN.

FQDN	Hostname	Subdomain	Domain	TLD
www.google.com	www		google	com
jselvi.blogs.uv.es	jselvi	blogs	uv	es

In 2011, Bilge et al. published EXPOSURE (Bilge, Kirda, Kruegel, & Balduzzi, 2011), later improved in Bilge, Sen, Balzarotti, Kirda, and Kruegel (2014). EXPOSURE is a detection system of malicious domains that employs passive DNS analysis techniques relying on 15 composed features, organized in 4 groups: time-based features, DNS answer-based features, TTL-based features, and domain name-based features. Most of these features must be extracted from real-time (or recently captured) DNS traffic, since they rely on volatile data that may change over time.

These features were selected after a manual analysis process, targeting well-known malware behaviors. In terms of lexical features, they focused on two values: the percentage of numerical characters and the percentage of the longest meaningful substring length. These features were selected in order to detect DGA-based domain names. However, the approach lacked a full lexical analysis for domain names. EXPOSURE achieved 98.4% successful detection with around 1% of false positives, using a J48 decision tree (Witten, Frank, & Hall, 2011).

Antonakakis et al. introduced Pleiades in Antonakakis et al. (2012). Their approach focused on failed DNS requests, since a DGA generates hundreds to thousands of domain names, but only a few of them are successfully resolved. Pleiades clusters those failed domains with lists of previously known DGA-based and legitimate domains. In addition to those features based on volatile information, a set of statistical features of the domain names (e.g., similar length, level of randomness, and frequency distribution of characters) was used as a detection trigger. A hidden Markov model was finally used to cluster the suspicious domains.

Schiavoni et al. proposed Phoenix in Schiavoni, Maggi, Cavallaro, and Zanero (2014), a layered system that detects DGA-based domain names and classifies them based on the IP addresses resolved. Their approach relies on two features widely used in linguistics: meaningful characters ratio and N-gram normality score. These features provide a measurement of meaning and pronounceability, as a means to detect randomly generated strings. They used a well-known list of non-malicious domain names (namely, Alexa's list) to calculate the probabilistic distribution for legitimate domains based on those linguistic features. The Mahalanobis distance and an empirical threshold were finally used to classify a suspicious domain name as a DGA-based domain name. Later on, they clustered those malicious domain names together with well-known DGA-based domains to classify them as belonging to a specific malware family.

Finally, da Luz (2013) used a combination of 18 lexical features (such as N-gram statistics, Shannon entropy, length, number of vowels per consonant, among others) and 17 network features (such as TTL statistics and number of IP subnetworks, to name a couple) extracted from a (passive) DNS system. His approach is based on some of the lexical features used by Antonakakis et al. (2012), although most of them were simplified to reduce dimensionality and to improve speed. These features were used with two different datasets and three different machine learning models (namely, k-Nearest Neighbors, Decision

Trees, and Random Forests). The best results were achieved with Random Forests.

Note that although all the previous works produce very good results (in terms of detection accuracy), they rely strongly on volatile information that may change over time. For instance, IP addresses in a DNS response can differ at any given moment from those obtained with the same requests a few weeks earlier. In such cases, it is very unlikely that experiments can be reproduced obtaining similar results since the datasets are not captured at the same time interval. Therefore, it is difficult to train a machine learning model with algorithmically generated domain names that are inactive today or that cannot be artificially generated, since real DNS responses cannot be obtained.

For example, Kraken was a famous piece of malware using a DGA. It was active for several years and, at some point, started using a new DGA. When the first DGA is no being longer used, it is impossible to capture some characteristics used in previous works for this classification problem, so it is impossible to train a model using this information.

To overcome these issues, in this paper we propose a detection technique based on a purely lexical analysis of the domain names. Domain names, unlike other fields within DNS packets, never change over time, and thus they are suitable for training a machine learning model with DGA-based domain names that are currently inactive. Furthermore, our approach allows other researchers to reproduce our work, verify our results, and use them as a baseline for future research. Thus, our work seeks to maintain the good results from previous works, but with a more restricted set of characteristics.

3. Datasets and features

This section first describes the dataset that we built for experimentation. The features that are extracted from the domain names are then introduced.

3.1. Dataset

To evaluate our approach, we needed a representative set of *clean* domain names and of algorithmically generated domain names. Unfortunately, the lack of best practices and interest in sharing between academia and business communities in the realm of computer security makes it difficult for academic researchers to access datasets (Rossow et al., 2012). Therefore, researchers have to create their own datasets for experimentation using the data that they are able to collect – which may not be an easy task.

