Characterizing Automatically Generated Botnet Domains from Passive DNS Monitoring

Abstract

Modern botnets rely on domain-generation algorithms (DGAs) to build resilient command-and-control infrastructures. Recent work focuses on recognizing automatically generated domains (AGDs) from DNS traffic, which potentially allows to identify previously unknown AGDs to hinder or disrupt botnets' communication capabilities.

Unfortunately, such approaches require to deploy low-level DNS sensors to access data whose collection poses practical and privacy issues, making their adoption problematic. Instead, we propose a novel way to exploit publicly available and privacy-preserving databases of historical recursive-level DNS traffic. Analyzing such data through linguistic-based models of suspicious domains, we are able to identify automatically generated domain names, characterize their DGAs, isolate logical groups of domains that represent the respective botnets, enrich those groups with new previously unknown automatically generated domain names, and produce novel knowledge about the evolving behavior of each tracked botnet.

We evaluated our approach on millions of real-world domains. Our approach correctly isolates families of automatically generated domains that belong to distinct DGAs, and distinguishes automatically generated from non-automatically generated domains in 94.8 percent of the cases. We also verified our findings against live botnets.

1. INTRODUCTION

Botnets continue to play a significant role in today's cybercrime ecosystem, and have become a commodity platform for online lucrative activities. The latest ENISA Threat Landscape [12] highlighted that one of the adverse effects of this malware-as-a-service trend is an increased number of small, distinct botnets, which are predicted to replace traditional large botnets: Smaller botnets are likely to fly under the radar because of their size. This creates the problem of keeping track of such a diverse population of distinct, yet related threats. Furthermore, botnets are used for a variety of malicious purposes, ranging from spamming to information stealing and espionage.

Identifying malicious activities related to botnets is a well-studied problem with many proposed solutions. One of the most promising directions consists in observing and obstructing the traffic of their communication infrastructure. Typically, the security defenders strive to find the IPs or the domain names of the command-and-control (C&C) server of a botnet with the goal of creating sinkholing IPs: Eventually, the bots will starve not being able to contact their masters.

The most reliable yet easy-to-manage bot-to-master centralized communication mechanism relies on time-dependent rendezvous points via domain-generation algorithms (DGAs). In these mechanisms the bots and the master agree on pseudo-random algorithms that generate a large number of domains where the bot tries to contact its master. The key is that, at any moment, only a small number of domains is active thus revealing the true IPs to the master. This characteristic makes dealing with DGAs very expensive for the security defenders, because they would need to track several thousands of domains before finding the one used as the rendezvous point.

Researchers have proposed various approaches for recognizing automatically generated domains (AGDs) with the goal of correlating this information with other datasets (e.g., blacklists or domain reputation scores). We notice two main shortcomings. Although they provide rich details on *individual* malicious domains (e.g., Antonakakis et al. [2], Bilge et al. [6]), they fail in *correlating* distinct but related abuses of such domains. Unfortunately, the approaches that are able to automatically correlate malicious domains to the botnets that lie behind them (e.g., Antonakakis et al. [3]) require access to Internet traffic whose collection poses issues. More precisely, the state-of-the-art approach [3] requires visibility on the infected machines' IP address. Besides privacy issues, this creates the research problem of repeatability and, more importantly, requires a low-level traffic sensor deployed between the infected machines and the DNS servers that they contact, with visibility of the original DNS queries.

We propose Phoenix, which leverages publicly available passive DNS traffic to (1) find AGDs, (2) characterize the generation algorithms, (3) isolate logical groups of domains that represent the respective botnets, and (4) produce novel knowledge about the evolving behavior of each tracked botnet. Phoenix requires no knowledge of the DGAs active in the wild and, in particular, no reverse engineering of the malware. Being based on recursive-level passive DNS traffic, our approach guarantees scientific repeatability of malware experiments [17] and preserves the privacy of the infected computers.

Phoenix creates linguistic models of *non*-AGDs (e.g., benign domains). Domains that violate such models are considered to be automatically generated. Then, Phoenix groups these domains according to the domain-to-IP relations. From these groups, Phoenix derives a generic set of fingerprints to label new domains as belonging to some botnet. Such fingerprints are useful to characterize the evolution of botnets and gather insights on their activity (e.g., migrations).

We evaluated Phoenix on millions of real-world domains. Phoenix correctly isolated families of domains that belong to different DGAs. Also, on February 9th we obtained an undisclosed list of AGDs for which no knowledge of the respective botnet was available. Phoenix labeled these domains as belonging to Conficker: Further investigation eventually confirmed that it was indeed Conficker.B.

In summary, we make the following contributions:

- our system identifies groups of AGDs and model the characteristics of the generation algorithms, with less requirements than previous work; with this we ensure a) privacy-preserving data collection, b) ease of large-scale deployment, and c) repeatable evaluation;
- our system automatically associates *new* malicious domains to the activity of known botnets;
- we show that the above findings can be used to build *new correlated knowledge* that allow security analysts to track the evolution of a botnet.

2. AGD NAMES

Thanks to the Domain Name System (DNS), applications and users do not need to keep track of IP addresses, but can use human-readable aliases, called humanly generated domain (HGD) names from hereinafter. The hierarchical structure of the DNS protocol efficiently maintains cached copies of the name-IP associations, so that the resolution is fast, reliable, and highly distributed. The domain name resolution has many uses, from load balancing to failover setups. Unfortunately, these characteristics are useful for malicious developers, too. Miscreants began using DNS to make *centralized*[1] botnet infrastructures more reliable and, more importantly, harder to map and take down [2, 3, 9, 15, 16, 22]. In particular, botnets rely on the DNS (also) to implement call-home functionalities, such that the bots can contact their C&C server to receive instructions, updates, or to upload stolen data. For instance, the botnet operator, commonly referred to as the *botmaster*, can change the IP address of the C&C dynamically, by updating the DNS record, for instance when he or she needs to migrate the C&C to other (more lenient) autonomous systems (ASs). Similarly, the botmaster can associate short-lived multiple IPs of infected hosts to an individual DNS domain over time [9, 15] to improve the botnet's redundancy, while hiding the IP-based coordinates of a malicious host (i.e., the mothership). In this context, a very effective technique to improve centralized botnets' resiliency to take downs and tracking is domain flux, which resorts to equip a bot with DGAs.

Domain-generation Algorithms. DGAs have become the technique of choice[2] for building effective rendezvous mechanisms between the bots and C&C servers: The bots and the C&C servers implement the same algorithm to generate a large and time-dependent list of AGDs, based on pseudo-unpredictable seeds (e.g., trending topics on Twitter, results of a Google search). Only one of these AGDs is actually registered and pointing to the true IP address of the C&C. The bots will then generate and query all these domains, according to the DGA, until a DNS server answers with a non-NXDOMAIN reply, that is the IP address of the respective (existing) AGD. The key is that only the DGA authors (or the botmasters) know exactly when the upcoming rendezvous domain has to be registered and activated[3].

Finding "families", or groups, of related AGDs is therefore fundamental, because they provide valuable insights to recognize, for instance, groups of botnets that implement a DGA with common characteristics—which refer, possibly, to the same botnet, or an evolution. Therefore, the analysts can follow the evolution of these botnets and their (changing) C&Cs over time, where these are hosted, and the number of machines involved. The task of finding families of related AGDs, however, is tedious and labor-intensive, although previous research has devised mechanisms to partially automate it.

In this work, we go beyond the state of the art and observe that instances of AGD names drawn from the same DGA can be generalized into "fingerprint" of the generation algorithm itself, without reversing its implementation. This boosts the detection and the classification of newly seen AGD names.

Recognizing DGAs. A side effect of the DGA mechanisms described above is that the bots will perform a disproportionately large amount of DNS queries, which will result in NXDOMAIN replies. This happens because the vast majority of such queries will hit a non-existing domain. On the other hand, legitimate hosts have no reasons to generate high volumes of queries that yield NXDOMAIN replies. This observation has been leveraged by Antonakakis et al. [3], who proposed to detect DGA-based bots by correlating the clients' IP found in the requests and the corresponding NXDOMAIN replies at one or more DNS resolvers. This allows to identify groups of bots while they are attempting to find the rendezvous domain to reach their C&C server. Also, the authors show that their system finds families of AGDs with similar characteristics (e.g., same upper-level domain name, lexicon or syntax), possibly coined by the same DGA.

