Popularity-based Detection of Domain Generation Algorithms

Jasper Abbink and Christian Doerr
Delft University of Technology, Cyber Security Group
Mekelweg 4
2628CD Delft
j.abbink@student.tudelft.nl,c.doerr@tudelft.nl

ABSTRACT

In order to stay undetected and keep their operations alive, cyber criminals are continuously evolving their methods to stay ahead of current best defense practices. Over the past decade, botnets have developed from using statically hardcoded IP addresses and domain names to randomly-generated ones, so-called domain generation algorithms (DGA). Malicious software coordinated via DGAs leaves however a distinctive signature in network traces of high entropy domain names, and a variety of algorithms have been introduced to detect certain aspects about currently used DGAs.

In this paper, we look ahead and evaluate the utility of today's detection mechanisms if botnets make the next obvious evolutionary step, and replace domain names generated from *random letters* with randomly selected, but actual *dictionary words*. We find that the performance of state-of-the-art solutions that rely on linguistic feature detection would significantly decline after this transition, and discuss an alternative novel approach to detect DGAs without making any assumptions on the internal structure and generating patterns of these algorithms.

KEYWORDS

malware, domain-generation-algorithm, threat intelligence

ACM Reference format:
Jasper Abbink and Christian Doerr. 2017. Popularity-based Detection of Domain Generation Algorithms. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 8 pages.
https://doi.org/10.1145/3098954.3107008

1 INTRODUCTION

Malware and specifically botnet infections are repeatedly identified as one of the major enablers of cyber crime. Symantec [19] reports that about 40% of all distributed denial-of-service attacks (DDoS) originated from botnets that offer their services on underground market places, and Demarest [7] estimates that approximately 500 million computers are infected per year world-wide.

In order to control this army of compromised computers, botnet owners typically establish a command-and-control (C&C) server to which hosts report, from where they receive a list of tasks to execute, and to where they upload material they gather. As this C&C server is publicly exposed and an obvious point-of-failure in the coordination of the botnet operation, a large share of botnet mitigation has focused on taking down C&C infrastructure and suppressing its communication channel. This has led to a co-evolution of new means to hide and detect infrastructure and control channels in a cat-and-mouse game.

ARES '17, August 29-September 01, 2017, Reggio Calabria, Italy

ACM ISBN 978-1-4503-5257-4/17/08…$15.00
https://doi.org/10.1145/3098954.3107008

Evolution of Botnet Control

A basic problem in the deployment of botnets that malware authors face is that they need to find a way so that the distributed clients can find and connect to the C&C server, and although the address is known to anyone in possession of the malware the channel should not be easily disrupted by network owners or law enforcement. This is for example the case with the most simple way of linking clients to the C&C server by means of a static and hard-coded IP address, which could trivially be blacklisted in firewalls or taken out of circulation by a hosting provider on request from the authorities.

In order to be able to dynamically update the location of the C&C infrastructure, botnets are hence finding the control server based on domain names. Although a domain can also be seized by law enforcement, the process takes significantly longer, as it may require cross-border actions. To further limit this angle, botnet owners are also not relying on a single static domain name in their client software, which after being seized would leave the entire system inoperable, but rather dynamically generate domain names on the fly.

The domains at which the C&C server is contacted are computed using a domain-generation algorithm (DGA) based on the current time or some other information publicly available across all hosts, each only valid for a short amount of time. In addition to the short validity, DGAs frequently generate also hundreds of candidate domains per time interval. Infected hosts lookup all candidate values to find the domain that was actually registered by the botnet owner, which from a defense perspective makes this mechanism very difficult and costly to suppress, as registering, seizing or sink holing thousands of domain names per day is often too administratively complex, costly or not scalable.

Current DGAs generate domain names by concatenating random letters and morphemes. This has the clear advantage that the resulting domain names are with very high likelihood available, but has the drawback that they leave in the network a characteristic trace of a series of unsuccessful lookups (NXDomains) for domain names, each with a high entropy name such as yfewtvnpdk.info or rwyoehbkhdhb.info that is unlikely requested by normal users.

There exist a number of algorithms to detect these lookups, which stand-out significantly from other DNS traffic, and thus are highly effective in locating this irregular traffic. Existing approaches for example find linguistic deviations in contacted domain names, check whether (parts of) the domain names are listed in a dictionary or assume that infected hosts behave exactly the same with regards to timing of DNS requests. Given the high mitigation performance, it can however be expected that botnet owners will transition to the next stage in DGA evolution as soon as today's countermeasures are widely enough applied to derogate from the malware owners' bottom line, as it was the case with previous developments described above. The bulk of these algorithms relies on entropy or word list measures in some form or the presence of periodicity in requests as part of their detection strategy, the obvious transition to make is to concatenate random words from a dictionary instead of concatenating random letters, and add some variation in time when these lookups are made.

In this paper, we show that a DGA generating names like sunnytailorballoons.info instead of rwyoehbkhdhb.info would significantly interfere with existing measures based on linguistic feature detection. These generated names also appear less suspicious in casual inspection, hence alternative approaches are needed that remove implicit reliance on the structure of the generated domain names. Furthermore, if lookups are made with some random delay, the resulting patterns are no longer strong enough to be reliably detected in a large scale network.