Clean domain names are associated to legitimate Internet services (i.e., not malicious). In this paper, we assume that most Internet traffic is related to web services (i.e., HTTP protocol). In this regard, we built our dataset of clean domains using the top lists provided by the well-known Internet portal Alexa (Alexa, 2016b). These lists include the most visited websites on the Internet, thus representing a good set of legitimate domain names. In particular, Alexa publishes various top rankings (top 10, 100, 1000, and so on) and classifies them under different criteria, such as global, per country, or per topic (e.g., news, computers, health, among others). To obtain as much information as possible, we downloaded Alexa's top 1 million list (Alexa, 2016a). We labeled this dataset as *CLEAN* dataset. Examples of these sites are google.com, amazon.com, or qq.com, to name a few.

For algorithmically generated domain names, we used the freely available repository of Bader (2016). This repository contains thousands of these domains, provided by 26 implementations of existing DGAs in malware samples (at the time of writing). A few examples of these domains are shown in Table 2.

Table 2
Example of DGA generated domains (extracted from Bader, 2016).

gvllisqi.eu	odgmmjwsdsrb.net
wfxspste.cc	uuummdqifcon.ru
pvtlkprr.co	washingtoncalanthe.net
hzsitbdm.eu	mjuwntiwmtya.net
xxlzgrom.cc	gacyzuh.com
qhbynrab.co	dbzwestnessbiophysicalohax.com

Bader published several Python scripts emulating the DGAs used by malware samples, plus some domain names generated by those scripts. These algorithms follow different approaches. The two main approaches to generating a pseudo-random string are using an initial seed and selecting characters at random, or randomly combining words from a given word list. As a result, the algorithmically generated domain names have different characteristics. Table 3 categorizes under these approaches the DGAs given in Bader (2016).

Unfortunately, Bader provided widely varying amounts of algorithmically generated domain names for each malware family in his repository. For instance, there are a thousand examples for the `Murofet` family, whereas only 12 examples are cited for the `Gozi` family. This data imbalance makes the data less useful for training a machine learning model aimed at detecting algorithmically generated domain names. In this study, we generated our own list of 100,000 domains per malware family to build a more balanced dataset, and then we randomly selected one thousand of these domains. We labeled this dataset as *MALWARE* dataset.

Note that although we analyzed 26 malware families, we obtained 32,000 DGA-based domain names. This is because there are different implementations of DGA for some of the malware families under consideration. In particular, the malware families with two (or more) types of DGA are `Fobber`, `Kraken` (each DGA of a different type, as shown in Table 3), `Locky`, `Murofet` (three different DGAs), and `Ranbyus`.

It is important to remark that we are detecting algorithmically generated domain names suspected of being used by malware, not malware itself. As a consequence, it could be that a legitimate service using algorithmically generated domain or host names with similar characteristics to those generated by malware would be detected as malware. However, such situations occur very rarely, and thus we will consider them as regular false positives.

Note that the combination of both datasets provides a dataset with very unbalanced classes. To solve this issue, as shown in Table 4, we decided to select the first 32,000 domain names from the Alexa dataset. We decided to select the top domains instead of a randomly selected set of domains, since the former are more representative of how a legitimate domain looks.

Following best practices (Rossow et al., 2012), to allow other researchers to replicate our work and to foster research in this area, the dataset built in this study is freely available at https://github.com/jselvi/phd/tree/master/dga.

3.2. Domain name features

There are a number of features that can be extracted from a FQDN. Since a FQDN is a sequence of strings separated by a dot character, it can be managed as a single string, or it can be split into different parts, such as the TLD, domain (and subdomains), or the hostname, thus allowing features to be extracted separately from all of them. For instance, da Luz (2013) used a set of lexical features applied to the whole string. Table 5 summarizes the lexical features of da Luz considered for our experiments, which are described below. As explained in Section 2, we have focused

Table 3
Types of DGAs of malware families from Bader's repository (Bader, 2016).

