From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware

Manos Antonakakis[‡,*], Roberto Perdisci[†,*], Yacin Nadji[*],
Nikolaos Vasiloglou[‡], Saeed Abu-Nimeh[‡], Wenke Lee[*] and David Dagon[*]
‡*Damballa Inc.*, †*University of Georgia*
{*manos,nvasil,sabunimeh*}*@damballa.com , perdisci@cs.uga.edu*
∗*Georgia Institute of Technology*
{*yacin.nadji, wenke*}*@cc.gatech.edu, dagon@sudo.sh*

Abstract

Many botnet detection systems employ a blacklist of known command and control (C&C) domains to detect bots and block their traffic. Similar to signature-based virus detection, such a botnet detection approach is static because the blacklist is updated only after running an external (and often manual) process of domain discovery. As a response, botmasters have begun employing domain generation algorithms (DGAs) to dynamically produce a large number of random domain names and select a small subset for actual C&C use. That is, a C&C domain is randomly generated and used for a very short period of time, thus rendering detection approaches that rely on static domain lists ineffective. Naturally, if we know how a domain generation algorithm works, we can generate the domains ahead of time and still identify and block botnet C&C traffic. The existing solutions are largely based on reverse engineering of the bot malware executables, which is not always feasible.

In this paper we present a new technique to detect randomly generated domains without reversing. Our insight is that most of the DGA-generated (random) domains that a bot queries would result in Non-Existent Domain (NXDomain) responses, and that bots from the same botnet (with the same DGA algorithm) would generate similar NXDomain traffic. Our approach uses a combination of clustering and classification algorithms. The clustering algorithm clusters domains based on the similarity in the make-ups of domain names as well as the groups of machines that queried these domains. The classification algorithm is used to assign the generated clusters to models of known DGAs. If a cluster cannot be assigned to a known model, then a new model is produced, indicating a new DGA variant or family. We implemented a prototype system and evaluated it on real-world DNS traffic obtained from large ISPs in North America. We report the discovery of twelve DGAs. Half of them are variants of known (botnet) DGAs, and the other half are brand new DGAs that have never been reported before.

1 Introduction

Botnets are groups of malware-compromised machines, or *bots*, that can be remotely controlled by an attacker (the *botmaster*) through a *command and control* (C&C) communication channel. Botnets have become the main platform for cyber-criminals to send spam, steal private information, host phishing web-pages, etc. Over time, attackers have developed C&C channels with different network structures. Most botnets today rely on a centralized C&C server, whereby bots query a predefined C&C domain name that resolves to the IP address of the C&C server from which commands will be received. Such centralized C&C structures suffer from the *single point of failure* problem because if the C&C domain is identified and taken down, the botmaster loses control over the entire botnet.

To overcome this limitation, attackers have used P2P-based C&C structures in botnets such as Nugache [35], Storm [38], and more recently Waledac [39], Zeus [2], and Alureon (a.k.a. TDL4) [12]. While P2P botnets provide a more robust C&C structure that is difficult to detect and take down, they are typically harder to implement and maintain. In an effort to combine the simplicity of centralized C&Cs with the robustness of P2P-based structures, attackers have recently developed a number of botnets that locate their C&C server through *automatically generated* pseudo-random domains names. In order to contact the botmaster, each bot periodically executes a *domain generation algorithm* (DGA) that, given a random seed (e.g., the current date), produces a list of *candidate* C&C domains. The bot then attempts to resolve these domain names by sending DNS queries until one of the domains resolves to the IP address of a C&C server. This strategy provides a remarkable level of *agility* because even if one or more C&C domain names or IP addresses are identified and taken down, the bots will eventually get the IP address of the relocated C&C server via DNS queries to the next set of automatically generated domains. Notable examples of DGA-

based botnets (or DGA-bots, for short) are Bobax [33], Kraken [29], Sinowal (a.k.a. Torpig) [34], Srizbi [30], Conficker-A/B [26], Conficker-C [23] and Murofet [31]. A defender can attempt to reverse engineer the bot malware, particularly its DGA algorithm, to pre-compute current and future candidate C&C domains in order to detect, block, and even take down the botnet. However, reverse engineering is not always feasible because the bot malware can be updated very quickly (e.g., hourly) and obfuscated (e.g., encrypted, and only decrypted and executed by external triggers such as time).

In this paper, we propose a novel detection system, called Pleiades, to identify DGA-based bots within a monitored network without reverse engineering the bot malware. Pleiades is placed "below" the local recursive DNS (RDNS) server or at the edge of a network to monitor DNS query/response messages from/to the machines within the network. Specifically, Pleiades analyzes DNS queries for domain names that result in *Name Error* responses [19], also called `NXDOMAIN` responses, i.e., domain names for which no IP addresses (or other resource records) exist. In the remainder of this paper, we refer to these domain names as NXDomains. The focus on NXDomains is motivated by the fact that modern DGA-bots tend to query large sets of domain names among which relatively few successfully resolve to the IP address of the C&C server. Therefore, to automatically identify DGA domain names, Pleiades searches for relatively large clusters of NXDomains that (i) have similar syntactic features, and (ii) are queried by multiple potentially compromised machines during a given epoch. The intuition is that in a large network, like the ISP network where we ran our experiments, multiple hosts may be compromised with the same DGA-bots. Therefore, each of these compromised assets will generate several DNS queries resulting in NXDomains, and a subset of these NXDomains will likely be queried by more than one compromised machine. Pleiades is able to automatically identify and filter out "accidental", user-generated NXDomains due to typos or mis-configurations. When Pleiades finds a cluster of NXDomains, it applies statistical learning techniques to build a model of the DGA. This is used later to detect future compromised machines running the same DGA and to detect *active domain names* that "look similar" to NXDomains resulting from the DGA and therefore probably point to the botnet C&C server's address.