This paper makes two main contributions:

- We evaluate the performance of existing DGA detection algorithms if domains would no longer be consisting of randomly assembled letters, but rather fall back on concatenating normal dictionary words, which no longer stand out from visible inspection against normal DNS traffic nor from the perspective of entropy. The detection performance is also negatively affected if requests are no longer strictly periodic.
- We show that DGAs in a large network can be efficiently found by detecting sudden rises and declines of popularities of domain names.

The remainder of this paper is structured as follows: section 2 discusses related work to detect botnet infections based on anomalous traffic patterns. Section 3 describes the collection of the data set and safeguards taken to maintain the privacy of users. Section 4 introduces our proposed approach, which is compared in section 5 in terms of detection performance against previous work. Section 6 concludes the work and summarizes our findings.

2 RELATED WORK

According to Feily et al. [8] botnet detection techniques can be classified into two major classes: signature-based and anomaly-based. The first works by finding and blacklisting known malicious signatures in network traffic. This approach can be compared to traditional locally installed malware detection mechanisms trying to detect specific signatures in executables [20]. Wurzinger et al. [22] applied these techniques to detect specific Internet Relay Chat (IRC) botnets by scanning for specific byte patterns in the network traffic. Detecting botnets with signatures, if the traffic is unencrypted, is straightforward, but requires a continuous effort to keep the database of signatures updated with the ever increasing amount of malware, that might behave slightly different compared to all previous instances. This reduces the practical applicability, as by definition, the actors operating a botnet (botmasters) are always one step ahead of being detected.

The second class, anomaly-based detection, tries to detect malicious applications (including botnets) based on behavioral analysis. Binkley et al. [3] create a system to automatically cross match characteristics of IRC messages with TCP SYN packets (commonly used for denial-of-service attacks) in order to detect botnet channels and infected hosts. Livadas et al. [12] compare multiple classifiers on network flows of IRC traffic to subsequently distinguish legitimate chat messages from botnet traffic. Strayer et al. [18] see problems with botnet detection techniques analyzing IRC traffic on a static port (6667, the default for IRC) and only responding after an attack has been executed. Therefore they propose a solution that detects patterns in network flows. Specifically, used bandwidth, timings of packets and burst duration are measured. The proposed model can detect IRC botnet traffic running on any port without analyzing all messages directly.

Gu et al. [9] also use network flow analysis to detect botnets, but instead of detecting specific IRC botnets, they try to correlate network flows of multiple clients in a network to detect similar traffic. This relies on the assumption that multiple instances of the same malware will behave the same and show similar network patterns. In addition to this, AsSadhan et al. [2] see complications in the scalability of analyzing network flows of single hosts and approach this problem with aggregated flows of multiple hosts. Singh et al. [16] see the amount of data generated by network-based detection systems as a challenge. Therefore they propose a system that analyzes full network traces in a horizontally scaling manner. Specifically they apply their system on detecting peer-to-peer botnets. This is done by automatically creating a Random Forest Decision Tree and applying this model to the captured data. BotHunter [5] uses a correlation engine combining data of three points in a network. The goal is to detect the common botnet infection life cycle (inbound scan, exploit usage, malware distribution, outbound bot coordination, outbound attack propagation) and determine new infections within a network. Problems with this technique include botnets being very obvious in their communication and the authors expect botmasters to circumvent (parts of) this system by scanning in a stealthy manner or by using encrypted communications to the C&C.

Historically, bots had a hard-coded IP address or domain name implemented in the malware binary. Blocking communication to a single IP address or domain name is usually a trivial task for law enforcement or network administrators and botnets utilizing this technique often do not exist for long periods of time. To circumvent getting taken down, while still using the ease of HTTP (instead of moving to a distributed peer-to-peer system), bot masters started to pseudo-randomly generate domain names based on a deterministic seed. This seed is often the current date, but can also be non-predictable public information (for example "trending topics" of twitter.com). This way, a bot would try to automatically connect to a new (set of) domain(s) every day which would make it harder to take the botnet down. In the case that these algorithms

create more than a single domain per day, it would also be very expensive for third parties to pre-register all domains in order to take over a botnet. When no public data is used as seed for the DGA, it is also impossible for law enforcement and researchers to predict domains that will be in use in the future.

As these DGAs heavily rely on the Domain Name System (DNS), a subset of anomaly detection focusses solely on detecting malicious applications based on this traffic. Thomas et al. [21] proposes to analyze NXDomain DNS traffic "at several premier Top Level Domain (TLD) authoritative name servers". They do this by calculating the Jaccard index of two sets of recursive name servers A and B requesting two different domains X and Y. According to the authors, the Jaccard similarity for NX domain responses of the same DGA is often above 0.9 and domains of botnets can be detected in that way. PsyBoG [11] approaches the problem of botnet detection by assuming that botnets have periodic and simultaneous behavioral pattern. To reduce the data to be processed, only DNS traffic is used. In order to find the periodicity of a host, all DNS traffic is stored with the requesting IP address and timestamp. Then a periodicity analyzer determines the power spectral density (PSD) [17]. Significant peaks on similar frequencies over multiple hosts then indicate a botnet infection on these hosts, as normal traffic would not result in a deducible periodicity.