Family	Example	Type of DGA
Banjori	zvfdestnessbiophysicalohax.com	First domain name is fixed and used as initial seed.
Corebot	ybylsvo0ahwpe2i0mdinibo.ddns.net	Static (fixed) "ddns.net" domain. Subdomain is generated by DGA.
Dircrypt	ktqyrmiyvnidd.com	Pseudo-random domain.
Dnschanger	tcfejerekw.com	Pseudo-random domain.
Fobber	ugovykiwouxhdlrtj.net	Pseudo-random domain.
Gozi	ulpurgatoriopetrum.com	Wordlist-based DGA.
Kraken v1	ozyqosysu.dyndns.org	Small set of fixed domains. Subdomain is generated by DGA.
Kraken v2	ygdcdhonlxxs.com	Pseudo-random domain.
Locky	qtysmobytagnrv.it	Pseudo-random domain.
Murofet	nwpyftn30gso51krkrnzh64f62aym29lvmtlu.biz	Pseudo-random domain.
Necurs	iqdpvmeywdb.kz	Pseudo-random domain.
Newgoz	p46prua8qn39yijcj2n2o0xq.net	Pseudo-random domain.
Nymaim	shlxqighem.com	Pseudo-random domain.
Padcrypt	nmcdnfbacafmecab.co.uk	Pseudo-random domain.
Pykspa	uyznvxlof.info	Pseudo-random domain.
Qadars	lensxmn0puz8.com	Pseudo-random domain.
Qakbot	xzzicwkdvayojtkckzzlr.biz	Pseudo-random domain.
Ramnit	ppvrnfkbarbnlm.com	Pseudo-random domain.
Ranbyus	tuusskblufagqnjan.pw	Pseudo-random domain.
Shiotob	hey9ydfb5v5o2.net	Pseudo-random domain.
Simda	purywoq.com	Pseudo-random domain.
Sisron	mjcwntiwmtya.net	Pseudo-random domain (based on timestamp).
Suppobox	willoughbyalbertson.net	Wordlist-based DGA.
Symmi	ofvaucifadvukii.ddns.net	Static (fixed) "ddns.net" domain. Subdomain is generated by DGA.
Tinba	srqpkllgskhw.in	First domain name is fixed and used as initial seed.
Unnamed_javascript_dga	rnjaigkfu.co	Pseudo-random domain.

Table 4
Experiment dataset.

Dataset	# domains	Selected	Classification label
Alexa 1M	1,000,000	32,000	CLEAN
Bader repo extended	3,198,304	32,000	MALWARE

Table 5
Lexical features considered by da Luz (2013).

ID	Description
F1 – 3	Mean, variance and standard deviation (1-gram).
F4 – 6	Mean, variance and standard deviation (2-gram).
F7 – 9	Mean, variance and standard deviation (3-gram).
F10 – 12	Mean, variance and standard deviation (4-gram).
F13 – 14	Shannon entropy of the second and third level domain.
F15	Number of different characters.
F16	Number of digits/domain name length.
F17	Number of consonants/domain name length.
F18	Number of consonants/number of vowels.

on these characteristics extracted from the domain name and discarded the network features proposed by da Luz.

Features F1 to F12 are statistical information (namely, mean, variance, and standard deviation) from the 1- to 4-grams extracted from the raw domain name. These features provide interesting results when the domain name is large enough, since the statistical information is more significant than with shorter domains, where just a few N-grams can be extracted, as described in Section 1.

Features F13 and F14 compute the Shannon entropy (Shannon, 1948) for the second and the third level domain. The Shannon entropy measures the randomness in the domain name string, excluding the TLD. As a randomness score, Shannon entropy scores higher for algorithmically generated domain names than for regular domain names, which are usually composed of more human-friendly terms.

Finally, features F15 to F18 describe the balance between vowels, consonants, digits, and string length. Compared to the outcomes of a DGA, natural language has a completely different distribution of vowels and consonants.

In general, all these features are statistical features that aim to measure the randomness of domain names as a way to identify algorithmically generated domain names.

In this paper, we propose using features based on the numbers of *N-gram occurrences* in addition to the lexical, statistical features proposed by da Luz (2013). For instance, when we evaluate a string such as "www.google.com", the feature of the N-gram "ww" has a value of two (since it appears twice in the string), whereas other features like "go" have a value of one.

However, using the occurrence of N-grams as a feature has a huge disadvantage: the number of features that we need to manage increases exponentially. For instance, a 3-gram occurrence approach means that we need to manage a feature for every possible 3-gram. In an alphabet that consists of 36 elements, that means 36^3 additional features. Similarly, the same alphabet for every possible 4-gram means 36^4 additional features. Hence, the computational complexity makes it impossible to manage a model with such a large number of features.