Pleiades has the advantage of being able to discover and model new DGAs without labor-intensive malware reverse-engineering. This allows our system to detect new DGA-bots before any sample of the related malware family is captured and analyzed. Unlike previous work on DNS traffic analysis for detecting malware-related [4] or malicious domains in general [3, 6], Pleiades leverages *throw-away traffic* (i.e., unsuccessful DNS resolutions) to (1) discover the rise of new DGA-based botnets, (2) accurately detect bot-compromised machines, and (3) identify and block the active C&C domains queried by the discovered DGA-bots. Pleiades achieves these goals by monitoring the DNS traffic in local networks, without the need for a large-scale deployment of DNS analysis tools required by prior work.

Furthermore, while botnet detection systems that focus on network flow analysis [13, 36, 44, 46] or require deep packet inspection [10, 14] may be capable of detecting compromised machines within a local network, they do not scale well to the overwhelming volume of traffic typical of large ISP environments. On the other hand, Pleiades employs a *lightweight* DNS-based monitoring approach, and can detect DGA-based malware by focusing on a small fraction of all DNS traffic in an ISP network. This allows Pleiades to scale well to very large ISP networks, where we evaluated our prototype system.

This paper makes the following contributions:

- We propose Pleiades, the first DGA-based botnet identification system that efficiently analyzes streams of unsuccessful domain name resolutions, or NXDomains, in large ISP networks to automatically identify DGA-bots.

- We built a prototype implementation of Pleiades, and evaluated its DGA identification accuracy over a large labeled dataset consisting of a mix of NXDomains generated by four different known DGA-based botnets and NXDomains "accidentally" generated by typos or mis-configurations. Our experiments demonstrate that Pleiades can accurately detect DGA-bots.

- We deployed and evaluated our Pleiades prototype in a large *production* ISP network for a period of 15 months. Our experiments discovered twelve new DGA-based botnets and enumerated the compromised machines. Half of these new DGAs have never been reported before.

The remainder of the paper is organized as follows. In Section 2 we discuss related work. We provide an overview of Pleiades in Section 3. The DGA discovery process is described in Section 4. Section 5 describes the DGA classification and C&C detection processes. We elaborate on the properties of the datasets used and the way we obtained the ground truth in Section 6. The experimental results are presented in Section 7 while we discuss the limitations of our systems in Section 8. We conclude the paper in Section 9.

2 Related Work

Dynamic domain generation has been used by malware to evade detection and complicate mitigation, e.g., Bobax, Kraken, Torpig, Srizbi, and Conficker [26]. To uncover the underlying domain generation algorithm (DGA), researchers often need to reverse engineer the bot binary. Such a task can be time consuming and requires advanced reverse engineering skills [18].

The infamous Conficker worm is one of the most aggressive pieces of malware with respect to domain name generation. The "C" variant of the worm generated 50,000 domains per day. However, Conficker-C only queried 500 of these domains every 24 hours. In older variants of the worm, A and B, the worm cycled through the list of domains every three and two hours, respectively. In Conficker-C, the length of the generated domains was between four and ten characters, and the domains were distributed across 110 TLDs [27].

Stone-Gross et al. [34] were the first to report on domain fluxing. In the past, malware used IP fast-fluxing, where a single domain name pointed to several IP addresses to avoid being taken down easily. However, in domain fluxing malware uses a domain generation algorithm to generate several domain names, and then attempt to communicate with a subset of them. The authors also analyzed Torpig's DGA and found that the bot utilizes Twitter's API. Specifically, it used the second character of the most popular Twitter search and generated a new domain every day. It was updated to use the second character of the 5^{th} most popular Twitter search. Srizbi [40] is another example of a bot that utilizes a DGA by using unique magic number. Researchers identified several unique magic numbers from multiple copies of the bot. The magic number is XOR'ed with the current date and a different set of domains is generated. Only the characters "q w e r t y u i o p a s d f" are used in the generated domain names.

Yadav et. al. proposed a technique to identify botnets by finding randomly generated domain names [42], and improvements that also include NXDomains and temporal correlation [43]. They evaluated their approaches by automatically detecting Conficker botnets in an offline dataset from a Tier-1 ISP in South Asia in the first paper, and both the ISP dataset and a university's DNS logs in the second.

Villamarin-Salomon and Brustoloni [37] compared two approaches to identify botnet C&Cs. In their first approach, they identified domains with high query rates or domains that were temporally correlated. They used Chebyshev's inequality and Mahalanobis distance to identify anomalous domains. In their second approach, they analyzed recurring "dynamic" DNS replies with NXDomain responses. Their experiments showed that the first approach was ineffective, as several legitimate services use DNS with short time-to-live (TTL) values. However, their second approach yielded better detection and identified suspicious C&C domains.

Pleiades differs from the approaches described above in the following ways. **(A)** Our work models five different types of bot families including Conficker, Murofet, Sinowal, and Bobax. **(B)** We model these bot families using two clustering techniques. The first utilizes the distribution of the characters and 2-grams in the domain name. The second relies on historical data that shows the relationship between hosts and domain names. **(C)** We build a classification model to predict the maliciousness of domains that deviate from the two clustering techniques.

Unlike previous work, our approach does not require active probing to maintain a fresh list of legitimate domains. Our approach does not rely on external reputation databases (e.g., DNSBLs); instead, it only requires access to local DNS query streams to identify new clusters of DGA NXDomains. Not only does our approach identify new DGAs, but it also builds models for these DGAs to classify hosts that will generate similar NXDomains in the future. Furthermore, among the list of identified domains in the DGAs, our approach pinpoints the C&C domains. Lastly, we note that our work is complementary to the larger collection of previous research that attempts to detect and identify malicious domain names, e.g., [3, 4].