Another approach to this problem comes from liguistical analysis of queried domain names. Antonakakis et al. [1] assume that computers in a network, infected with the same malware, will generate similar DNS queries that result in similar NXDomain responses. This is done by calculating 33 statistical features per DNS request returning a non-existent domain error response (NXDomain) and using a combination of the X-means clustering algorithm [14] and a Hidden Markov Model to detect malicious domains, and subsequently classify them into clusters belonging to the same botnet. Schiavoni et al. [15] created a system called Phoenix, which uses a set of linguistic features on the domain name to initially determine whether or not it might be a legitimate domain. The baseline for the legitimate classification is built from the top 100000 Alexa[1] domains. After this, domains get marked as illegitimate when the Mahalanobis distance [13] to the centroid of the cluster of the legitimate domains exceeds a fixed threshold.

A combination of both behavioural and linguistic detection mechanisms can be seen in Choi et al. [6] system, the BotGAD framework, which collects all DNS traffic passing through. It stores this traffic with the requesting IP address and the corresponding timestamp. Per domain name, BotGAD creates a matrix of timeslots and requesting IP addresses. To find computers that are infected with the same malware, the cosine similarity coefficient [23] is calculated in order to determine the similarity of these matrices. Then the data is enriched to calculate lexicological features and the distinctive features of the DNS queries. For DNS features, BotGAD's focus is mainly on the Time To Live (TTL) of the DNS records, information about the autonomous system (AS) and hosting country (resolved by the database of MaxMind[2]). A short TTL is a good indication for Dynamic DNS (DDNS) [4] or Fast-Flux Service Network (FFSN) [10] usage.

[1]http://www.alexa.com
[2]http://dev.maxmind.com/geoip/legacy/downloadable/

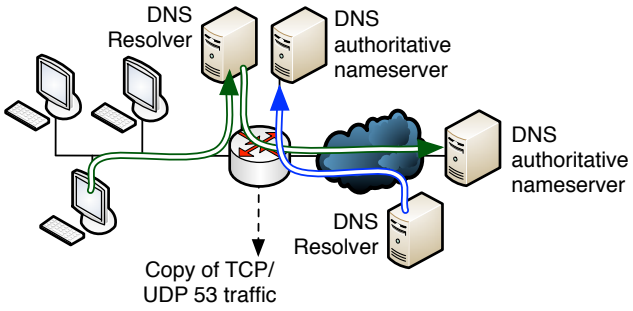

Figure 1: DNS queries and responses were collected through port mirrors on the core routers, thereby intercepting both internal and external request/response pairs from internal hosts and external DNS advertisers and resolvers.

3 DATA COLLECTION

The evaluation of anomalies using a synthetic evaluation context runs the risk that an algorithm's performance is heavily influenced by the rules and procedures used for creating the evaluation dataset. For example, as benign and malicious traffic is generated and merged to create a synthetic data set, the way how DNS requests are created may greatly influence the performance of the detection algorithms. For example, as an algorithm such as PsyBoG will consider periodicity as part of its detection strategy, the type and configuration of an artificial user model that generates network requests may or may not contain certain features, thereby greatly influencing the evaluation results.

Thus, in order to create a realistic evaluation scenario, we will test the DGA detection algorithms in this paper against an actual snapshot of DNS traffic obtained from a research network backbone. This section discusses the method used for collecting and anonymizing the dataset to safeguard the privacy of the network's users.

3.1 Data Acquisition

In order to eliminate potential biases and artifacts from a synthetic workload and network traffic, we developed an evaluation scenario based on the actual DNS traffic generated by a population of 22,329 hosts. The data trace was obtained by installing a filtering rule on the core routers in a research network that would deliver a copy of any TCP and UDP traffic on port 53 leaving and entering the network as well as in between the network's organizational zones. Figure 1 shows the abstract topology of the network.

The filter on the core router would hence see every DNS request leaving the client network either towards the network's DNS resolvers (or the Internet if the client were reconfigured to use an alternative recursive resolver), packets querying the authoritative name servers hosted within the research network, as well as any recursive lookups from the network's internal resolvers to other name servers on the Internet. The analysis presented in this paper spans a time period of 14 days, in which a total of 612,011,001 DNS lookups and responses were recorded.

3.2 Data Anonymization

As DNS lookups precede the bulk of network traffic and operating systems conduct minimal caching of responses, the sequence of DNS requests provides a highly accurate view of the activities of a client and thus the individual user. This makes it critical to anonymize the data sufficiently to safeguard the privacy of the users, this section will describe the anonymization protocol used, which was developed with the research network's privacy officer.