To overcome this issue, in this paper we propose to use *masked N-grams* of the domain names. In our approach, every character is substituted by a symbol representing its character type: a constant is substituted by 'c', a vowel by 'v', a digit by 'n', and any other symbol by 's'. For instance, the website "www.my-website.com" is masked as "cccscscvccvcvscvc". This approach reduces the elements of the alphabet to just 4, thus also reducing the number of combinations of N-grams to 4^N. As a result, considering the aforementioned 3-gram example, the number of additional features would be 4^3 features. This number of features is easily handled by state-of-the-art machine learning models.

Let us remark that, unfortunately, we are obviously losing some of the original information provided by the domain name: N-grams such as "car", "bus", or "yet" are represented by the same masked N-gram, "cvc". However, our approach provides a much higher level of detail than when we only use statistical information.

A combination of the features proposed by da Luz (based on raw domain names) and masked N-Grams (N-Grams from masked domain names) have been used in our experiments in order to evaluate the quality of the new proposed features.

4. Selection of machine learning model and tuning of the algorithm

In this section, we introduce in detail the machine learning model on which we rely and the particular tuning of the algorithm that we have performed.

Decision Tree techniques (Breiman, Friedman, Olshen, & Stone, 1984) have been extensively used in the cybersecurity industry (Dua & Du, 2011; Gandotra, Bansal, & Sofat, 2014; Markey, 2011) due to their good performance: once the model is built, it is a very fast classification model. In the cybersecurity industry, the faster the classification algorithm is, the better, since the threat responses will be performed in a timely manner (as soon as possible once the threat is detected).

A technique that builds a number of decision trees using bootstrapped training samples is Random Forest (Hastie, Tibshirani, & Friedman, 2009). When building such trees, a subset of features is randomly selected from a full set of features every time the tree is split. The final prediction of the algorithm is selected using a voting schema from all the (sub)trees' results. In particular, Random Forest offers a very good generalization, since malware or other malicious threats are usually similar rather than identical.

In this paper, we chose Random Forest as a machine learning model. In particular, Random Forest works well with both numerical and literal features, and it does not require any specific normalization or tuning of the dataset. In addition, it was previously used to address similar problems with good results (da Luz, 2013).

Furthermore, we used Boruta (Kursa & Rudnicki, 2010), a feature selection algorithm based on Random Forest. Boruta creates randomness by means of duplicating features and shuffling their values. Once the Random Forest is built, an importance function evaluates the degree of importance of the original features compared with the artificial ones, thus providing a good insight for relevant feature selection.

The Random Forest algorithm is available in many different machine learning libraries and for many different programming languages. In this paper, we use the R language (R Core Team, 2017) and the Caret library (Kuhn, 2008), which provides an abstraction layer to other well-known, powerful R libraries.

Specifically, we use the `rf` method, a wrapper of the `randomForest` library as implemented in R (Liaw & Wiener, 2002). By default, it uses a forest composed of 500 trees and evaluates the model with a different number of variables randomly sampled as candidates at each split. The number of randomly sampled variables is chosen considering the value that produces the best Receiver Operating Characteristic curve.

Finally, the full (CLEAN + MALWARE) dataset was split into two parts: 60% of randomly selected domain names were chosen for the training dataset, whereas the rest were chosen for the testing dataset. The training dataset was equally split into ten parts in order to implement a k-fold cross validation. This operation was performed three times.

5. Experiments and discussion of results

In this section, we first describe our experiments and the experimental settings. We then discuss our findings.

5.1. Design of experiments

In this paper, we designed three different experiments:

Experiment 1. As a first experiment, we used the (standard) Random Forest classification method as implemented in Caret (i.e., no specific pre-processing or tuning were used). We have evaluated this model from unigrams to four-grams in terms of its accuracy, sensitivity, and performance.

Table 6
Results of the first experiment (standard Random Forest classification).

	1-grams	2-grams	3-grams	4-grams
Number of features	22	**34**	82	274
Training time	0.65 h	**1.21 h**	4.21 h	47.74 h
Testing time	1 s	**1 s**	2 s	4 s
Accuracy	0.9890	**0.9891**	0.9879	0.9873
Kappa	0.9780	**0.9783**	0.9758	0.9747
Sensitivity	0.9896	**0.9859**	0.9857	0.9848
Specificity	0.9884	**0.9924**	0.9901	0.9899
Clean prediction value	0.9884	**0.9924**	0.9900	0.9899
DGA prediction value	0.9896	**0.9860**	0.9858	0.9848

Experiment 2. Here, we applied the Boruta method considering all the features in order to establish a top ranking of the most important features (in terms of classification). This ranking shows us the importance of the masked N-gram features compared with the statistical features described in Section 3.2. As an outcome of this experiment, we generated a new labeled dataset that contains all the statistical features plus features for N-grams of different sizes.