3 System Overview

In this section, we provide a high-level overview of our DGA-bot detection system Pleiades. As shown in Figure 1, Pleiades consists of two main modules: a *DGA Discovery* module, and a *DGA Classification and C&C Detection* module. We discuss the roles of these two main modules and their components, and how they are used in coordination to *actively learn* and update DGA-bot detection models. We describe these components in more detail in Sections 4 and 5.

3.1 DGA Discovery

The *DGA Discovery* module analyzes streams of unsuccessful DNS resolutions, as seen from "below" a local DNS server (see Figure 1). All NXDomains generated by network users are collected during a given epoch (e.g., one day). Then, the collected NXDomains are clustered according to the following two similarity criteria: (1) the domain name strings have similar statistical characteristics (e.g., similar length, similar level of "randomness", similar character frequency distribution, etc.) and (2) the domains have been queried by overlapping sets of hosts. The main objective of this NXDomain clustering process is to group together domain names that likely are automatically generated by the same algorithm running on multiple machines within the monitored network.

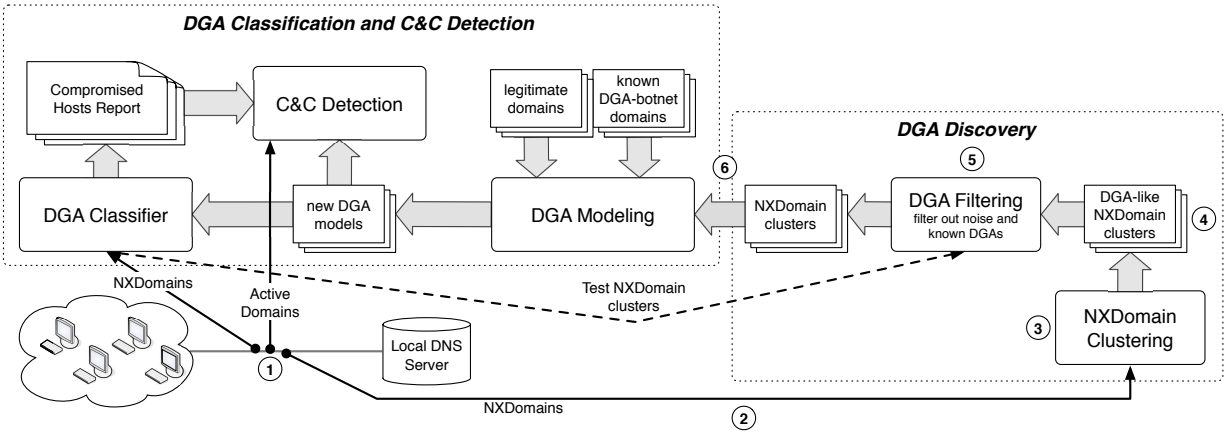

Figure 1: A high level overview of Pleiades.

Naturally, because this clustering step is unsupervised, some of the output NXDomain clusters may contain groups of domains that happen to be similar by chance (e.g., NXDomains due to common typos or to misconfigured applications). Therefore, we apply a subsequent filtering step. We use a supervised *DGA Classifier* to prune NXDomain clusters that appear to be generated by DGAs that we have previously discovered and modeled, or that contain domain names that are similar to popular legitimate domains. The final output of the *DGA Discovery* module is a set of NXDomain clusters, each of which likely represents the NXDomains generated by previously unknown or not yet modeled DGA-bots.

3.2 DGA Classification and C&C Detection

Every time a new DGA is discovered, we use a supervised learning approach to build models of what the domains generated by this new DGA "look like". In particular, we build two different statistical models: (1) a statistical multi-class classifier that focuses on assigning a specific DGA label (e.g., `DGA-Conficker.C`) to the *set of NXDomains* generated by a host h_i and (2) a Hidden Markov Model (HMM) that focuses on finding *single active domain names* queried by h_i that are likely generated by a DGA (e.g., `DGA-Conficker.C`) running on the host, and are therefore good *candidate C&C domains*.

The *DGA Modeling* component receives different sets of domains labeled as `Legitimate` (i.e., "non-DGA"), `DGA-Bobax`, `DGA-Torpig/Sinowal`, `DGA-Conficker.C`, `New-DGA-v1`, `New-DGA-v2`, etc., and performs the training of the multi-class *DGA Classifier* and the HMM-based *C&C Detection* module.

The *DGA Classification* module works as follows. Similar to the *DGA Discovery* module, we monitor the stream of NXDomains generated by each client machine "below" the local recursive DNS server.

Given a subset of NXDomains generated by a machine, we extract a number of statistical features related to the NXDomain strings. Then, we ask the *DGA Classifier* to identify whether this subset of NXDomains resembles the NXDomains generated by previously discovered DGAs. That is, the classifier will either label the subset of NXDomains as generated by a known DGA, or tell us that it does not fit any model. If the subset of NXDomains is assigned a specific DGA label (e.g., `DGA-Conficker.C`), the host that generated the NXDomains is deemed to be compromised by the related DGA-bot.

Once we obtain the list of machines that appear to be compromised with DGA-based bots, we take detection one step further. While all previous steps focused on NXDomains, we now turn our attention to domain names for which we observe valid resolutions. Our goal is to identify which domain names, among the ones generated by the discovered DGA-based bots, actually resolve into a valid IP address. In other words, we aim to identify the botnet's active C&C server.

To achieve this goal, we consider all domain names that are successfully resolved by hosts which have been classified as running a given DGA, say `New-DGA-vX`, by the *DGA Classifier*. Then, we test these successfully resolved domains against an HMM specifically trained to recognize domains generated by `New-DGA-vX`. The HMM analyzes the sequence of characters that compose a domain name d, and computes the likelihood that d is generated by `New-DGA-vX`.