However, the NXDOMAIN responses alone carry little information that can be directly used to identify the families of AGDs. Specifically, an NXDOMAIN response only holds the queried domain name plus some timing data. Therefore, as also noticed by Perdisci et al. [16], the NXDOMAIN criterion requires knowledge of the querying hosts (i.e., the bots). For this reason, Antonakakis et al. [3] had to rely on the client IP to group together NXDOMAINS queried by the same set of hosts. Yadav and Reddy [20] instead group together DNS queries originated by the same client to define the correlation between distinct requests that target the same domains.

Drawbacks of Current Approaches. Setting aside the problem of accuracy, the crucial issue of the current approaches is the type of information they need as an input. Indeed, relying on the IP addresses of the querying hosts has some drawbacks. First, it is error prone, because no assumptions can be made on and IP-(re)assignment and masquerading policies employed by ASs. Secondly, and more importantly, obtaining access to this information is difficult because it is available from DNS servers placed *below* the recursive DNS level (e.g., host DNSs). This can be problematic for researchers, but also for practitioners who want to operate these systems. Last, the need for information about the querying hosts raises privacy issues and, consequently, leads to non-repeatable experiments [17], as datasets that include these details are not publicly available. Moreover, the requirement constrains the deployment of detection tools to the lowest level of DNS hierarchy, preventing a large-scale, high-level use of the proposed solutions.

3. GOAL AND CHALLENGES

After considering the above motivations, we conclude that a method to find families of AGDs yielded by the same DGAs, without requiring access to low- or top-level DNS data is necessary.

AGDs appear completely random at sight. Nevertheless, creating automated procedures capable of modeling and characterizing such "randomness" is hard, in particular when observing one domain at a time, as one sample is not representative of the whole random generation process. On the other hand, grouping AGD samples to extract the characteristics of the DGA is also challenging. How to group AGDs samples together? How to avoid spurious samples that would bias the result?

Collecting and dealing with DNS traffic presents some challenges on (1) where to place the observation point and (2) how to process the

[1]Although DNS may be used by botnets of arbitrary network topology, in this paper we focus on centralized domain flux botnet infrastructures.

[2]`https://blog.damballa.com/archives/1906`

[3]Differently from current DGAs, former forms of DGA (e.g., those based on sequential queries of AGDs) and failure to promptly register domains have allowed botnet takeovers in the past [19].

collected data. The amount of information that can be collected varies depending on where the sensors are deployed in the DNS hierarchy (e.g., low-level sensors have access to the querying hosts, but are difficult to deploy, whereas higher-level sensors are easier to deploy but their visibility is limited to the stream of queried domains and their IPs). Moreover, the domain-to-IP relations are highly volatile by their nature, with both dense and sparse connections; this impacts the space and computation complexity of the algorithms that are needed to analyze DNS traffic.

Last, little or no ground truth is usually available regarding DGAs. When available, such ground truth is limited to the knowledge of a specific implementation of DGA as a result of a specific reverse engineering effort [19]. Therefore, this knowledge is outdated once released and, more importantly, not representative of the whole domain generation process but only of a limited view.

4. SYSTEM OVERVIEW

Phoenix is divided into three modules (see Fig. 1). The core of Phoenix is the **DGA Discovery** module, which identifies and models AGDs by mining a stream of domains. The **AGD Detection** module receives one or more domain names with the corresponding DNS traffic, and uses the models built by the **DGA Discovery** module to tell whether such domain names appear to be automatically generated. If that is the case, this module labels those domains with an indication of the DGA that is most likely behind the domain generation process. The **Intelligence and Insights** module aggregates, correlates and monitors the results of the other modules to extract meaningful insights and intelligence information from the observed data (e.g., whether an unknown DGA-based botnet is migrating across ASs).

4.1 DGA Discovery Module

This module receives as input (1) a stream of domain names that are known to be malicious and (2) a stream of DNS traffic (i.e., queries and replies) collected above the recursive resolvers and related to such domains. This information is publicly accessible and can be obtained easily from (1) a blacklist or domain reputation system (e.g., Exposure) and (2) a passive and privacy-preserving DNS monitor (e.g., ISC/SIE). Note that the blacklists that we rely on are generated from privacy-preserving DNS traffic too. We make no assumptions on the type of malicious activity which these domains are used for (e.g., phishing websites, spamming campaigns or drive-by download websites, botnet communication protocols).

This module follows a 3-step pipeline to recognize domains that are used in DGA-based botnets.

Step 1 (AGD Filtering): We extract a set of linguistic features from the domain names. The goal is to recognize the (domain) names that appear to be the results of automatic generations. For instance, we distinguish between `5ybdiv.cn`, which appears to be an AGD, and `searchsmart.tk`, which appears to be a HGD. Differently from previous work ([3, 21]), our features work on *individual instances* of ADG, not on groups of AGDs. We make no assumptions about the type of DGA that have generated the domains, although we do assume that at least one exists. The output is a set of AGDs, possibly generated by different DGAs. We accomplish this using a semi-supervised techniques, which only requires limited knowledge on HGDs (not on existing ADGs).

Step 2 (AGD Clustering): We extract some features from the DNS traffic of the domains that have passed **Step 1**. We use these features to cluster together the AGDs that have resolved to similar sets of IP addresses—possibly, the C&C servers. These *AGD clusters* will partition the AGDs according to the DNS replies observed. For example, if the AGDs `5ybdiv.cn` and `hy093.cn` resolved to the same pool of IPs, we will cluster them together. Here, we assume that AGDs generated by different DGAs are used by distinct botnets, or distinct malware variants, or at least by different botmasters, who have customized or tuned a DGA for their C&C strategy. Therefore, this partitioning will, to some extent, reflect the different groups of botnets that they have been employed for.

Step 3 (DGA Fingerprinting): We extract *other* features from the AGD clusters to create models that define the fingerprints of the respective DGAs. The **AGD Detection** module uses these fingerprints as a lookup index to identify the type of previously unseen domains. For instance, as clarified in the remainder of this paper, `epu.org` and `xmsyt.cn` will match two distinct fingerprints.

4.2 AGD Detection Module

This module receives in input a (previously unseen) domain name d, which can be either malicious or benign, and uses once again the **AGD Filtering** step to verify whether it is automatically generated.

The domain names that will pass this filter will undergo further checks, which may eventually flag them as not belonging to any AGD cluster (i.e., not matching any of the fingerprints). Therefore, in this step, flagging as AGD a (benign) domain that do not belong to some DGA is not strictly important. It is instead more important not to discard suspicious domains. Therefore, for this module only, we configure the **AGD Filtering** step with looser parameters (as described in §5.1), such that we do not discard domains that exhibit, even slightly, the linguistic characteristics that are typical of AGDs.

Then, this module leverages the AGD clusters and their respective fingerprints to find the DGA, if any, that may lie behind d.

4.3 Intelligence and Insights Module

Once an AGD is labeled as such, it can be recognized among others as belonging to a given "type" of DGA. Therefore, the outcome of the previous modules allows the extraction of summarized information and novel knowledge.

With this knowledge, the addresses of the C&C servers and lists of malicious AGDs can be grouped together in small sets that are easier to analyze than if considered into one, large set, or distinctly. For instance, if an analyst knows that 100 domains are malicious, he or she can use the label information to split them into two smaller sets: one that contains domain names "similar" to `5ybdiv.cn` and `hy093.cn`, and one with domains "similar" to `epu.org`. The top level domain is not distinctive, we use it here as a mere example. With this information, the analyst can track separately the evolution of the IPs that the two groups point to and take actions. For example, recognizing when a C&C is migrated to a new AS or undergoing a takedown operation is easier when the set of IPs and domains is small and the characteristics of the DGA are known and uniform.

Generally speaking, these analyses, for which we provided two use cases in §6.4, can lead to high-level intelligence observations and conjectures, useful for the mitigation of DGA-related threats. In this, we advance the state of the art by providing a tool that goes beyond blacklists and reputation systems.

5. SYSTEM DETAILS

We implemented Phoenix in Python using the NumPy package, for statistical functions, and the SciPy [10] package, for handling sparse matrices. Instead of relying entirely on existing libraries, we implemented and customized the machine-learning algorithms described in the remainder of this section for efficiency reasons. For this, we leveraged domain-specific knowledge (e.g., data dimensionality).