Recall from figure 1 that the collection point at the core router provided three basic types of DNS traffic flows: first, DNS requests from the local area network to the network's internal resolvers and soon after responses from the internal resolvers to the internal clients, second, traffic between the internal recursive resolvers to external authoritative DNS name servers, and third, – in case of a lookup of a DNS record somewhere else in the Internet that is hosted by the research network – traffic between external DNS resolvers and the network's own authoritative name servers. This means that we can characterize the network into three zones, (a) the internal network of clients, (b) the DNS servers hosted in the research network, and (c) any host external to the network.

We see that traffic flows from (b) to (c) are not privacy-sensitive. The authoritative name servers for DNS zones are public knowledge, the recursive lookups contain no connection to a local machine and the recursive DNS lookups aggregated over tens of thousands of simultaneous users do not allow the identification of an individual. Requests from zone (c) to (b) and even more drastically from zone (a) to (b) may however be linked back to an individual person. Local laws and policies stipulate that such a linkage is established via the IP address as a personal identifier (PII), but application of a non-reversible transformation of this function is sufficient, as it prevents a lookup of a specific activity and assignment towards a concrete PII and person. If the transformation is deterministic, this provides the added benefit for cross-correlating sequences of lookups – as can be expected from a DGA – while ensuring the anonymity of the originating user.

Such non-reversible but deterministic transformation is trivially accomplished using a hash function, in our implementation SHA-256. However, since the set of IP addresses is known and denumerable, a basic hash of the IP address does not suffice. Instead, a unique salt was generated for each IP address x and each IP address on the internal network and the Internet hashed together with its unique salt $s[x]$ to obtain the anonymized $y \leftarrow \text{hash}(s[x]||x)$. This procedure guarantees that given an anonymized record y it is not possible to find the corresponding IP address x, and as well as given a known IP address x it is not possible to determine the associated DNS records in the dataset. In the reverse, given a domain name we can lookup an anonymized list of hosts that have requested the record but are not able to link it to a specific user.

As we aim to find infected hosts on the internal network, it is essential to know between which zones a DNS request and response actually flows. To preserve information about the location of a host in the anonymized IP address, the highest bit of the network address is set to 1 if the address was part of the internal network (zone (a)), a 0 indicating a position somewhere on the Internet (zone (c)). A regular DNS lookup to an external domain would thus show the flow pairs {(1??????? → network-resolver, domain name), (network-resolver → 0???????, domain name)}, while a local client trying to bypass the internal recursive resolver will leave the record (1??????? → 0???????, domain name). Although the above procedure will slightly increase the chance for collision in comparison to the collision probability of the hash function itself, experimental evaluation has determined this issue to be irrelevant in practice. The original data and salts were maintained and processed by a separate party, and only the anonymized traces were used in this research.

4 METHODOLOGY

The key differentiating part of the algorithm presented in this paper is that we make no assumptions on the structure and mechanics of the domain name candidates used within the DGA, i.e., domain names can be generated in any arbitrary way including dictionary words. As it is the very nature of domain generation algorithms to dynamically generate names – as static domains would otherwise be trivially blocked –, we will use this fact that they are only used for a short period of time as a strategy to detect them. Our method will exploit that DGA-based DNS traffic has to be transient to be effective, hence we calculate the popularity of domain names and detect sudden increases and decreases of traffic over multiple days. This allows for a very simplistic, yet efficient method in detecting DGA anomalies. A drawback of this method is that detection of a new botnet can only happen after the first day of infection.

4.1 Parsing data

DNS traffic enters the system as packet captures (pcap) containing all packets the name servers encountered over port 53 (used for DNS). Each file contains one hour of traffic and is parsed with `libpcap`. Processing is done by iterating over all packets in a given pcap file. As the pcaps contain more than just DNS traffic, we start by filtering out these unwanted packets. The pcaps not only contain the requests from within the network to the network's name servers, but also from these name servers to the external authoritative name servers, and external requests about names that these name servers are authoritative for. Since we are only interested in detecting infected hosts from within the network, we discard the traffic to and from external sources. We also completely ignore all traffic where our name servers replied with authoritative answer (AA) flag, as we assume that we do not facilitate domain names used for DGAs ourselves. For all remaining requests, we store the requesting IP address, the affected domain name and the exact time in memory. After iterating over all packets in a pcap, we reduce the accumulated data to only store a mapping of IP addresses requesting specific registered domain names on a given day (without the exact time).

Content Delivery Networks (CDNs) and other load-balanced services create a large amount of fully qualified domain names, where each domain name is only requested a few times across all network hosts. In order to reduce the amount of noise and fluctuation and match requests which in practice functionally belong together, we only use the name of the registered domain to significantly reduce the amount of data as a DGA may chose to prepend more data to lower levels of the domain name. In practice we use Mozilla's

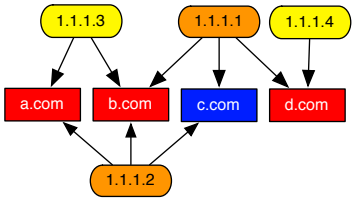

Figure 2: Method used to find IP addresses I' broadly related to a domain D. In this figure, D is indicated in blue, D' in red. The elements of set I are colored orange, I' yellow.