We use an HMM, rather than the *DGA Classifier*, because for the C&C detection phase we need to classify *single domain names*. The *DGA Classifier* is not suitable for this task because it expects as input *sets* of NXDomains generated by a given host to assign a label to the

DGA-bot running on that host. Some of the features used by the *DGA Classifier* cannot be reliably extracted from a single domain name (see Sections 4.1.1 and 5.2).

4 DGA Discovery

The *DGA Discovery* module analyzes sequences of NXDomains generated by hosts in a monitored network, and in a *completely unsupervised* way, clusters NXDomains that are being automatically generated by a DGA. We achieve this goal in multiple steps (see Figure 1). First (*Step 1*), we collect sequences of NXDomains generated by each host during an epoch E. Afterwards (*Step 2*), we split the overall set of NXDomains generated by all monitored hosts into small subsets, and translate each set into a statistical feature vector (see Section 4.1.1). We then apply the X-means clustering algorithm [24] to group these domain subsets into larger clusters of domain names that have similar *string-based* characteristics.

Separately (*Step 3*), we cluster the NXDomains based on a completely different approach that takes into account whether two NXDomains are being queried by overlapping sets of hosts. First, we build a bipartite *host association* graph in which the two sets of vertices represent distinct hosts and distinct NXDomains, respectively. A host vertex V_{h_i} is connected to an NXDomain vertex V_{n_j} if host h_i queried NXDomain n_j. This allows us to identify different NXDomains that have been queried by overlapping sets of hosts. Intuitively, if two NXDomains are queried by multiple common hosts, this indicates that the querying hosts may be running the same DGA. We can then leverage this definition of similarity between NXDomains to cluster them (see Section 4.1.3).

These two *distinct views* of similarities among NXDomains are then reconciled in a *cluster correlation* phase (*Step 4*). This step improves the quality of the final NXDomains clusters by combining the clustering results obtained in *Step 2* and *Step 3*, and reduces possible noise introduced by clusters of domains that may appear similar purely by chance, for example due to similar typos originating from different network users.

The final clusters represent different groups of NXDomains, each containing domain names that are highly likely to be generated by the same DGA. For each of the obtained NXDomain clusters, the question remains if they belong to a known DGA, or a newly discovered one. To answer this question (*Step 5*), we use the *DGA Classifier* described in Section 5.2, which is specifically trained to distinguish between sets of NXDomains generated by currently known DGAs. Clusters that match previously modeled DGAs are discarded. On the other hand, if a cluster of NXDomains does not resemble any previously seen DGAs, we identify the cluster of NXDomains as having been generated by a new, previously unknown DGA. These NXDomains will then be sent (*Step 6*) to the *DGA Modeling* module, which will update (i.e., re-train) the *DGA Classifier* component.

4.1 NXDomain Clustering

We now describe the *NXDomain Clustering* module in detail. First, we introduce the statistical features Pleiades uses to translate small sets of NXDomains into feature vectors, and then discuss how these feature vectors are clustered to find similar NXDomains.

4.1.1 Statistical Features

To ease the presentation of how the statistical features are computed, we first introduce some notation that we will be using throughout this section.

Definitions and Notation A domain name d consists of a set of labels separated by dots, e.g., `www.example.com`. The rightmost label is called the *top-level* domain (TLD or $TLD(d)$), e.g., `com`. The *second-level* domain (2LD or $2LD(d)$) represents the two rightmost labels separated by a period, e.g., `example.com`. The *third-level* domain (3LD or $3LD(d)$) contains the three rightmost labels, e.g., `www.example.com`, and so on.

We will often refer to splitting a sequence $NX = \{d_1, d_2, ..., d_m\}$ of NXDomains into a number of subsequences (or subsets) of length α, $NX_k = \{d_r, d_{r+1}, ..., d_{r+\alpha-1}\}$, where $r = \alpha(k-1)+1$ and $k = 1, 2, ..., \lfloor \frac{m}{\alpha} \rfloor$. Subscript k indicates the k-th subsequence of length α in the sequence of m NXDomains NX. Each of the NX_k domain sequences can be translated into a feature vector, as described below.

n-gram Features Given a subsequence NX_k of α NXDomains, we measure the frequency distribution of n-grams across the domain name strings, with $n = 1, .., 4$. For example, for $n = 2$, we compute the frequency of each 2-gram. At this point, we can compute the median, average and standard deviation of the obtained distribution of 2-gram frequency values, thus obtaining three features. We do this for each value of $n = 1, ..., 4$, producing 12 statistical features in total. By measuring the median, average and standard deviation, we are trying to capture the *shape* of the frequency distribution of the n-grams.

Entropy-based Features This group of features computes the entropy of the character distribution for separate domain levels. For example, we separately compute the character entropy for the 2LDs and 3LDs extracted from the domains in NX_k. To better understand how these features are measured, consider a set NX_k of α domains. We first extract the 2LD of each domain $d_i \in NX_k$, and for each domain we compute the entropy $H(2LD(d_i))$ of the characters of its 2LD. Then, we compute the average and standard deviation of the set of values $\{H(2LD(d_i))\}_{i=1...\alpha}$. We repeat this for 3LDs and for the overall domain name strings. We measure a total

of six features, which capture the "level of randomness" in the domains. The intuition is that most DGAs produce random-looking domain name strings, and we want to account for this characteristic of the DGAs.

Structural Domain Features This group of features is used to summarize information about the structure of the NXDomains in NX_k, such as their length, the number of unique TLDs, and the number of domain levels. In total, we compute 14 features. Specifically, given NX_k, we compute the average, median, standard deviation, and variance of the length of the domain names (four features), and of the number of domain levels (four features). Also, we compute the number of distinct characters that appear in these NXDomains (one feature), the number of distinct TLDs, and the ratio between the number of domains under the `.com` TLD and the number of domains that use other TLDs (two features). The remaining features measure the average, median, and standard deviation of the occurrence frequency distribution for the different TLDs (three features).