For the purpose of this work, a *domain name* is a sequence of labels separated by dots (e.g., `www.example.com`). Specifically, the latter parts form the *public suffix* (e.g., `.com`, `.co.uk`), under which Internet users can register a name, which is called *chosen prefix* (e.g., `example`). The public suffix, which is often referred to as top-

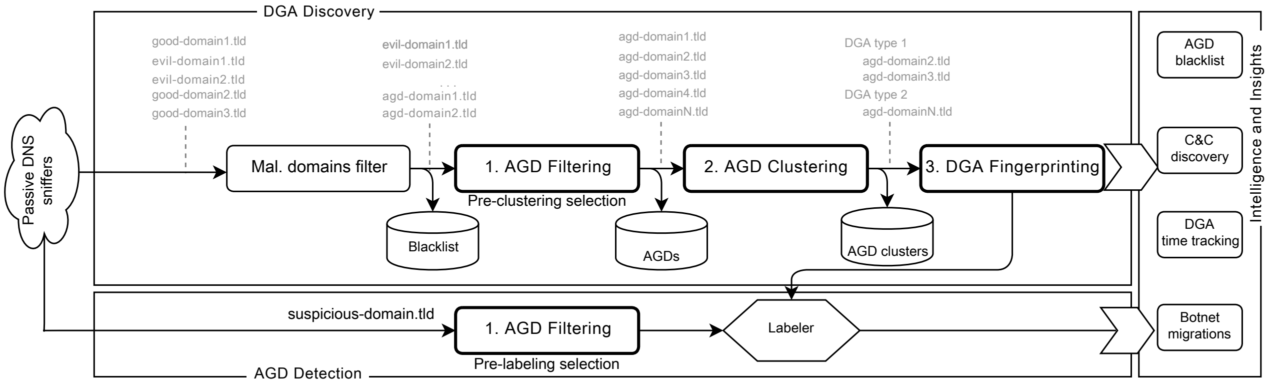

Figure 1: System overview of Phoenix. The DGA Discovery module processes the domain names from a domain reputation system and identifies interesting AGDs. The AGD Detection module analyzes a stream of DNS traffic and recognizes the names that appear to be automatically generated. This module also labels newly discovered AGDs as results of a particular DGA. The last module, Intelligence and Insights, provides the analyst with information useful, for instance, to track a botnet over time, as described in §6.4.

level domain (TLD), can contain more than one label (e.g., `.co.uk`). The term effective TLD (eTDL) is thus more correct. eTLDs are enumerated at `http://publicsuffix.org/` to allow parsing. A domain name can be organized hierarchically into more subdomains (e.g., `www.example.com`, `ssh.example.com`). We only consider the first level of a chosen prefix, simply because a DGA that works on further levels makes little sense, as the first level would still be the single point of failure.

5.1 Step 1: AGD Filtering

In this step we make the assumption that if a domain is automatically generated it has different linguistic features than a domain that is generated by a human. This assumption is reasonable because HGDs have the primary purpose of being easily remembered and used by human beings, thus are usually built in a way which meets this goal. On the other hand, AGDs exhibit a certain degree of linguistic randomness, as numerous samples of the same randomized algorithm exist. Corner cases of this assumption are discussed in §7.

5.1.1 *Linguistic Features*

For ease of explanation and implementation, Phoenix considers the linguistic features based on the English language, as discussed in §7. Given a domain d and its prefix $p = p_d$, we extract two classes of linguistic features to build a 4-element feature vector for each d. These features are devised to work well on single domains d. Previous work, instead, relied on features extracted from *groups* of domains, which creates the additional issue on how to select such groups. Previous work circumvented this problem by choosing *random* groups of domains. However, there is no rigorous way to verify the validity of such assumption. Therefore, we made an effort to design features that require no groupings of domains.

LF1: Meaningful Characters Ratio. This feature models the ratio of characters of the chosen prefix p that comprise a meaningful word. Domains with a low ratio are likely automatically generated.

Specifically, we split p into n meaningful subwords w_i of at least 3 symbols: $|w_i| \geq 3$, leaving out as few symbols as possible:

$$R(d) = R(p) := \max \frac{\sum_{i=1}^{n} |w_i|}{|p|}$$

In the case of $p =$ `facebook`, $R(p) = (|$`face`$| + |$`book`$|)/8 = 1$, the prefix is fully composed of meaningful words, whereas $p =$ `pub03str`, $R(p) = (|$`pub`$|)/8 = 0.375$.

fa	ac	ce	eb	bo	oo	ok	
109	343	438	29	118	114	45	$S_2 = 170.8$

aa	aw	wr	rq	qv	
4	45	17	0	0	$S_2 = 13.2$

Figure 2: 2-gram normality score S_2 for `facebook.com` and `aawrqv.biz`.

LF2: n-gram Normality Score. This class of features captures the pronounceability of the chosen prefix of a domain. This problem is well studied in the field of linguistics, and can be reduced to quantifying the extent to which a string adheres to the phonotactics of the (English) language. The more permissible the combinations of phonemes [5, 18], the more pronounceable a word is. Domains with a scarce number of such combinations are likely automatically generated.

We calculate this class of features by extracting the n-grams of p, which are the substrings of p of length $n \in \{1,2,3\}$, and counting their occurrences in the (English) language dictionary. The features are thus parametric to n:

$$S_n(d) = S_n(p) := \frac{\sum_{n\text{-gram } t \text{ in } p} \text{count}(t)}{|p| - n + 1}$$

where count(t) are the occurrences of the n-gram t in the dictionary.

Fig. 2 shows the value of S_2, along with its derivation, for one HGD and one AGD.

5.1.2 *Statistical Linguistic Filter*

Phoenix uses **LF1-2** to build a feature vector $\boldsymbol{f}(d) = [R(d), S_{1,2,3}(d)]^T$. It extracts these features from a dataset of HGDs (Alexa top 100,000) and calculates their mean $\boldsymbol{\mu} = \left[\overline{R}, \overline{S_1}, \overline{S_2}, \overline{S_3}\right]^T$ and covariance (matrix) $\boldsymbol{C}$, which respectively represent the statistical average values of the features and their correlation. Strictly speaking, the mean defines the centroid of the HGD dataset in the features' space, whereas the covariance identifies the shape of the hyperellipsoid around the centroid containing all the samples. Our filter constructs a confidence interval, with the shape of such hyperellipsoid, that allows us to separate HGDs from AGDs with a measurable, statistical error that we set a priori.

The rationale is that obtaining a dataset of HGDs is straightforward and does not constrain the filtering to specific AGDs: Our filter thus models non-AGDs by means of the generic modeling of HGDs.

Distance Measurement. To tell whether a previously unseen domain d' resembles the typical features of HGDs, the filter mea-

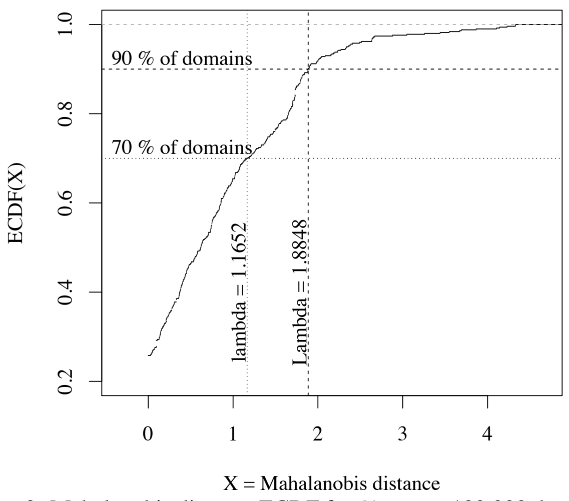

Figure 3: Mahalanobis distance ECDF for Alexa top 100,000 domains with λ and Λ identification.

sures the distance between the feature vector $\boldsymbol{f}(d') = \boldsymbol{x}$ and the centroid. To this end, we leverage the Mahalanobis distance: $\mathrm{d}_{Mah}(\boldsymbol{x}) = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$. This distance has the property of (1) taking into account the correlation between features—which is significant, because of how the features are defined, and (2) operating with scale-invariant datasets.

Distance Threshold. A previously unseen domain d' is considered as automatically generated when its feature vector identifies a point that is too distant from the centroid: $\mathrm{d}_{Mah}(\boldsymbol{x}) > t$. To take a proper decision we define the threshold t as the p-percentile of the distribution of $\mathrm{d}_{Mah}(\boldsymbol{x})$, where $(1-p)$ is the fraction of HGDs that we allow to confuse as AGDs. In this way, we can set a priori the amount of errors.