Public Suffix List[3] and take one extra level (e.g., test.example.org gets reduced to example.org). From this dataset, we calculate the total amount of requests to a specific domain on an hourly basis. Finally, when all files have been processed, we further reduce this data to store the amount of requesters per domain name for every day encountered in the original data set.

4.2 Filtering data

After importing all data, we filter the list of domain names to only include those that are requested on one to three consecutive days and not before or after that. This rule hooks on the transient nature of a DGA, the algorithms employed by the infected hosts will at some point start to generate a particular domain and as the seed or variables progress stop to do so. At the same time, this rule also ensures that the detection is insensitive to spontaneously emerging trends, for example requests triggered in response to an advertisement campaign for a new product or sudden interest due to a political or sport event. While certain triggers may spark a sudden interest in a topic and a peak in DNS lookups for the associated names, the requests from all hosts will not emerge and cease across all clients in a synchronized fashion, some users will have known about the product, sports event etc. before popularity set in, others will continue to visit these sites after the hype has vanished.

From these remaining domains, we look up all IP addresses that requested similar domains in the following way. Given a domain D:

(1) Find all IP addresses I that requested a domain D
(2) Find all domains D' requested by all IP addresses in I
(3) Find all IP addresses I' that requested domains in D'

A visual representation of this process is given in figure 2. The reason we apply this method is not to miss any domains contacted by a DGA contacting domains in a locally random order which might not get contacted by all infected hosts. If we would not do this, these domains would still get recognized as malicious, but cannot be clustered together as belonging to the same botnet afterwards. While this allows to differentiate between single and multi-infections on individual hosts, this post-processing step has the drawback that it can include unrelated domains into a cluster consisting of DGA domains. If the differentiation into individual botnets is not needed, this post-processing may be dropped.

[3]https://publicsuffix.org/

4.3 Finding clusters

After collecting all broadly related IP addresses per suspicious domain name, we need to determine how distinctive a particular pattern is across the entire population. If A denotes the subset of hosts requesting a suspicious domain name across all hosts B, we are thus interested in $\frac{|A \cap B|}{|A \cup B|}$, which is denoted as the Jaccard similarity between A and B. We leave the Jaccard similarity as an adjustable threshold in our algorithm, as the clustering of domain names depends on the similarity of the requesting hosts within the network. Our assumption is, that multiple botnet infections in a network do not cover the same hosts, and therefore we want to determine which suspicious domain names are probably related to the same DGA.

In order to limit the amount of output, we make use of two more variables: the average amount of unique hosts requesting a domain in a detected cluster and the minimum of hosts requesting a domain required for it to be included in a cluster. The rationale for this is to limit the output to clusters that are requested by multiple hosts within the system, rather than focussing on single hosts.

5 EVALUATION

In order to evaluate our system, we tested our algorithm on 14 days of DNS traffic generated from 22,329 internal hosts. Establishing a baseline truth of all true positives is a hard task, as after all it would necessary to have a complete and labeled set of all malicious domain names currently in use by botnets. As an alternative approach and in order to preserve the statistical characteristics of the regular user traffic, we classified a random subset of all hosts as "malicious", and for these hosts change some of their DNS requests to query domain names generated by one or more DGAs. Each malicious bot would request, in a locally random order, up to 100 domains per day with at least one domain being requested by all hosts. We applied this experiment with different infection rates of the network, specifically 0.02%, 0.1%, 0.5%, 1.0% and 5.0%. Assuming for now the baseline to be clean of malware, we can thus evaluate how much of the now known DGA instances are correctly detected by the algorithms, and how accurate the detected clusters represent the entirety of the botnet.

Jaccard similarity

As discussed above, we chose the Jaccard index as a metric for comparing the similarity of the request patterns of hosts in the network. Depending on the amount of infected hosts (and in case the lookup patterns of the population are highly diverse), it is necessary to adjust this variable to correctly classify two domains A and B together as belonging to the same botnet. For this we leave the two variables to filter clusters to low values to not filter out any potential clusters already.

In figure 3 and figure 4 we can see the precision, the ratio of true positives over the amount of all elements predicted as positive by the algorithm, and recall, the ratio of true positives over all positive elements or the true positive rate, of the detected cluster(s) containing our test botnet. As can be seen in the figures, the algorithm is fairly robust with respect to the Jaccard similarity and in terms of the percentage of infected hosts in the network. We find that

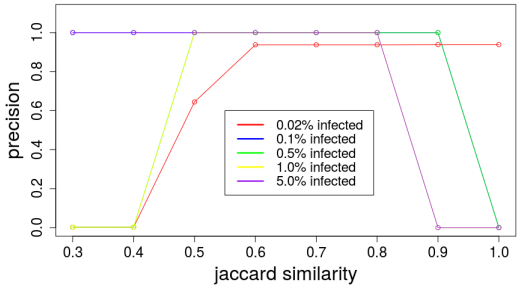

Figure 3: Precision for different thresholds for Jaccard similarity and the size of the botnet

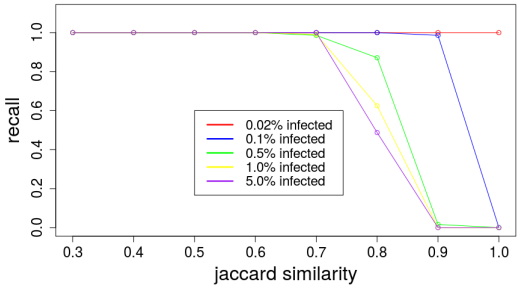

Figure 4: Recall for different thresholds for Jaccard similarity and the size of the botnet

a threshold within the range of 0.6 and 0.7 provides the highest precision or positive predictive value and no unidentified elements.