4.1.2 Clustering using Statistical Features

To find clusters of similar NXDomains, we proceed as follows. Given the set NX of all NXDomains that we observed from all hosts in the monitored network, we split NX into subsets of size α, as mentioned in Section 4.1.1. Assuming m is the number of distinct NXDomains in NX, we split the set NX into $\lfloor \frac{m}{\alpha} \rfloor$ different subsets where $\alpha = 10$.

For each of the obtained subsets NX_k of NX, we compute the aforementioned 33 statistical features. After we have translated each NX_k into its corresponding feature vector, we apply the X-means clustering algorithm [24]. X-means will group the NX_k into X clusters, where X is automatically computed by an optimization process internal to X-means itself. At this point, given a cluster $C = \{NX_k\}_{k=1..l}$ of l NXDomain subsets, we simply take the union of the NX_k in C as an NXDomain cluster.

4.1.3 Clustering using Bipartite Graphs

Hosts that are compromised with the same DGA-based malware naturally tend to generate (with high probability) partially overlapping sets of NXDomains. On the other hand, other "non-DGA" NXDomains are unlikely to be queried by multiple hosts. For example, it is unlikely that multiple distinct users make identical typos in a given epoch. This motivates us to consider NXDomains that are queried by several common hosts as similar, and in turn use this similarity measure to cluster NXDomains that are likely generated by the same DGA.

To this end, we build a sparse association matrix M, where columns represent NXDomains and rows represent hosts that query more than two of the column NXDomains over the course of an epoch. We discard hosts that query only one NXDomain to reduce the dimensionality of the matrix, since they are extremely unlikely to be running a DGA given the low volume of NXDomains they produce. Let a matrix element $M_{i,j} = 0$, if host h_i did not query NXDomain n_j. Conversely, let $M_{i,j} = w_i$ if h_i did query n_j, where w_i is a weight.

INPUT : Sparse matrix $M \in \Re^{l \times k}$, in which the rows represent l hosts and the columns represent k NXDomains.

[1] : Normalize M: $\forall j = 1,..,k \quad \sum_{i=1}^{l} M_{i,j} = 1$

[2] : Compute the similarity matrix S from M: $S = M^T \cdot M$

[3] : Compute the first ρ eigenvectors from S by eigen-decomposition.

Let $U \in \Re^{\rho \times k}$ be the matrix containing k vectors $u_1, ..., u_k$ of size ρ resulting from the eigen-decomposition of S

(a vector u_i is a reduced ρ-dimensional representation of the i-th NXDomain).

[4] : Cluster the vectors (i.e., the NXDomains) $\{u_i\}_{i=1,..,k}$ using the X-means algorithm

OUTPUT: Clusters of NXDomains

Algorithm 1: Spectral clustering of NXDomains.

All non-zero entries related to a host h_i are assigned the same weight $w_i \sim \frac{1}{k_i}$, where k_i is the number of NXDomains queried by host h_i. Clearly, M can be seen as a representation of a bipartite graph, in which a *host vertex* V_{h_i} is connected to an *NXDomains vertex* V_{n_j} with an edge of weight w_i if host h_i queried NXDomain n_j during the epoch under consideration. The intuition behind the particular method we use to compute the weights w_i is that we expect that the higher the number of unique NXDomains queried by a host h_i (i.e., the higher k_i) the less likely the host is "representative" of the NXDomains it queries. This is in a way analogous to the *inverse document frequency* used in the text mining domain [1, 7].

Once M is computed, we apply a graph partitioning strategy based on spectral clustering [21, 22], as summarized in Algorithm 1. As a first step, we compute the first ρ eigenvectors of M (we use $\rho = 15$ in our experiments), and then we map each NXDomain (each column of M) into a ρ-dimensional vector. In effect, this mapping greatly reduces the dimensionality of the NXDomain vectors from the total number of hosts (the number of rows in M) to ρ. We then used the obtained ρ-dimensional NXDomain representations and apply X-means to cluster the NXDomains based on their "host associations". Namely, NXDomains are grouped together if they have been queried by a similar set of hosts.

4.1.4 Cluster Correlation

We now have two complementary views of how the NXDomains should be grouped based on two different definitions of similarity between domain names. Nei-

ther view is perfect, and the produced clusters may still contain noise. Correlating the two results helps filter the noise and output clusters of NXDomains that are more likely to be generated by a DGA. Cluster correlation is performed in the following way.

Let $\mathscr{A} = \{A_1, .., A_n\}$ be the set of NXDomain clusters obtained by using statistical features, as described in Section 4.1.2, and $\mathscr{B} = \{B_1, .., B_m\}$ be the set of NXDomain clusters derived from the bipartite graph partitioning approach discussed in Section 4.1.3. We compute the intersection between all possible pairs of clusters $I_{i,j} = A_i \cap B_j$, for $i = 1, .., n$ and $j = 1, .., m$. All correlated clusters $I_{i,j}$ that contain less than a predefined number λ of NXDomains (i.e., $|I_{i,j}| < \lambda$) are discarded, while the remaining correlated clusters are passed to the DGA filtering module described in Section 4.2. Clusters that are not sufficiently agreed upon by the two clustering approaches are not considered for further processing. We empirically set $\lambda = 40$ in preliminary experiments.