As mentioned in §4.2, the **DGA Discovery** module employs a strict threshold, $t = \Lambda$, whereas the **AGD Detection** module requires a looser threshold, $t = \lambda$, where $\lambda < \Lambda$.

Threshold Estimation. To estimate proper values for λ and Λ, we compute $\mathrm{d}_{Mah}(\boldsymbol{x})$ for $\boldsymbol{x} = \boldsymbol{f}(d), \forall d \in \mathbb{D}_{HGD}$, whose distribution is plotted in Fig. 3 as ECDF. We then set Λ to the 90-percentile and λ to the 70-percentile of that distribution, as annotated in the figure. Fig. 4 depicts the 99%-variance preserving 2D projection of the hyperellipsoid associated to $\mathbb{D}_{HGD}$, together with the confidence interval thresholds calculated as mentioned above.

5.2 Step 2: AGD Clustering

This step receives as input the set of domains $d \in \mathbb{D}$ that have passed **Step 1**. These domains are such that $\mathrm{d}_{Mah}(\boldsymbol{f}(d)) > \Lambda$, which means that d is likely to be automatically generated, because it is too far from the centroid of the HGDs.

Our goal is to cluster domains according to their similarity. We define as *similar* two domains that resolved to similar sets of IP addresses. The rationale is that the botmaster of a DGA-based botnet registers several domains that, at different points in time, resolve to the same set of IPs (i.e., the C&C servers). To find similar domains, we represent the domain-to-IP relation as a bipartite graph, which we convert in a proper data structure that allows us to apply a spectral clustering algorithm [14] that returns the groups of similar domains (i.e., nodes of the graph).

In this graph, two sets of node exists: $K = |\mathbb{D}|$ nodes represent the domains, and $L = |\mathrm{IPs}(\mathbb{D})|$ nodes represent the IPs. An edge exists

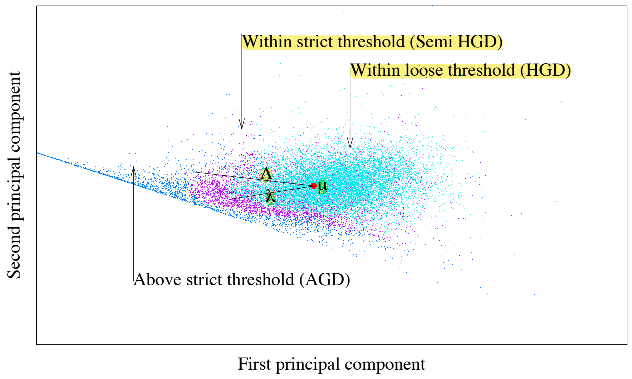

Figure 4: Principal components of the Alexa top 100,000 domains hyperellipsoid with annotation of the confidence interval thresholds.

from a node $d \in \mathbb{D}$ to node $l \in \mathrm{IPs}(\mathbb{D})$ whenever a domain pointed to an IP.

5.2.1 Bipartite Graph Recursive Clustering

To cluster the domain nodes $\mathbb{D}$, we leverage the DBSCAN clustering algorithm [8], which is particularly fast and easy to implement in our scenario.

Data Structure. We encode the bipartite graph as a sparse matrix $\boldsymbol{M} \in \mathbb{R}^{L \times K}$ with L rows and K columns. Each cell $M_{l,k}$ holds the weight of an edge $k \to l$ in the bipartite graph, which represents the fact that domain d_k resolves to IP l. The weight encodes the "importance" of this relation. For each IP l in the graph, the weights $M_{l,k}, \forall k = 1, \ldots, K$ are set to $\frac{1}{|\mathbb{D}(l)|}$, where $\mathbb{D}(l) \subset \mathbb{D}$ is the subset of domains that point to that IP. This weight encodes the peculiarity of each IP: The less domains an IP is pointed by, the more characterizing it is.

Domain Similarity. At this point we calculate the matrix $\boldsymbol{S} \in \mathbb{R}^{K \times K}$, whose cells encode the similarity between each pair of domains d and d'. We want to consider two domains as highly similar when they have peculiar IPs in common. Therefore, we calculate the similarity matrix from the weights, as $\boldsymbol{S} = \boldsymbol{N}^T \cdot \boldsymbol{N} \in \mathbb{R}^{K \times K}$, where $\boldsymbol{N}$ is basically $\boldsymbol{M}$ normalized by columns (i.e., $\sum_{l=1}^{L} M_{l,k} = 1, \forall k = 1, K$). This similarity matrix implements the rationale that we mentioned at the beginning of this section.

Domain Features and Clustering. We apply the DBSCAN algorithm hierarchically. We compute the first normalized eigenvector $\boldsymbol{v}$ from $\boldsymbol{S}$. At this point, each domain name d_k can be represented by its feature v_k, the k-th element of $\boldsymbol{v}$, which is fed to the DBSCAN algorithm to produce the set of R clusters $\mathcal{D} = \{\mathbb{D}^1, \ldots, \mathbb{D}^R\}$ at the current recursive step.

Clustering Stop Criterion. We recursively repeat the clustering process on the newly created clusters until one of the following conditions is verified:

- a cluster of domains $\mathbb{D}' \in \mathcal{D}$ is too small (e.g., it contains less than 25 domains) thus it is excluded from the final result;
- a cluster of domains has its $\boldsymbol{M}$ matrix with all the elements greater than zero, meaning that the bipartite graph it represents is strongly connected;
- a cluster of domains cannot be split further by the DBSCAN algorithm with the value of ε set. In our experiments, we set ε to a conservative low value of 0.1, so to avoid the generation of clusters

that contain domains that are not similar. Manually setting this value is possible because ε, and the whole DBSCAN algorithm, works on normalized features.

The final output of DBSCAN is $\mathcal{D}^\star = \{\mathbb{D}^1, \dots, \mathbb{D}^R\}$. The domains within each $\mathbb{D}^r$ are similar among each other.

5.2.2 *Dimensionality Reduction*

The clustering algorithm employed has a space complexity of $O(|\mathbb{D}|^2)$. To keep the problem feasible we randomly sample our dataset $\mathbb{D}$ of AGDs into I smaller datasets $\mathbb{D}_i, i = 1, \dots, I$ of approximately the same size, and cluster each of them independently, where I is the minimum value such that a space complexity in the order of $|\mathbb{D}_i|^2$ is affordable. Once each $\mathbb{D}_i$ is clustered, we recombine the I clustered sets, $\mathcal{D}_i^\star = \{\mathbb{D}^1, \dots, \mathbb{D}^{R_i}\}$, onto the original dataset $\mathbb{D}$. Note that each $\mathbb{D}_i$ may yield a different number R_i of clusters. This procedure is very similar to the MapReduce programming model, where a large computation is parallelized into many computations on smaller partitions of the original dataset, and the final output is constructed when the intermediate results become available.

We perform the recombination in the following post-processing phase, which is run anyway, even if we do not need any dimensionality reduction—that is, when $I = 1$ and thus $\mathbb{D}_1 \equiv \mathbb{D}$.

5.2.3 *Clustering Post Processing*

We post process the set of clusters of domains $\mathcal{D}_i^\star, \forall i$ with the following **Pruning** and **Merging** procedures. For simplicity, we set the shorthand notation $\mathbb{A} \in \mathcal{D}_i^\star$ and $\mathbb{B} \in \mathcal{D}_j^\star$ to indicate any two sets of domains (i.e., clusters) that result from the previous DBSCAN clustering, possibly with $i = j$.

Pruning. Clusters of domains that exhibit a nearly one-to-one relation with the respective IPs are considered unimportant because, by definition, they do not reflect the concept of DGA-based C&Cs (i.e., many domains, few IPs). Thus, we filter out the clusters that are flat and show a pattern-free connectivity in their bipartite domain-IP representation. This allows to remove "noise" from the dataset.

Formally, a cluster $\mathbb{A}$ is removed if $\frac{|\mathrm{IPs}(\mathbb{A})|}{|\mathbb{A}|} > \gamma$, where γ is a threshold that is derived automatically as discussed in §6.

Merging. Given two independent clusters $\mathbb{A}$ and $\mathbb{B}$, they are merged together if the intersection between their respective sets of IPs is not empty. Formally, $\mathbb{A}$ and $\mathbb{B}$ are merged if $\mathrm{IPs}(\mathbb{A}) \cap \mathrm{IPs}(\mathbb{B}) \neq \emptyset$. This merging is repeated out iteratively, until every combination of two clusters violates the above condition.