Experiment 3. As a last experiment, we repeated the first experiment but with a reduced set of features. In particular, we considered only those features classified as the most important features by the second experiment. The results of this experiment were compared with the best performance and accuracy obtained from the first experiment.

As a hardware platform, we used a computer with an Intel Core i5 (7600) 3.5 GHz, 40GB RAM DDR4 2400 MHz, and 512GB SSD. As software, R version 3.4.2 was run on top of MacOS X 10.12.7.

5.2. Discussion of results

First experiment. We performed our first experiment for N-grams of different sizes, with N ranging from 1 to 4. We obtained a high accuracy rate, up to 98.91%, with a very low rate of false positives (i.e., a few clean domain names were classified as malicious domains) – around 0.16% for bigram. This misclassification is acceptable from an intrusion detection context.

There are two possible approaches to cybersecurity. When detecting but not blocking threats, such as in an Intrusion Detection System, it is better to detect all threats, even if some legitimate domains are wrongly detected as threats. This is because detections are handled by a group of cybersecurity analysts and no automatic action is taken before the analysts verify that an alert is a true positive. On the other hand, when detecting and automatically blocking threats, such as in an Intrusion Prevention System, it works the other way round. It is better to make sure that every detection is a true positive, even if some malicious domain names are missed, since false positives could create Denial of Service conditions for legitimate users.

Obviously, both the testing time and, especially, the training time are increased when bigger N-grams are used, since the number of features increases and, as a result, more time to converge is needed. Usually, performance decreases in a similar way as accuracy increases when the number of features is increased. However, based on our results, we observe that the best accuracy is obtained from a smaller number of features, corresponding to the 1-grams and 2-grams setup. Table 6 summarizes the results for the first experiment. Training time is expressed in hours (h), while testing time is expressed in seconds (s). The best results are in bold in the table.

Second experiment. With this experiment we aimed to select the most relevant features. We discarded features of four-grams because of the long training time. Furthermore, we created another