4.2 DGA Filtering

The DGA filtering module receives the NXDomain clusters from the clustering module. This filtering step compares the newly discovered NXDomain clusters to domains generated by known DGAs that we have already discovered and modeled. If the NXDomains in a correlated cluster $I_{i,j}$ are classified as being generated by a known DGA, we discard the cluster $I_{i,j}$. The reason is that the purpose of the *DGA Discovery* module is to find clusters of NXDomains that are generated (with high probability) by a new, never before seen DGA. At the same time, this filtering step is responsible for determining if a cluster of NXDomains is too noisy, i.e., if it likely contains a mix of DGA and "non-DGA" domains.

To this end, we leverage the *DGA Classifier* described in detail in Section 5. At a high level, we can treat the *DGA Classifier* as a function that takes as input a set NX_k of NXDomains, and outputs a set of tuples $\{(l_t, s_t)\}_{t=1..c}$, where l_i is a label (e.g., `DGA-Conficker.C`), and s_i is a score that indicates how confident the classifier is on attributing label l_i to NX_k, and c is the number of different classes (and labels) that the *DGA Classifier* can recognize.

When the DGA filtering module receives a new correlated cluster of NXDomains $I_{i,j}$, it splits the cluster into subsets of α NXDomains, and then passes each of these subsets to the *DGA Classifier*. Assume $I_{i,j}$ is divided into n different subsets. From the *DGA Classifier*, we obtain as a result n sets of tuples $\{\{(l_t, s_t)\}^{(1)}_{t=1..c}, \{(l_t, s_t)\}^{(2)}_{t=1..c}, ..., \{(l_t, s_t)\}^{(n)}_{t=1..c}\}$.

First, we consider for each set of tuples $\{(l_t, s_t)\}^{(k)}_{t=1..c}$ with $k = 1, .., n$, the label $\hat{l}^{(k)}$ that was assigned the maximum score. We consider a cluster $I_{i,j}$ as too noisy if the related labels $\hat{l}^{(k)}$ are too diverse. Specifically, a cluster is too noisy when the majority label among the $\hat{l}^{(k)}, k = 1, ..n$ was assigned to less than $\theta_{maj} = 75\%$ of the n domain subsets. The clusters that do not pass the θ_{maj} "purity" threshold will be discarded. Furthermore, NXDomain clusters whose majority label is the `Legitimate` label will also be discarded.

For each remaining cluster, we perform an additional "purity" check. Let the majority label for a given cluster $I_{i,j}$ be l^*. Among the set $\{\{(l_t, s_t)\}^{(k)}_{t=1..c}\}_{k=1..n}$ we take all the scores s_t whose related $l_t = l^*$. That is, we take the confidence score assigned by the classifier to the domain subsets that have been labeled as l^*, and then we compute the average $\mu(s_t)$ and the variance $\sigma^2(s_t)$ of these scores (notice that the scores s_t are in $[0, 1]$). We discard clusters whose $\sigma^2(s_t)$ is greater than a predefined threshold $\theta_\sigma = 0.001$, because we consider the domains in the cluster as not being *sufficiently similar* to the majority label class.

At this point, if $\mu(s_t) < \theta_\mu$, with $\theta_\mu = 0.98$, we deem the NXDomain cluster to be not similar enough to the majority label class, and instead we label it as "new DGA" and pass it to the *DGA Modeling* module. On the other hand, if $\mu(s_t) \geq \theta_\mu$, we confirm the majority label class (e.g., `DGA-Conficker.C`) and do not consider it further.

The particular choice for the values of the above mentioned thresholds are motivated in Section 7.2.

5 DGA Classification and C&C Detection

Once a new DGA is reported by the *DGA Discovery* module, we use a supervised learning approach to learn how to identify hosts that are infected with the related DGA-based malware by analyzing the set of NXDomains they generate. To identify compromised hosts, we collect the set of NXDomains NX_{h_i} generated by a host, h_i, and we ask the *DGA Classifier* whether NX_{h_i} likely "belongs" to a previously seen DGA or not. If the answer is yes, h_i is considered to be compromised and will be labeled with the name of the (suspected) DGA-bot that it is running.

In addition, we aim to build a classifier that can analyze the set of active domain names, say AD_{h_i}, resolved by a compromised host h_i and reduce it to a smaller subset $CC_{h_i} \subset AD_{h_i}$ of likely C&C domains generated by the DGA running on h_i. Finally, the set CC_{h_i} may be manually inspected to confirm the identification of C&C domain(s) and related IPs. In turn, the list of C&C IPs may be used to maintain an IP blacklist, which can be employed to block C&C communications and mitigate the effects of the malware infection. We now describe the components of the DGA classification and C&C detection module in more detail.

5.1 DGA Modeling

As mentioned in Section 4.2, the NXDomain clusters that pass the *DGA Filtering* and do not fit any known DGA model are (automatically) assigned a `New-DGA-vX` label, where `X` is a unique identifier. At this point, we build two different statical models representative of `New-DGA-vX`: (1) a statistical multi-class classifier that can assign a specific DGA label to the set of NXDomains generated by a host h_i and (2) a Hidden Markov Model (HMM) that can compute the probability that a single *active* domain queried by h_i was generated by the DGA running on the host, thus producing a list of *candidate C&C domains*.

The *DGA Modeling* module takes as input the following information: (1) a list of popular legitimate domain names extracted from the top 10,000 domains according to `alexa.com`; (2) the list of NXDomains generated by running known DGA-bots in a controlled environment (see Section 6); (3) the clusters of NXDomains received from the *DGA Discovery* module. Let NX be one such newly discovered cluster of NXDomains. Because in some cases NX may contain relatively few domains, we attempt to extend the set NX to a larger set NX' that can help build better statistical models for the new DGA. To this end, we identify all hosts that "contributed" to the NXDomains clustered in NX from our sparse association matrix M and we gather all the NXDomains they generated during an epoch. For example, for a given host h_i that generated some of the domains clustered in NX, we gather all the other NXDomains domains NX'_{h_i} generated by h_i. We then add the set $NX' = \bigcup_i NX'_{h_i}$ to the training dataset (marked with the appropriate new DGA label). The reader may at this point notice that the set NX'_{h_i} may contain not only NXDomains generated by a host h_i due to running a DGA, but it may also include NXDomains "accidentally' generated by h_i. Therefore, this may introduce some noisy instances into the training dataset. However, the number of "accidental" NXDomains is typically very small, compared to the number of NXDomains generated by a DGA. Therefore, we rely on the generalization ability of the statistical learning algorithms we use to smooth away the effects of this potential source of noise. This approach works well in practice, as we will show in Section 7.