The outcome of the post-processing phase is thus a set of clusters of domains $\mathcal{E} = \{\mathbb{E}^1, \dots, \mathbb{E}^Q\}$ where each $\mathbb{E}^q$ (1) exhibits a domain-to-IP pattern and (2) is disjunct to any other $\mathbb{E}^p$ with respect to its IPs. In conclusion, each cluster $\mathbb{E}$ contains the AGDs employed by the same botnet backed by the C&C servers at IP addresses $\mathrm{IPs}(\mathbb{E})$.

5.3 Step 3: DGA Fingerprinting

The AGD clusters identified with the previous processing are used to extract fingerprints of the DGAs that generated them. In other words, the goal of this step is to extract the invariants of a DGA. We use these fingerprints in the **AGD Detection** module to assign labels to previously unseen domains, if they belong to one of the clusters.

Given a generic AGD cluster $\mathbb{E}$, corresponding to a given DGA, we extract the following cluster features:

CF1: C&C Servers Addresses. defined as $\mathrm{IPs}(\mathbb{E})$.

CF2: Chosen Prefix Length Range. captures the lengths of the chosen prefix allowed for the domains in $\mathbb{E}$. The boundaries are defined as the lengths of the shortest and longest chosen prefixes of the domains of $\mathbb{E}$.

CF3: Chosen Prefix Character Set. C employed for the chosen prefixes of the domains, defined as $C := \bigcup_{e \in \mathbb{E}} \mathrm{charset}(p_e)$, where p_e is the chosen prefix of e. It captures which characters are used during the random generation of the domain names.

CF4: Chosen Prefix Numerical Characters Ratio Range. $[r_m, r_M]$ captures the ratio of numerical characters allowed in the chosen prefix of a given domain. The boundaries are, respectively, the minimum and the maximum of $\frac{\mathrm{num}(p_e)}{|p_e|}$ within $\mathbb{E}$, where $\mathrm{num}(p_e)$ is the number of numerical characters in the chosen prefix of e.

CF5: Public Suffix Set. The set of eTDL employed by the domains in $\mathbb{E}$.

To some extent, these features define the aposteriori linguistic characteristics of the domains found within each cluster $\mathbb{E}$. In other words, they define a model of $\mathbb{E}$.

5.4 AGD Detection Module

This module receives a previously unseen domain d and decides whether it is automatically generated by running the **AGD Filtering** step with a loose threshold λ. If d is automatically generated, it is matched against the fingerprints of the known DGAs on the quest for correspondences.

In particular, we first select the candidate AGD clusters $\{\mathbb{E}\}$ that have at least one IP address in common with the IP addresses that d pointed to: $\mathrm{IPs}(d) \cap \mathrm{IPs}(\mathbb{E}) \neq \emptyset, \forall \mathbb{E}$. Then, we select a subset of candidate clusters such that have the same features **CF1–5** of d. Specifically, the length of the chosen prefix of d, its character set, its numerical characters ratio, and the eTLD of d must lie within the ranges defined above.

The clusters that survive this selection are chosen as the labels of d.

6. EXPERIMENTAL EVALUATION

Validating the results of the Phoenix is by no means trivial, precisely because, as we claim, Phoenix produces novel knowledge. The reason is that we do not have a ground truth available that would allow us to validate the correctness of Phoenix quantitatively. For example, no information is available, to our knowledge, about the membership of a malicious domain to one family of AGDs. If such ground truth were available, then there would be no need for Phoenix. Therefore, in lack of an established ground truth, we proceed as follows. We validate *quantitatively* the internal components of each module (e.g., to verify that they do not produce meaningless results and to assess the sensitivity of the parameters), and *qualitatively* the whole approach, to make sure that it produces useful knowledge with respect to publicly available information.

6.1 Evaluation Dataset and Setup

The **DGA Discovery** module of Phoenix requires a feed of recursive DNS traffic and a reputation system that tells whether a domain is generally considered as malicious. For the former data source, we obtained access to the ISC/SIE framework[4], which provides DNS traffic data shared by hundreds of different network operators. Differently from previous work [3], this type of traffic is privacy preserving and very easy to collect. We used Exposure [6] as a blacklist, which included 107,179 distinct domains as of October 1st, 2012.

To *validate* the components of Phoenix we relied on ground truth generated by publicly available implementations of the DGAs used by

[4]`https://sie.isc.org/`

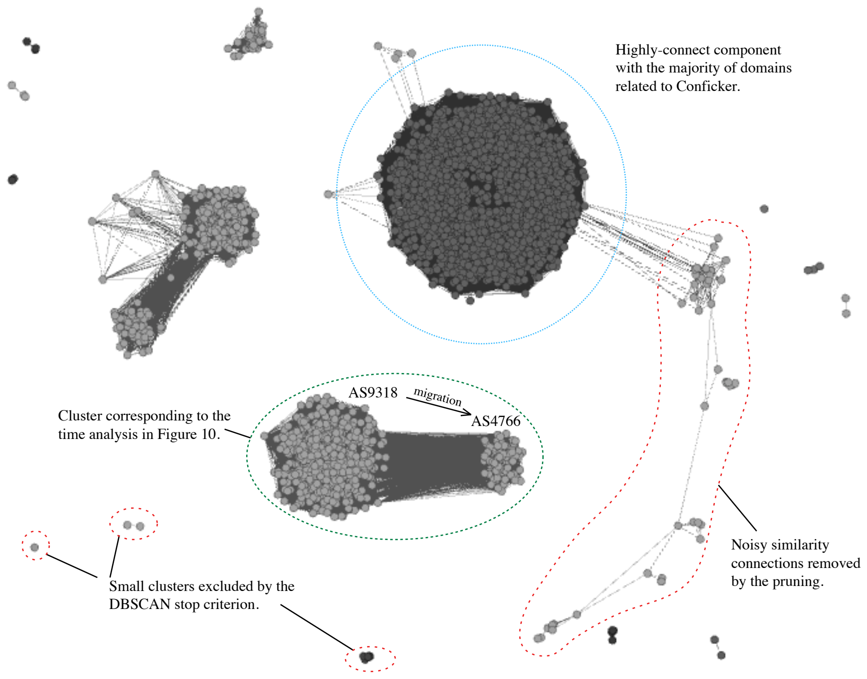
Highly-connect component with the majority of domains related to Conficker.
Cluster corresponding to the time analysis in Figure 10.
AS9318
migration
AS4766
Noisy similarity connections removed by the pruning.
Small clusters excluded by the DBSCAN stop criterion.

Figure 5: Graph representation of the similarity matrix $\boldsymbol{S}$ during the first run of the DBSCAN clustering algorithm, as defined in §5.2.1. The nodes represent the domains, while edges represent the similarity relations. The purpose of the DBSCAN is to isolate strongly connected "communities" of domains from "noisy" and uncorrelated domains.

Conficker [11] and Torpig [19], which have been among the earliest and most widespread botnets that relied on DGAs for C&C communication. After Conficker and Torpig, the use of DGAs kept rising. With these DGAs we generated five datasets of domains, which resemble (and in some cases are equivalent to) the domains generated by the actual botnets: 7500, 7750 and 1,101,500 distinct AGDs for the **Conficker.A**, **Conficker.B** and **Conficker.C** malware, respectively, and 420 distinct AGDs for the **Torpig** dataset. Moreover, we collected the list of 36,346 AGDs that Microsoft claimed in early 2013 to be related to the activity of **Bamital**[5]. Differently from [21], we used AGDs merely as a ground truth for validation, not for bootstrapping our system before run time.

We ran our experiments on a 4-core machine equipped with 24GB of physical memory. All the runs required execution times in the order of the minutes.

6.2 DGA Discovery Validation

6.2.1 Step 1: AGD Filtering

The AGD filter is used in two contexts: by the **DGA Discovery** module as a pre-clustering selection to recognize the domains that appear automatically generated within a feed of malicious domains, and by the **AGD Detection** module as a pre-labeling selection. For pre-clustering, the strict threshold Λ is enforced to make sure that no non-AGD domains pass the filter and possibly bias the clustering, whereas for pre-labeling the loose threshold λ is used to allow more domains to be labeled. Recall that, the **Labeler** will eventually filter out the domains that resemble no known AGD. We test this component in both the contexts against the AGD datasets of **Conficker**, **Torpig** and **Bamital** (that we had never seen before).