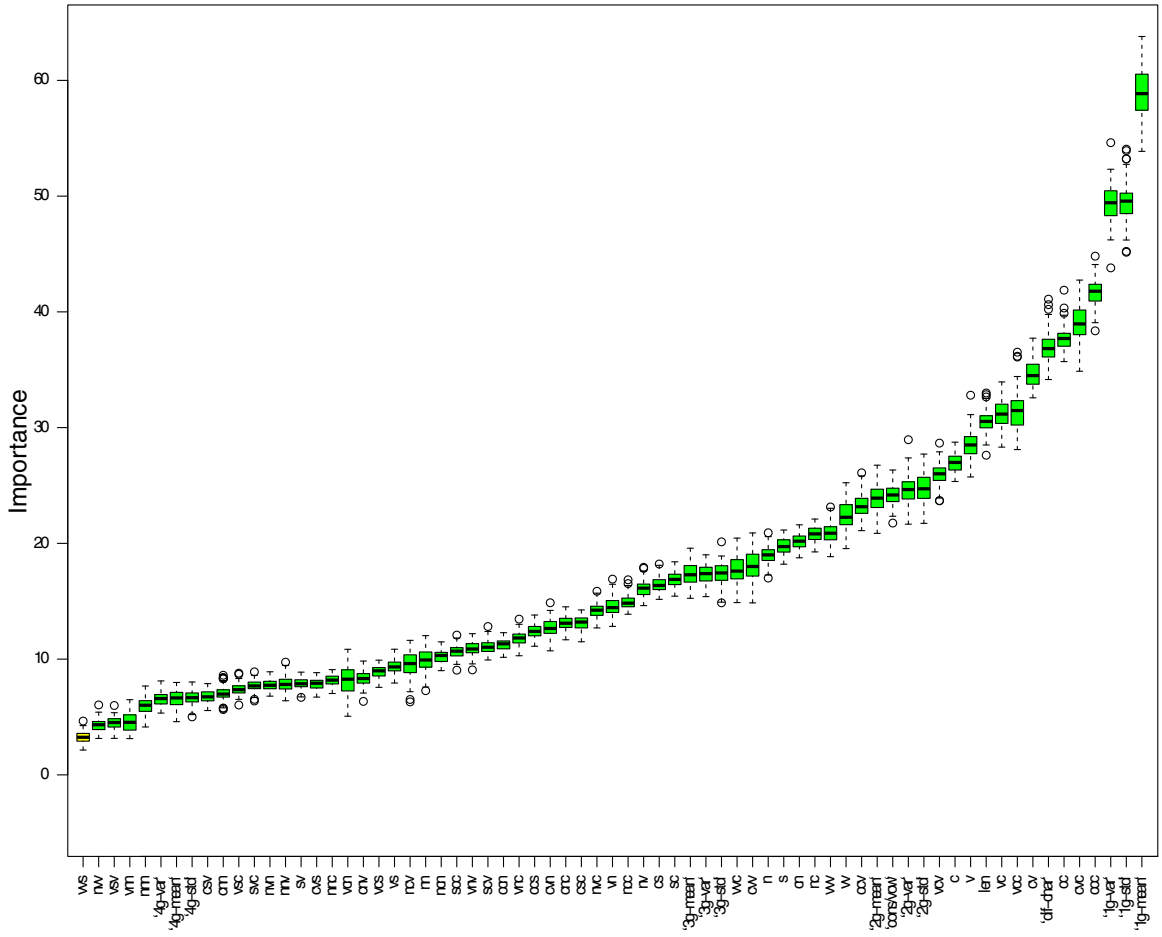

Fig. 1. Boxplot of the results of Boruta feature selection algorithm.

Table 7
Results of the third experiment (considering features by their importance as given by Boruta).

	2-grams	Top 25	Top 20	Top 15	Top 10
Features	**34**	25	20	**15**	10
Training time	**1.21 h**	1.18 h	0.82 h	**0.63 h**	0.44 h
Accuracy	**0.9891**	0.9865	0.9865	**0.9873**	0.9832
Kappa	**0.9783**	0.9730	0.9730	**0.9747**	0.9664
Sensitivity	**0.9859**	0.9851	0.9848	**0.9859**	0.9827
Specificity	**0.9924**	0.9880	0.9883	**0.9888**	0.9837
Clean prediction	**0.9924**	0.9879	0.9882	**0.9888**	0.9836
DGA prediction	**0.9860**	0.9851	0.9848	**0.9859**	0.9828

Table 8
Top 15 features found by Boruta. Masked N-grams are ranked at the bottom (from seventh to fifteenth position).

Rank	Feature
1	Mean (1-gram)
2	Variance (1-gram)
3	Standard deviation (1-gram)
4	Standard deviation (2-gram)
5	Different characters
6	String length
7	**'ccc'**
8	**'cvc'**
9	**'vcc'**
10	**'vcv'**
11	**'cv'**
12	**'vc'**
13	**'cc'**
14	**'c'**
15	**'v'**

dataset combining the lexical and statistical features together with N-grams of different sizes (from unigrams to trigrams) to evaluate their importance in our classification problem.

Fig. 1 plots the results of the Boruta algorithm (we only depict the features with an importance greater than the best random variable created by Boruta). As shown, the feature importance increases linearly up to 15 features. After that point, it starts to increase exponentially. The three most important features are the mean, the variance, and the standard deviation of unigrams, as previously proposed by da Luz (2013). However, more than half of the other features in the top 15 (specifically, nine features) are based on masked N-grams.

Unsurprisingly, the accuracy of the model is reduced when the number of features is reduced. Surprisingly, we have observed that the accuracy is only reduced by 0.5%, while the training time is clearly improved. Specifically, the training process took 7 times less time when reducing from 22 features to the top 5 most important. This performance improvement is usually highly desirable in most real systems. In our case, it is particularly beneficial for two reasons: First, a better performance will enable us to train our model with a larger amount of data. Second, fewer characteristics to extract means a faster classification, which could allow real-time classification and be used in an intrusion prevention approach. A more detailed description of these results is provided in the following experiment.

Table 9
Features based on N-gram statistics from the domain name.

	1-Gram Mean	1-Gram Std	1-Gram Var	2-Gram Std
facebook.com	1.33	0.71	0.50	0.00
wxhyqqrbouru.pw	1.36	0.50	0.25	0.00

Table 10
Features based on other statistics from the domain name.