5.2 DGA Classifier

The *DGA Classifier* is based on a multi-class version of the Alternating Decision Trees (ADT) learning algorithm [9]. ADT leverages the high classification accuracy obtained by Boosting [17], while producing compact classification rules that can be more easily interpreted.

To detect hosts that are compromised with DGA-based malware, we monitor all NXDomains generated by each host in the monitored network and periodically send this information to the *DGA Classifier*. Given a set NX_{h_i} of NXDomains generated by host h_i, we split NX_{h_i} into subsets of length α, and from each of these subsets we extract a number of statistical features, as described in Section 4.1.1 If one of these subsets of NXDomains is labeled by the *DGA Classifier* as being generated by a given DGA, we mark host h_i as compromised and we add its IP address and the assigned DGA label to a malware detection report.

5.3 C&C Detection

The *C&C Detection* module is based on Hidden Markov Models (HMM) [28]. We use one distinct HMM per DGA. Given the set $NX_{\mathscr{D}}$ of domains generated by a DGA $\mathscr{D}$, we consider each domain $d \in NX_{\mathscr{D}}$ separately, and feed these domains to an HMM for training. The HMM sees the domain names simply as a sequence of characters, and the result of the training is a model $HMM_{\mathscr{D}}$ that given a new domain name s in input will output the likelihood that s was generated by $\mathscr{D}$.

We use *left-to-right* HMM as they are used in practice to decrease the complexity of the model, effectively mitigating problems related to under-fitting. The HMM's emission symbols are represented by the set of characters allowed in valid domain names (i.e., alphabetic characters, digits, '_', '-', and '.'). We set the number of hidden states to be equal to the average length of the domain names in the training dataset.

During operation, the *C&C Detection* module receives active domain names queried by hosts that have been previously classified by the *DGA Classifier* as being compromised with a DGA-based malware. Let h_i be one such host, and $\mathscr{D}$ be the DGA running on h_i. The *C&C Detection* module will send every domain s resolved by h_i to $HMM_{\mathscr{D}}$, which will compute a likelihood score $f(s)$. If $f(s) > \theta_{\mathscr{D}}$, s is flagged as a good candidate C&C domain for DGA $\mathscr{D}$.

The threshold $\theta_{\mathscr{D}}$ can be learned during the training phase. First, we train the HMM with the set $NX_{\mathscr{D}}$. Then, we use a set L of legitimate "non-DGA" domains from Alexa. For each domain $l \in L$, we compute the likelihood $f(l)$ and set the threshold $\theta_{\mathscr{D}}$ so to obtain a maximum target false positive rate (e.g., max FPs=1%).

6 Data Collection

In this section we provide an overview of the amount of NXDomain traffic we observed during a period of fifteen consecutive months (our evaluation period), starting on November 1st, 2010 and ending on January 15th, 2012. Afterwards, we discuss how we collected the domain names used to train and test our *DGA Classifier* (see Section 5).

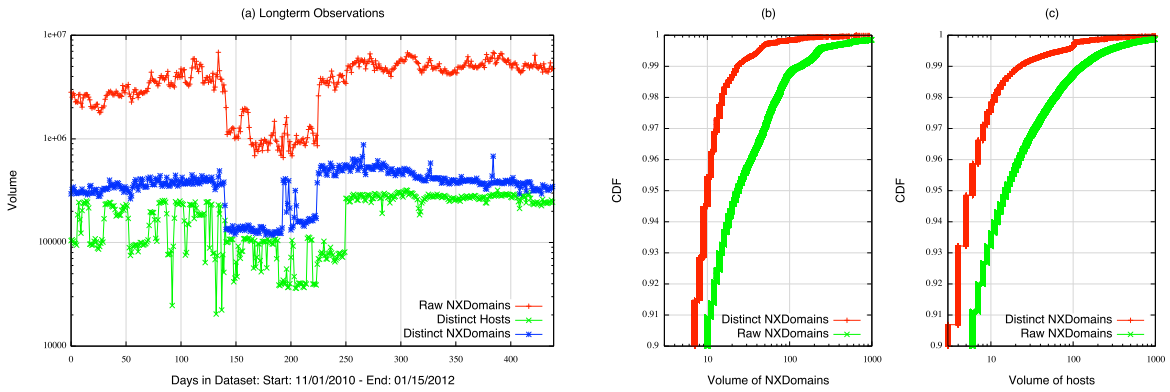

Figure 2: Observations from NXDomain traffic collected below a set of ISP recursive DNS servers over a 439 day window.

6.1 NXDomain Traffic

We evaluated Pleiades over a 15-month period against DNS traffic obtained by monitoring DNS messages to/from a set of recursive DNS resolvers operated by a large North American ISP. These servers were physically located in the US, and served (in average) over 2 million client hosts per day[1]. Our monitoring point was "below" the DNS servers, thus providing visibility on the NXDomains generated by the individual client hosts.