The filter, which is the same in both the contexts, is best visualized by means of the ECDF of the Mahalanobis distance. Fig. 6 shows the ECDF from the AGD datasets, compared to the ECDF from the

[5]http://noticeofpleadings.com/

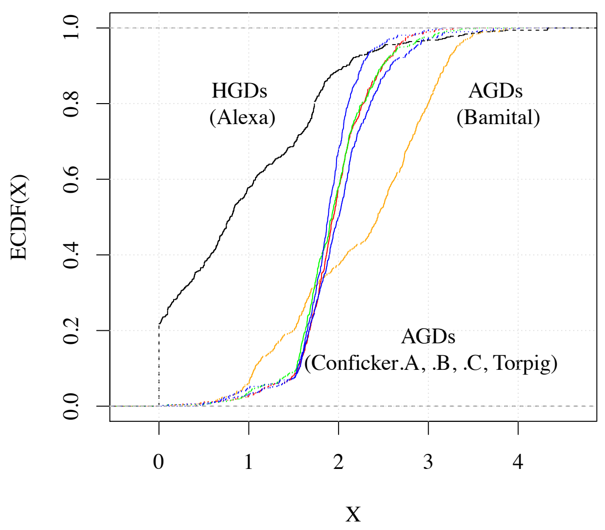

Figure 6: Mahalanobis distance ECDF for different datasets. A KS statistical test to compare distinct AGD distributions against the HGD distribution yields p-values close to zero, confirming that they are drawn from diverse processes.

Alexa top 100,000 domains. The plot shows that each datasets of AGDs and HGDs have different distribution: This confirms that our linguistic features are well suited to perform the discrimination. Interestingly, each AGD dataset has a distinctive distribution: **Conficker** and **Torpig** have the same linguistic features, which differ from the linguistic features of **Bamital** (and, of course, from the non-AGD domains). We may argue that the DGAs that are responsible of such characteristic features are also different, whereas **Conficker** and **Torpig** rely on DGAs with very similar linguistic characteristics. Then, we verify which fraction of AGDs passes the filter and reaches the

AGD Clustering (Λ) step or the **Labeler** (λ). The results obtained are reported in the first column of Tab. 1 and show that roughly half of the domains would not contribute to the generation of the clusters: The conservative settings ensure that only the domains that exhibit the linguistic features more remarkably are used for clustering. Ultimately, most of the true AGD domains will be labeled as such before reaching the **Labeler**. Overall, Phoenix has a recall of 81.4 to 94.8%.

6.2.2 Step 2: AGD Clustering

We ran Phoenix on our dataset and, after the first run of the DBSCAN clustering, we obtained the similarity matrix depicted in Fig. 5. Even with one run of the algorithm, we can already see some interesting groups of domains that are similar. The annotations on the figure are clarified in the reminder of this section.

Reality Check. We searched for qualitative ground truth that could confirm the usefulness of the clusters obtained by running Phoenix on our dataset. To this end, we queried Google for the IP addresses of each AGD cluster to perform manual labeling of such clusters with evidence about the malware activity found by other researchers.

We gathered evidence about a cluster with 33,771 domains allegedly used by Conficker (see also Fig. 5) and another cluster with 3870 domains used by Bamital. A smaller cluster of 392 domains was assigned to SpyEye, and two clusters of 404 and 58 domains, respectively, were assigned to Palevo. We were unable to find information to label the remaining six clusters as related to known malware.

This reality check helped us confirming that we successfully isolated domains related to botnet activities and IP addresses hosting C&C servers. The remainder of this section evaluates how well such isolation performs in general settings (i.e., not on a specific dataset).

Sensitivity From **γ***.* We evaluated the sensitivity of the clustering result to the γ threshold used for cluster pruning. To this end, we studied the number of clusters generated with varying values of γ. A steady number of cluster indicates low sensitivity from this parameter, which is a desirable property. Moreover, abrupt changes of the number of clusters caused by certain values of γ can be used as a decision boundary to this parameter. Fig. 7 shows such a change at $\gamma = 2.8$.

We also assessed how γ influences the quality of the clustering to find safety bounds of this parameter within which the resulting clusters do not contain spurious elements. In other words, we want to study the influence of γ on the cluster features calculated within each cluster. To this end, we consider the cluster features for which a simple metric can be easily defined: **CF2 (Chosen Prefix Length Range)**, **CF4 (Chosen Prefix Numerical Characters Ratio Range)** and **CF5 (Public Suffix Set)**. A clustering quality is high if all the clusters contain domains that are uniform with respect to these features (e.g., each cluster contain elements with common public suffix set or length). We quantify such "uniformity" as the entropy of each features. As Fig. 7 shows, all the features reflect an abrupt change in the uniformity of the clusters around $\gamma = 2.8$, which corroborates the above finding.

Malware	$d_{Mah} > \Lambda$	$d_{Mah} > \lambda$
	Pre-clustering selection	Recall
Conficker.A	46.5%	**93.4%**
Conficker.B	47.2 %	**93.7%**
Conficker.C	52.9 %	**94.8%**
Torpig	34.2%	**93.0%**
Bamital	62.3%	**81.4%**

Table 1: AGD pre-clustering selection and recall.

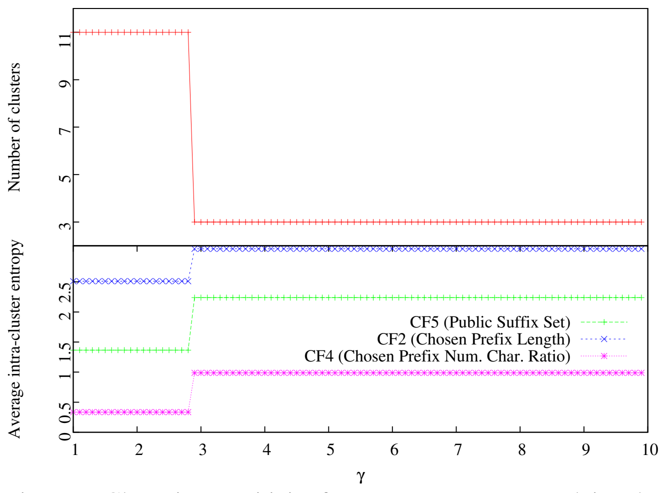

Figure 7: Clustering sensitivity from parameter γ. By studying the number of clusters (top) and the average intra-cluster entropy over **CF2**, **4**, **5** (bottom), we can choose the best $\gamma \in (0, 2.8)$.

In conclusion, values of γ outside $(0, 2.8)$ do not allow the clustering algorithm to optimally separate clusters of domains.

Correctness. Our claim is that the clustering can distinguish between domains generated by different DGAs by means of the representative IPs used by such DGAs (which are likely to be the C&C servers). To confirm this claim in a robust way, we evaluate the quality of the clustering with respect to features other than the IP addresses. In this way, we can show that our clustering tells different DGAs apart, regardless of the IP addresses in common. In other words, we show that our clustering is independent from the actual IP addresses used by the botnets but it is capable of recognizing DGAs in general.

To this end, we ignore **CF1** and calculate the features **CF2-5** of each cluster and show that they are distributed differently between any two clusters. We quantify this difference by means of the p-value of the Kolmogorov-Smirnov (KS) statistical test, which tells how much two samples (i.e., our **CF2-5** calculated for each couple of clusters) are drawn from two different stochastic processes (i.e., they belong to two different clusters). p-values toward 1 indicate that two clusters are not well separated, because they comprise domains that are likely drawn from the same distribution. On the other hand, p-values close to zero indicate sharp separation.

The results summarized in Tab. 2 confirm that most of the clusters are well separated, because their p-value is close to 0. In particular 9 of our 11 clusters are highly dissimilar, whereas two clusters are not distinguishable from each other (Clusters 2 and 4). From a manual analysis of these two clusters we can argue that a common DGA is behind both of them, even if there is no strong evidence (i.e. DNS features) of this being the case. Cluster 2 include domains such as `46096.com` and `04309.com`, whereas two samples from Cluster 4 are `88819.com` and `19527.com`.

6.3 AGD Detection Evaluation

We want to evaluate qualitatively how well the **AGD Detection** module is able to assign the correct labels to previously unseen suspicious domains. To this end, we first run the **AGD Discovery** module using the historical domain-to-IP relations extracted from the ISC/SIE database for the domains indicated as generically malicious by the malicious domain filter (which is Exposure in our case). Once this module produced the clusters, we validated the outcome of the **AGD Detection** against another (random) split of the same type of data extracted from the ISC/SIE dataset (never observed before).