	Different #chars	Length
facebook.com	9	12
wxhyqqrbouru.pw	11	15

Third experiment. Finally, in our third experiment we considered the features classified as most important following the previous experiment. As a baseline for comparison, we considered the best results in the first experiment (i.e., 2-grams).

Table 7 shows the results for this third experiment. The best results are achieved when the Random Forest method uses only the top most important features as indicated by the Boruta results. Furthermore, we have observed that a Random Forest model using the top 15 features found by Boruta obtains similar results to the same model but using a combination of statistical and 2-grams information. However, the training time in the first scenario decreased by almost half compared to the second scenario.

The top 15 features provided by Boruta are shown in Table 8. As mentioned previously, nine out of fifteen features are based on the masked N-grams that we proposed in this study (bold values in the table). Hence, we conclude that masked N-grams can benefit machine learning models especially during the training phase, maintaining a good trade-off between accuracy and performance.

5.3. Running examples

In order to illustrate how our model works, we have chosen a legitimate domain (facebook.com) and a malware domain (wxhyqqrbouru.pw) from our testing dataset as running examples. Facebook.com is a well-known legitimate domain, whereas wxhyqqrbouru.pw is a DGA generated domain by the Tiny Banker Trojan, also known as Tinba. Both domains were excluded from the training dataset in our experiments, so our model does not have any previous knowledge about them. For this example, we focus on the 15 features which proved to be the most important for our classification problem.

First, as shown in Table 9, unigram and bigram statistics are extracted from the string of the domain name. We calculate mean, standard deviation and variance for the distribution of N-grams. This gives a sense of randomness. Domain names in which each character is used only once will result in a smaller standard deviation and variance than domain names where some characters are repeated.

Second, other statistics not related to N-grams (see Table 10) are extracted: the number of unique characters and the domain name's total length. These features are also extracted from the string of the domain name and, together with the previous ones, represent a subset of the features previously used by da Luz and other authors.

Finally, we generate the masked domain name as explained in Section 3.2, and we extract from it a set N-grams (sizes between one and three) that have been proven to be good features for our classification problem. We count how many times each N-gram appears in the masked domain name, as shown in Table 11. For example, it can be seen that facebook.com (masked to cvcvcvvc.cvc) does not contain any substring composed of three consonants, so the masked N-gram feature "ccc" is zero. Similarly, the domain name wxhyqqrbouru.pw (masked to ccccccccvvcv.cc) contains up to 6 substrings composed of three consonants, so the "ccc" feature is 6. This feature is representative for this example, since it is easy to see that a string with too many consonants together does not look like natural language to the human eye.

6. Conclusions

Malware is normally detected by capturing communications with malicious servers in network traffic. In recent years, malware has adopted a stealthier communication strategy in order to remain undetected in a compromised system: instead of communicating with fixed IP addresses or domain names, it algorithmically generates domain names that host its malicious servers.

In this paper, we have proposed a set of lexical features based on masked N-grams of the domain name as a way to detect algorithmically generated domains. In masked N-grams, every character of the given string is substituted by a character that represents its type (i.e., consonants as 'c', vowels as 'v', digits as 'n', or other symbols as 's'). Furthermore, we have evaluated masked N-grams with other lexical features using Random Forest as a machine learning model and Boruta as a feature selection algorithm. Our findings show that masked N-grams are good features for the detection of algorithmically generated domain names. In particular, bigrams and trigrams representing combinations of vowels and consonants (such as "cc", "ccc", or "vcv", to name a few) were found to have a high importance for classification. However, the three most important features are still based on the statistical features of unigrams (namely, mean, variance, and standard deviation).

Our results also shown that using N-grams of a fixed size considerably increases the number of features, as well as the training time. Furthermore, the detection accuracy is decreased. We found that a combination of lexical and statistical features together with bigrams achieved the best results in terms of accuracy. Nevertheless, a selection of the 15 most important features, which combines statistical features and masked unigrams, bigrams, and trigrams, gave very similar results with much better performance.

For the sake of reproducibility, we have publicly released the dataset that we built for the experimentation. This dataset contains

Table 11
Features based on N-gram from the masked domain name.