If we were to tune this value outside of the optimal corridor, botnet detection will either begin to fuse or become fragmented. For the smallest set of infected hosts, 0.02%, even higher Jaccard similarities result in a high precision and recall, but the algorithm detects the domains contacted by the hosts as multiple different clusters, whereas starting from 70%, the injected domains are detected as a single cluster. As can be seen, using a lower threshold for the Jaccard similarity results in significantly lower precisions in the detected cluster for small groups of infected hosts. With larger infected parts of the network, this has a smaller impact on the precision. Figure 5 shows the Jaccard similarities between all domains in the injected traffic. From this, we can also see that in order to detect all domains belonging to the same injected set of domain names, we must allow use a threshold for the Jaccard similarity around 60% to 70%. In case of a DGA assembling candidate domains from random words, this result is expected as domains will follow the similarity patterns of normal word lists.

Detection Performance for Random Letter and Random Word DGAs

As some other algorithms are very time and memory consuming to run, we limited our comparison to other algorithms based on data of a single day. On this day, we injected DNS traffic for 1% (220) of the hosts with either domains generated by the Locky ransomware[4],

[4]https://dgarchive.caad.fkie.fraunhofer.de/site/families.html

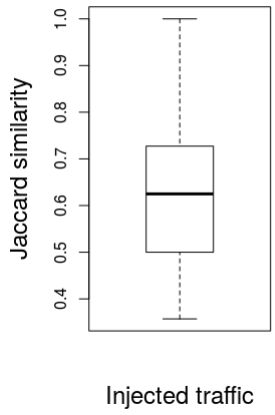

Figure 5: Jaccard similarity between all injected domains for 0.02% infected hosts

which generates domain names based on the random characters (e.g., hrgcmmihpxth.in) and easily stands out within regular DNS traffic. As an alternate condition, the same infected hosts requested malicious domains generated by concatenating random words from the English dictionary.

In order to mimic reality, we injected the traffic as if the botnet would query 200 domains in a locally random order, with 1 request exactly every 10 minutes. Once a host reached the domain that would be registered and return a valid DNS answer, it would continue to request that domain until the end of the day.

Table 1 shows an overview of detection rates of the four evaluated algorithms, for the case of Locky and a word-based DGA. The domains and host infections were randomly seeded and placed, and the results shown are the arithmetic average over 10 runs. One step in the Pleiades [1] algorithm relies on a spectral clustering of domain names, which in turn necessitates a Eigenvalue decomposition of an incident matrix. Standard algorithms for this step have a complexity of $O(n^3)$, and while only taking into account NXDomain traffic, the time and memory intensity of this algorithm makes it hard to apply in practice. In fact, in the tests we tried run on the dataset of a full day, it could not finish on a 24-core server with 200 GB of RAM for the dataset of 22,000 hosts, and in smaller sample sizes, it only managed to find clusters related to reverse DNS lookups. As it heavily relies on linguistics, it can be expected to fail to detect domains generated by a DGA specifically trying to evade this. In order to enable a comparison, we hence also create a smaller synthetic evaluation scenario which will be discussed below.

Phoenix [15] works in three steps, of which the first one tries to determine whether or not a domain might be suspicious. After applying this step to all traffic from the selected day, Phoenix already filtered out more than 60% of the Locky and natural language domains, while still leaving 39% of all domains occuring in the dataset. As the detection of Phoenix is built against a baseline of actual domain names (the Alexa 100000 list), we see a drop in detection of

		Linguistic-based		Periodicity-based	
detection rates	our algorithm	Pleiades	Phoenix	BotGAD	PsyBoG (hosts)
detected Locky domains	200 (100%)	-	77 (38.50%)	1 (0.50%)	125 (56.82%)
total detected with Locky domains	72282	-	271686	68965	7424
detected natural language domains	200 (100%)	-	46.5 (23.25%)	1 (0.50%)	125 (56.82%)
total detected with natural language domains	72282	-	271655.5	68965	7424

Table 1: Detected locky domains and natural language domains on the compared day of traffic containing 707826 **different domains. The detected domain by BotGAD is the "active" DGA domain, contacted by all infected hosts. Domains were generated with a seeded random number generator and results were averaged over ten runs.**

about 40% in detection with the transition from random letters to random words.