Previously unseen domains

hy613.cn	5ybdiv.cn	73it.cn	39yq.cn
69wan.cn	hy093.cn	08hhwl.cn	hy267.cn
hy673.cn	onkx.cn	xmsyt.cn	fyf123.cn
watdj.cn	dhjy6.cn	algxy.cn	g3pp.cn

Cluster 9 (Palevo)

pjrn3.cn	3dcyp.cn	x0v7r.cn	0iwzc.cn
0bc3p.cn	hdnx0.cn	9q0kv.cn	4qy39.cn
5vm53.cn	7ydzr.cn	fyj25.cn	m5qwz.cn
qwr7.cn	xq4ac.cn	ygb55.cn	v5pgb.cn

Previously unseen domains

dky.com	ejm.com	eko.com	blv.com
efu.com	elq.com	bqs.com	dqu.com
bec.com	dpl.com	eqy.com	dyh.com
dur.com	bnq.com	ccz.com	ekv.com

Cluster 10 (Palevo)

uon.org	jhg.org	eks.org	kxc.com
mzo.net	zuh.com	bwn.org	khz.net
zuw.org	ldt.org	lxx.net	epu.org
ntz.com	cbv.org	iqd.com	nrl.net

Figure 8: Labeling of previously unseen domains (see Appendix A).

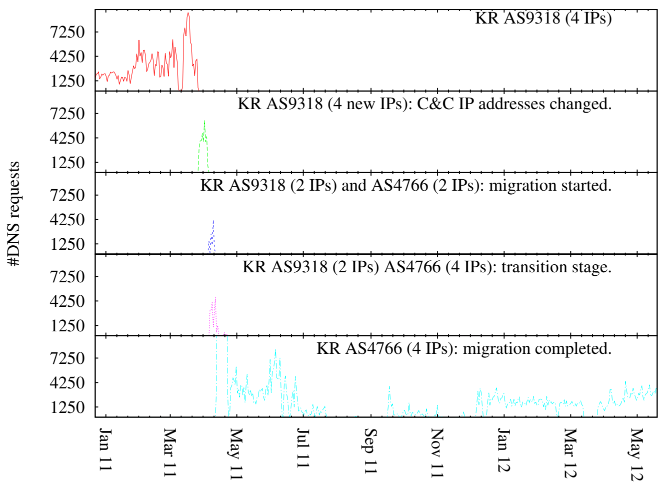

Figure 9: **Bamital**: Migration of C&C from AS9318 to AS4766.

The result of the **AGD Detection** is a list of previously unseen domains, assigned to a cluster (i.e., a DGA). Some examples of previously unseen domains are depicted in Fig. 8 along with some samples of the clusters where they have been assigned to. These examples show that Phoenix is capable of assigning the correct cluster to unknown suspicious domains. Indeed, despite the variability of the eTLD, which is commonly used as anecdotal evidence to discriminate two botnets, our system correctly models the linguistic features and the domain-to-IP historical relations and performs a better labeling. In the second case the domains were all registered under .cn, and it is also clear that they share the same generation mechanism.

6.4 Intelligence and Insights

In this section, we describe two use cases of the **Intelligence and Insights** module, which provides the analyst with valuable knowledge from the outputs of the other modules. The correctness of the conclusions drawn from this module is predicated on the correctness of the two upstream modules, already discussed in prevoius sections.

Unknown DGA Recognition From Scarce Data. Our system is designed to automatically label the malicious domains related to botnet activities. This is done by using the information of the DNS traffic related to them. Interestingly, some conclusions can be drawn on previously unseen domains even in the unlucky case that such information is missing (i.e., when no DNS data is available).

While asking on mailing lists for information about sinkholed IPs, we received an inquiry by a group of researchers on February 9th. They had found a previously unseen list of AGDs which resembled no known botnet. Such list was the only information that they provided us with. Phoenix labeled these domains with the fingerprints of a Conficker cluster. This allowed the researchers to conduct further investigation, which eventually confirmed that it was Conficker.B.

In conclusion, starting from the sole knowledge of a list of malicious domains that Phoenix had never seen before, we discovered that, according to our datasets, the only DGA able to produce domains with that linguistic features was the DGA associated with **Conficker**.

2	e-12									
3	e-8	e-9								
4	e-12	1.00	e-9							
5	e-26	e-32	e-5	e-36						
6	e-33	e-39	e-27	e-44	0.00					
7	e-3	e-4	0.01	e-3	e-16	e-33				
8	e-20	e-11	0.14	e-12	e-55	e-276	e-5			
9	e-18	e-5	e-4	e-5	e-142	e-305	e-4	e-16		
10	e-14	e-23	e-19	e-24	e-50	e-52	e-17	e-46	e-46	
11	e-22	e-28	0.61	e-31	e-163	0.00	e-8	e-27	e-82	e-52
	1	2	3	4	5	6	7	8	9	10

Table 2: The low p-values of the pairwise KS test between the lengths of the chosen prefix of the elements within each couple of clusters indicate that the elements of each pair of clusters have diverse features (we hereby exemplify the length of the chosen prefixes for the purpose of visualization). Thus, the clusters are well separated.

Time Evolution. Associating AGDs to the activity of a specific botnet allows to gather further information on that botnet, by using the DGA fingerprints as a "lookup index" to make precise queries. We can track the behavior of a botnet to study its evolution over time.

For instance, given a DGA fingerprint or AGD sample, we can select the domains of the corresponding cluster $\mathbb{E}_{DGA}$ and partition this set at different granularity (e.g., IPs or ASs) by considering the *exact* set of IPs (or ASs) that they point to. Given the activity that we want to monitor, for instance, the DNS traffic of that botnet, we can then plot one time series for each partition. In our example, we count the number of DNS requests seen for the domains in that partition at a certain sampling frequency (e.g., daily). The analysis of the stacked time series generated allows to draw conclusion about the behavior over time of the botnet. Fig. 9 shows the case of (a) a migration (the botmaster moved the C&C servers from one AS to another) followed by (b) a load balancing change in the final step (the botmaster shut down 2 C&C servers thus reducing the load balancing).

In a similar vein, Fig. 10 shows an evolution that we may argue being a takedown operated by security defenders. In particular, at the beginning the botnet C&C backend was distributed across three ASs in two countries (United States and Germany). Armed with the knowledge that the IPs in AS2637 and AS1280 are operated by computer security laboratories, we discover that this "waterfall" pattern concludes into a sinkhole. Without knowledge of the sinkholed IPs, we could still argue that the C&C was moved from some ASs to some other ASs.

The aforementioned conclusions were drawn by a semi-automatic analysis and can be interpreted and used as novel intelligence knowledge. The labels of the DGAs produced by Phoenix were fundamental to perform this type of analysis.

7. DISCUSSION

Despite the good results, Phoenix has some limitations. Previous work leveraged NXDOMAIN responses to identify those AGDs that the botmaster did not register yet. This allows early detection of DGA

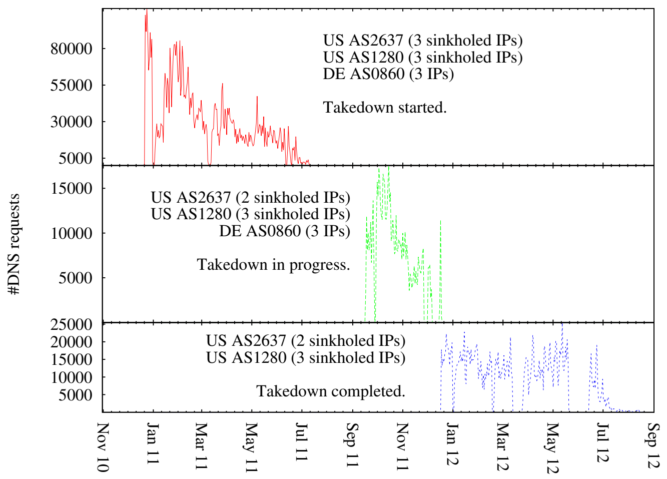

Figure 10: **Conficker**: Evolution that resembles a C&C takedown: the C&C had 3 IPs in AS0860 and 3 sinkholed IPs in AS2637.

activities, because the bots yield overwhelming amounts of NXDOMAIN replies. Our system, instead, requires *registered* AGDs to function. Therefore, it is fed with data that takes longer collection periods. This results in a less-responsive detection of previously unseen DGAs. The advantage is that, differently from previous work, we can fingerprint the DGAs and, more importantly, we lift the observation point such that Phoenix is easier to adopt. Indeed, we believe that not using NXDOMAIN replies represents a strength of our work, as it makes our system profoundly different from previous work in ease of deployment and testing under less-constraining requirements.