Domain name	Masked domain	ccc	cvc	cc	cv	vcc	vc	v	c	vcv
facebook.com	cvcvcvvc.cvc	0	3	0	4	0	4	5	6	2
wxhyqqrbouru.pw	ccccccccvvcv.cc	6	0	8	2	0	1	3	11	1

32,000 algorithmically generated domain names from real malware families and the same quantity of legitimate domain names. Furthermore, we have classified these malware families according to their domain name generation algorithms.

Authors' contributions

J. Selvi

- Conception and design of study
- Acquisition of data
- Analysis and/ or interpretation of data
- Drafting the manuscript
- Revising the manuscript critically for important intellectual content
- Approval of the version of the manuscript to be published

R. J. Rodríguez

- Conception and design of study
- Analysis and/ or interpretation of data
- Drafting the manuscript
- Revising the manuscript critically for important intellectual content
- Approval of the version of the manuscript to be published

E. Soria-Olivas

- Conception and design of study
- Analysis and/ or interpretation of data
- Drafting the manuscript
- Revising the manuscript critically for important intellectual content
- Approval of the version of the manuscript to be published

Acknowledgments

The research of Ricardo J. Rodríguez was supported in part by the University, Industry and Innovation Department of the Aragonese Government under *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo research group, ref. T21-17R).

References

29A Labs website. (1995). Accessed on October 02, 2016. [Online; http://vxheaven.org/29a/].

Alexa 1 million. (2016a). [Online; http://s3.amazonaws.com/alexa-static/top-1m.csv.zip]. Accessed on October 02, 2016.

Alexa top sites. (2016b). Accessed on October 02, 2016. [Online; http://www.alexa.com/topsites].

Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. (2012). From throw-away traffic to bots: Detecting the rise of DGA-based malware. In *Part of the 21st USENIX security symposium (USENIX security 12)* (pp. 491–506). Bellevue, WA: USENIX.

AV Test (2017). Malware statistics & trends report. Accessed on February 23, 2017. [Online: https://www.av-test.org/en/statistics/malware/].

Bader, J. (2016). Some results of my DGA reversing efforts. Accessed on October 02, 2016. [Online: https://github.com/baderj/domain_generation_algorithms].

Bilge, L., Kirda, E., Kruegel, C., & Balduzzi, M. (2011). EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proceedings of the network and distributed system security symposium, (NDSS 2011)*.

Bilge, L., Sen, S., Balzarotti, D., Kirda, E., & Kruegel, C. (2014). Exposure: A passive DNS analysis service to detect and report malicious domains. *ACM Transactions on Information and System Security, 16*, 14:1–14:28.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth and Brooks.

Dua, S., & Du, X. (2011). *Data mining and machine learning in cybersecurity* (1st ed.). Boston, MA, USA: Auerbach Publications.

Europol (2013). EU serious and organised crime threat assessment. *Techreport*. European Union Agency for Law Enforcement Cooperation (Europol).

Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security, 5*, 56–64.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning. *Springer series in statistics* (2nd ed.). New York, NY, USA: Springer.

Kaspersky Lab (2014). Kaspersky security bulletin 2014. [Online; http://securelist.com/files/2014/12/Kaspersky-Security-Bulletin-2014-EN.pdf].

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software, 28*, 1–26.

Kursa, M. B., & Rudnicki, W. R. (2010). Feature selection with the Boruta package. *Journal of Statistical Software, 36*, 1–13.

Liaw, A., & Wiener, M. (2002). Classification and regression by random forest. *R News, 2/3*, 18–22.

da Luz, P. M. (2013). *Botnet detection using passive DNS*. Department of Computing Science, Radboud University, Nijmegen masters thesis.

Markey, J. (2011). Using decision tree analysis for intrusion detection: A how-to guide. *Technical Report*. SANS Institute.

Mockapetris, P. (1987). RFC 1035: Domain names - Implementation and specification. *Technical Report*. Internet Engineering Task Force. Available at http://www.rfc-editor.org/rfc/rfc1035.txt.

Porras, P., Saïdi, H., & Yegneswaran, V. (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*. Berkeley, CA, USA: USENIX Association. 7–7.

R Core Team (2017). *R language definition*.

Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., et al. (2012). Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE symposium on security and privacy* (pp. 65–79).

Schiavoni, S., Maggi, F., Cavallaro, L., & Zanero, S. (2014). Phoenix: DGA-based botnet tracking and intelligence. In *Proceedings of the 11th international conference on detection of intrusions and malware, and vulnerability assessment (DIMVA)* (pp. 192–211). Cham: Springer International Publishing.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal, 27*, 623–656.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.