In contrast to Pleiades and Phoenix which heavily rely on linguistic features in their detection, BotGAD and PsyBoG mainly build on the synchronization and periodicity of requests. BotGAD [6] also filters domains before attempting to cluster domains together. After this step, it still retained all of the domains detected by our algorithm, but it also marked 73% of all domains as possibly suspicious. After this, it would further reduce the set of detected domains based on linguistic features of the domain names and features of the DNS query and answer itself. Furthermore it relies on short periodicities of DNS traffic over multiple hosts and it needs to be fine tuned to a time window, where a larger time window results in significantly less accuracy, and a smaller time window could miss suspicious domains (as a bot might not use DNS in a periodic manner). PsyBoG [11] detects infected hosts by finding periodicities in the timing of DNS requests issued from any given host in the network. As the inserted traffic between the Locky domain names and natural language domain names are on the same times, the results do not differ.

Until now, we have made the assumption of the baseline dataset to be clean of malware, which allowed us in the evaluation of the four algorithms to benchmark against a known reference. Without further filtering, our algorithm detected 7041 additional clusters in the original dataset. We manually inspected these and tried to find actual malicious domain names in these and found 18 suspicious sets, based on the randomness of the names. We crossmatched the domains we detected with DGArchive[5] without a result. Interestingly, all except for one set of suspicious names, was only queried by a single host on one or two days. The final cluster, consisting of 26 domain names, we found turned out to be from an advertising network utilizing a few random looking domain names to distribute their ads. Also, for all detected clusters we manually checked the network traces of the detected hosts to make sure the algorithm found all relevant domain names (as requested by the affected hosts). This was always the case, although our algorithm did always include a minority of benign domains as well.

Since we want to detect botnets active in a network, one of the filtering parameters in our algorithm allows to reduce the output by only returning detected clusters with a minimum average requesters per domain name within a cluster. In our tests with the injected traffic, we noticed that even the hypothetical infection of 0.02% of the network with a DGA would make this average stand out significantly significantly from other detected clusters that are actually benign. After applying this filter, to an average of 2 unique requesters per domain name in a cluster, the output would be reduced to a manageable volume for a network admin to manually check (22 clusters in the tested traffic).

A constrained evaluation scenario. Given the algorithmic complexity of the Pleiades approach, a detection for the full network was intractable. To compare detection in networks with a known infection, we constructed a network of 1,000 hosts, of which 200 are infected with a botnet performing periodic DNS requests every 10 minutes to DGA domains (constructed as done by the Locky ransomware and domains generated by concatenating dictionary words). This scenario results in an Eigenvalue decomposition that is still computationally feasible, and is therefore testable with Pleiades. For four days in a row, all hosts had a predetermined start and shutdown time, mimicking hosts being turned on in the morning and being shut down in the afternoon and evening. Additionally, all 1,000 hosts would randomly contact domains of the Alexa top 200. The precision and recall for Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm on this constructed network is shown in table 2. As can be seen, the algorithms most heavily relying on linguistics of domain names, Pleiades and Phoenix, do not perform well with the DGA domains consisting of dictionary words. As expected, request-based algorithms such as BotGAD, PsyBoG and our algorithm do not see any change in this synthetic network.

Fluctuations in Access Times

As shown in the previous section, linguistic-based detection can be easily circumvented by a next-generation DGA if domain names contain less entropy and resemble more normally requested domain names. This issue of better resembling background traffic can however also be said about the request times, as users do not access web resources and generate DNS lookups in a strict pattern as would be generated by an algorithm. In this section, we hence investigate the effect of variations in request times on the performance of detection algorithms. For a botmaster, this step would be equally trivial to make - instead of letting the malware contact a candidate domain every x seconds, requests are issued every $x + k$ seconds, where k would be a random number drawn from a Gaussian distribution.

Figure 6 shows exactly this scenario, the detection accuracy given that the periodicity of requests is shifted by a random delay. As we can see in the figure, a periodicity-based approach such as PsyBoG – which was very resilient to a change in domain name patterns – heavily degrades as soon as variability in the requests

[5] https://dgarchive.caad.fkie.fraunhofer.de/

		Linguistic-based		Periodicity-based	
	our algorithm	Pleiades	Phoenix	BotGAD	PsyBoG
Locky - precision	1.00	1.00	0.86	1.00	0.80
Locky - recall	1.00	0.02	0.51	1.00	1.00
natural language - precision	1.00	0.00	0.73	1.00	0.80
natural language - recall	1.00	0.00	0.23	1.00	1.00

Table 2: Precision and recall for DGA detection in a randomly constructed network for Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm. Domains were generated with a seeded random number generator and results were averaged over ten runs.

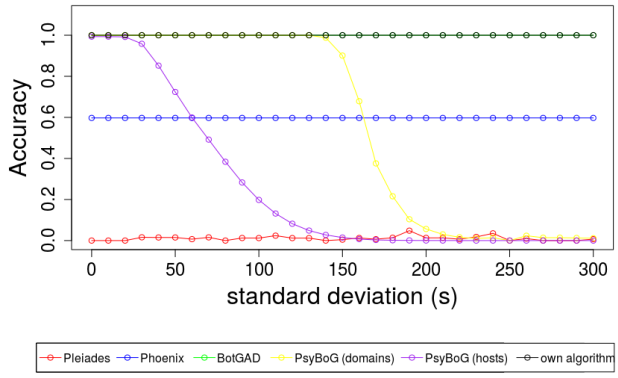

Figure 6: Detection accuracy of the five algorithms as a function of the periodicity of requests by infected hosts.

in introduced. Already at a standard deviation of 2 minutes, the accuracy is comparable to a random guess for the infected hosts in the network.