The linguistic features computed on the domain names, to decide whether they are automatically generated or not, capture the likelihood that a given domain targets English-speaking users. Taking into account different languages, possibly featuring totally different sounds like Chinese or Swedish, as well as different encondings, such as UTF8, would pose some challenges. In particular, computing language-independent features with a multilingual dictionary would flatten the underlying distributions, rendering the language features less discriminant. To tackle this limitation, a possible solution consists in inferring the linguistic target of a given domain (e.g, via TLD analysis or whois queries) so to evaluate its randomness according to the correct dictionary.

Future DGAs may attempt to evade our linguistic features by creating *pronounceable* AGDs. Besides the fact that, to the best of our knowledge, no such DGAs exist, creating *large amounts* of pronounceable domains is difficult: Such DGAs would have a narrow randomization space, which violates the design goals of domain flux [11, 19].

8. RELATED WORK

This section extends our overview of the state of the art in §2. Botnet mitigation is a widely covered topic in the literature [4].

Phoenix may seem similar to [16]. However, the focus of that work is on domains that are malicious from the viewpoint of the victims of attacks perpetrated through botnets (e.g., phishing, spam, drive-by download). Phoenix focuses on AGDs and, for this reason, it models the features of the DNS layer between bots and C&C servers. Moreover, the detection method of [16] is based on supervised learning, whereas Phoenix is unsupervised.

Bilge et al. [7] proposed a system that detects C&C communications from NetFlow data analysis. Using NetFlow data overcomes the problems of gathering raw network traffic and of large-scale processing. However, NetFlow poses challenges on how to use such summarized information (which, for instance, includes no packet payload) to tell legitimate and C&C traffic apart.

Neugschwandtner et al. [13] proposed a system that detects C&C failover strategies with techniques based on multi-path exploration. The system explores the behaviour of malware samples during simulated network failures. Backup C&C servers and AGDs are so unveiled, leading to new blacklists. The approach is very promising, although it may produce misleading results when the malware behaviour depends on time-dependent information.

Bilge et al. [6] proposed Exposure, a large-scale, passive DNS analysis technique to detect domains associated with malicious activities, including botnet C&C. The technique is based on the observation that malicious domains exhibit peculiar DNS behaviors. 15 features, ranging from time series to TTL values-based features, are computed and used to feed a classifier trained with real-world labelled data.

Although systems like Exposure (e.g., Notos [1]) rely on local recursive DNS, Kopis [2] analyzes DNS traffic collected from a global vantage point at the upper DNS hierarchy. The system introduces novel features, such as requester diversity, requester profile and resolved-IPs reputation, to leverage the global visibility and detect malicious domains. As the authors themselves notice, Kopis is ineffective on AGDs, because of their short lifespan.

Yadav et al. [21] were the first who addressed the problem of AGDs, later republished in [22]: The authors leverage the randomization of AGD names to distinguish them from HGDs. Linguistic features capturing the distribution of alphanumeric characters and bigrams are computed over domain *sets*, which are then classified as sets of AGDs or HGDs. They rely on *supervised* learning, requiring datasets of positive and negative samples. The work explores different strategies to group domain in sets before feeding them to the classifier: per-second-level-domain, per-IP and per-component. The first strategy groups the domains according to their second-level-domain, the second strategy to the IPs they resolve to, the third to the bipartite domain-IP graph components. Our work differs from [21] as we require no labeled datasets of AGDs to be bootstrapped, thus is able to find sets of AGDs with no prior knowledge. Moreover, our system classifies domains with single granularity, without the necessity of performing error-prone groupings.

Yadav and Reddy [20] extend [21] and introduce NXDOMAINs to speedup the detection of AGDs: *registered* AGDs are recognized because they are queried by any given client after a series of NXDOMAIN responses. The work differs from ours substantially, and requires DNS datasets that include the IP addresses of the querying clients. Moreover, the approach seems fragile on sampled datasets, which is a required step when dealing with high-traffic networks.

9. CONCLUSION

In this paper we described Phoenix, a novel technique able discover DGAs by telling AGDs and HGDs apart, detect previously unknown AGDs, and provide insightful intelligence (e.g., the tracking and monitoring of AGD-based C&C domains over time). For this, Phoenix, which we have extensively evaluated, combines linguistic features and publicly available DNS traffic information, and, contrary to the state-of-the-art, preserves client-side privacy (i.e., our approach does not rely on clients' IPs, it is not affected by NAT and DHCP leases nor it requires specific deployment contexts).

References

[1] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *USENIX Security*, pages 18–18. USENIX Association, 2010.

[2] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of the 20th USENIX Security Symposium, USENIX Security*, volume 11, pages 27–27, 2011.

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

[4] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 299–304. IEEE, 2009.

[5] Todd M Bailey and Ulrike Hahn. Determinants of wordlikeness: Phonotactics or lexical neighborhoods? *Journal of Memory and Language*, 44(4):568–591, 2001.

[6] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive DNS analysis. In *Proceedings of NDSS*, 2011.

[7] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *ACSAC*, pages 129–138. ACM, 2012.

[8] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[9] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001–. Accessed: 28/01/2013.

[11] Felix Leder and Tillmann Werner. Know your enemy: Containing conficker. *The Honeynet Project, University of Bonn, Germany, Tech. Rep*, 2009.

[12] Louis Marinos and Andreas Sfakianakis. ENISA Threat Landscape. Technical report, ENISA, September 2012.

[13] Matthias Neugschwandtner, Paolo Milani Comparetti, and Christian Platzer. Detecting malware's failover C&C strategies with Squeeze. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 21–30. ACM, 2011.

[14] Mark Newman. Networks: an introduction. 2010.

[15] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. Fluxor: Detecting and monitoring fast-flux service networks. In *DIMVA*, pages 186–206. Springer-Verlag, 2008.

[16] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early detection of malicious flux networks via large-scale passive DNS analysis. *Dependable and Secure Computing, IEEE Transactions on*, 9(5):714–726, 2012.

[17] Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 65–79. IEEE, 2012.

[18] Robert J Scholes. *Phonotactic grammaticality*. Number 50. Mouton, 1966.

[19] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *CCS '09*, pages 635–647. ACM, 2009.

[20] Sandeep Yadav and AL Narasimha Reddy. Winning with DNS failures: Strategies for faster botnet detection. *Security and Privacy in Communication Networks*, pages 446–459, 2012.

[21] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement*, pages 48–61. ACM, 2010.

[22] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/ACM TON*, 20(5):1663–1677, 2012.

A. SUPPLEMENTARY INFORMATION

A representative excerpt of the clustering produced by Phoenix from our dataset.

Cluster 6 (Bamital)

50e7f66b0242e579f8ed4b8b91f33d1a.co.cc
bad61b6267f0e20d08154342ef09f152.co.cc
62446a1af3f85b93f4eef982d07cc492.co.cc
0d1a81ab5bdfac9c8c6f6dd4278d99fb.co.cc
f1dad9a359ba766e9f5ec392426ddd30.co.cc
295e2484bddd43bc43387950a4b5da16.co.cc
501815bd2785f103d22e1becb681aa48.co.cc
341af50eb475d1730bd6734c812a60a1.co.cc
49b24bf574b7389bd8d5ba83baa30891.co.cc
a7e3914a88e3725ddafbbf67444cd6f8.co.cc

Cluster 9 (Palevo)

7cj1b.cn	ff88567.cn	ef44ee.cn
fwjp0.cn	0bc3p.cn	9i230.cn
3dcyp.cn	azeifko.cn	fyyxqftc.cn
	hfju38djfhjdi3kd.cn	

Cluster 10 (Palevo)

ewn.net	wyp.net	ews.net	kpk.net
khz.net	uon.org	lxx.net	kxc.com
yhv.com	nrl.net		

Cluster 11 (Conficker)

byuyy.biz	jbkxbxublgn.biz
kpqzk.org	tcmsrdm.org
lvzqxymji.org	fbhwgmb.info
aeyyiujxs.org	psaehtmx.info
vdrmgyxq.biz	mmdbby.biz