6 CONCLUSION

In this paper we have evaluated the detection performance of current DGA detection algorithms against the next likely step in the evolution of botnets: the generation of random, but human-readable domain names that no longer stand out from regular DNS traffic, as well as randomness introduced in the request times. Although to date this practice is not established in malware, historical experience shows that malware authors co-evolve their techniques as soon as defense mechanisms are sufficiently widespread and harm their installation base and return on investment. We find that existing approaches are not up to par in detecting this next evolutionary development, and propose an alternative approach – which does not make any assumption on the structure of the domains themselves – which can detect DGA based on human-readable words.

REFERENCES

[1] M. Antonakakis and R. Perdisci, "From throw-away traffic to bots: detecting the rise of DGA-based malware," in *Proceedings of the 21st USENIX Security Symposium*, 2012, p. 16.

[2] B. AsSadhan, J. M. F. Moura, D. Lapsley, C. Jones, and W. T. Strayer, "Detecting botnets using command and control traffic," in *Proceedings - 2009 8th IEEE International Symposium on Network Computing and Applications, NCA 2009*. IEEE, jul 2009, pp. 156–162.

[3] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," pp. 7–7, 2006.

[4] J. Bound and Y. Rekhter, "Dynamic Updates in the Domain Name System (DNS UPDATE)." [Online]. Available: https://tools.ietf.org/html/rfc2136

[5] P. Chen, L. Desmet, C. Huygens, A. D. Christopher Alberts, G. Gu, P. Porras, V. Yegneswaran, M. Fong, W. Lee, K. Jarvis, J. Milletary, A. T.-F. Y. Juels, C. Poulin, V. Sekar, Y. Xie, M. Reiter, H. Zhang, A. K. Sood, R. J. Enbody, S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, T. Bowen, R. M. Amin, J. J. C. H. Ryan, J. R. Van Dorp, C. Smutz, A. Stavrou, N. Virvilis, D. Gritzalis, P. Bhatt, E. T. Yano, and P. Gustavsson, "BotHunter: detecting malware infection through IDS-driven dialog correlation," *IEEE Security and Privacy*, vol. 10, no. 3, p. 12, 2014.

[6] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," *Computer Networks*, vol. 56, no. 1, pp. 20–33, 2012.

[7] J. Demarest, "Taking down botnets: Public and private efforts to disrupt and dismantle cybercriminal networks," Department of Justice, Tech. Rep., 2014.

[8] M. Feily, A. Shahrestani, and S. Ramadass, "A Survey of Botnet and Botnet Detection," in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009, pp. 268–273.

[9] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *SS'08: Proceedings of the 17th conference on Security symposium*, vol. 5, no. 2. USENIX Association, 2008, pp. 139–154.

[10] T. Holz, C. Gorecki, K. Rieck, and F. Freiling, "Detection and Mitigation of Fast-Flux Service Networks," *Ndss*, 2008.

[11] J. Kwon, J. Lee, H. Lee, and A. Perrig, "PsyBoG: A scalable botnet detection method for large-scale DNS traffic," *Computer Networks*, vol. 97, pp. 48–73, 2016.

[12] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer, "Usilng Machine Learning Technliques to Identify Botnet Traffic," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, nov 2006, pp. 967–974.

[13] Mahalanobis, "On the generalised distance in statistics," vol. 2, no. 1, 1936.

[14] D. Pelleg, D. Pelleg, A. Moore, and A. Moore, "X-means: Extending K-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning table of contents*. San Francisco: Morgan Kaufmann, 2000, pp. 727–734.

[15] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-based botnet tracking and intelligence," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8550 LNCS. Springer, 2014, pp. 192–211.

[16] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests," *Information Sciences*, vol. 278, pp. 488–497, sep 2014.

[17] P. Stoica and R. L. Moses, *Introduction to spectral analysis*. Prentice hall Upper Saddle River, 1997, vol. 1.

[18] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in *Proceedings - Conference on Local Computer Networks, LCN*. Springer, 2006, pp. 195–202.

[19] Symantec, "Internet security threat report," Tech. Rep., 2016.

[20] P. Szor, *The Art of Computer Virus Research and Defense*. Addison Wesley, 2005, vol. 43, no. 03.

[21] M. Thomas and A. Mohaisen, "Kindred domains," in *Proceedings of the 23rd International Conference on World Wide Web - WWW '14 Companion*, ser. WWW '14 Companion. New York, NY, USA: ACM, 2014, pp. 707–712.

[22] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, "Automatically Generating Models for Botnet Detection." Springer, Berlin, Heidelberg, 2009, pp. 232–249.

[23] S. W.-t. Yih and C. Meek, "Learning Vector Representations for Similarity Measures," 2010.