Figure 2(a) reports, per each day, (1) the number of NXDomains as seen in the *raw* DNS traffic, (2) the number of distinct hosts that in the considered day query at least one NXDomains, and (3) the number of distinct (de-duplicated) NXDomains (we also filter out domain names that do not have a valid effective TLD [15,19,20]). The abrupt drop in the number of NXDomains and hosts (roughly a 30% reduction) experienced between 2011-03-24 and 2011-06-17 was due to a configuration change at the ISP network.

On average, we observed about 5 millions (raw) NXDomains, 187,600 distinct hosts that queried at least one NXDomains, and 360,700 distinct NXDomains overall, per each day. Therefore, the average size of the association matrix M used to perform spectral clustering (see Section 4.1.3) was 187,600 $\times$ 360,700. However, it is worth noting that M is sparse and can be efficiently stored in memory. In fact, the vast majority (about 90%) of hosts query less than 10 NXDomains per day, and therefore most rows in M will contain only a few non-zero elements. This is shown in Figure 2(b), which reports the cumulative distribution function (CDF) for the volume of NXDomains queried by a host in the monitored network. On the other hand, Figure 2(c) shows the CDF for the number of hosts that query an NXDomain (this relates directly to the sparseness of M according to its columns).

[1]We estimated the number of hosts by computing the average number of distinct client IPs seen per day.

6.2 Ground Truth

In order to generate the ground truth to train and evaluate the *DGA Classifier* (Section 5), we used a simple approach. To collect the NXDomains generated by known DGA-based malware we used two different methods. First, because the DGA used by different variants of Conficker and by Murofet are known (derived through reverse-engineering), we simply used the respective algorithms to generate a set of domain names from each of these botnets. To obtain a sample set of domains generated by Bobax and Sinowal, whose exact DGA algorithm is not known (at least not to us), we simply executed two malware samples (one per botnet) in a VM-based malware analysis framework that only allows DNS traffic[2], while denying any other type of traffic. Overall we collected 30,000 domains generated by Conficker, 26,078 from Murofet, 1,283 from Bobax and, 1,783 from Sinowal.

Finally, we used the top 10,000 most popular domains according to `alexa.com`, with and without the `www.` prefix. Therefore, overall we used 20,000 domain names to represent the "negative" (i.e., "non-DGA") class during the training and testing of the *DGA Classifier*.

7 Analysis

In this section, we present the experimental results of our system. We begin by demonstrating Pleiades' modeling accuracy with respect to known DGAs like Conficker, Sinowal, Bobax and Murofet. Then, we elaborate on the DGAs we discovered throughout the fifteen month NXDomain monitoring period. We conclude the section by summarizing the most interesting findings from the twelve DGAs we detected. Half of them use a DGA algorithm from a known malware family. The other half,

[2]We only allowed `UDP port 53`.

Table 1: Detection results (in %) using 10-fold cross validation for different values of α.

	$\alpha = 5$ NXDomains			$\alpha = 10$ NXDomains		
Class	TP_{rate}	FP_{rate}	AUC	TP_{rate}	FP_{rate}	AUC
Bobax	95	0.4	97	**99**	**0**	**99**
Conficker	98	1.4	98	**99**	**0.1**	**99**
Sinowal	99	0.1	98	**100**	**0**	**100**
Murofet	98	0.7	98	**99**	**0.2**	**99**
Benign	96	0.7	97	**99**	**0.1**	**99**

to the best of our knowledge, have **no** known malware association.

7.1 DGA Classifier's Detection Results

In this section, we present the accuracy of the DGA classifier. We bootstrap the classifier with NXDomains from Bobax, Sinowal, Conficker-A, Conficker-B, Conficker-C and Murofet. We test the classifier in two modes. The first mode is bootstrapped with a "super" Conficker class composed of an equal number of samples from Conficker-A, Conficker-B and Conficker-C classes and another with each Conficker variant as its own class. As we mentioned in Section 5.2, the DGA classifier is based on a multi-class version of the Alternating Decision Trees (ADT) learning algorithm [9]. We build the vectors for each class by collecting NXDomains from one day of Honeypot traffic (in the case of Sinowal and Bobax) and one day of NXDomains produced by the DGAs for Conficker-A, Conficker-B, Conficker-C and Murofet. Finally, the domain names that were used to represent the benign class were the first 10,000 Alexa domain names with and without the `www.` child labels.

From the raw domain names in each of the classes, we randomly selected 3,000 sets of cardinality α. As a reminder, the values of α that we used were two, five, ten and 30. This was to build different training datasets in order to empirically decide which value of α would provide the best separation between the DGA models.

We generated additional testing datasets. The domain names we used in this case were from each class as in the case of the training dataset but we used different days. We do that so we get the minimum possible domain name overlap between the training and testing datasets. We evaluate the training datasets using two methods: 10-fold cross validation on the **training dataset** and by using the testing datasets computed from domains collected on **different days**. Both methods gave us very similar results. Our system performed the worst in the case of the 10-fold cross validation, therefore we chose to present this worst-case scenario.

In Table 1, we can see the detection results using two values for α, five and ten. We omit the results for the other values due to space limitations. The main confusion between the classes was observed in the datasets that contained separate Conficker classes, specifically between the classes of Conficker-A and Conficker-B. To address this problem, we created a generic Conficker class that had an equal number of vectors from each Conficker variant. This merging of the Conficker variants into a single "super" class allowed the DGA classifier to correctly classify 99.72% (Table 1) of the instances (7,986 correctly classified vs 22 incorrectly classified). Using the datasets with the five classes of DGAs, the weighted average of the TP_{rates} and FP_{rates} were 99.7% and 0.1%, respectively. As we see in Table 1, $\alpha = 5$ performs reasonably well, but with a higher rate of FPs.

7.2 NXDomain Clustering Results

In this section, we will discuss results from the DGA discovery module. In particular, we elaborate on the selection of the thresholds used, the unique clusters identified and the false alerts the DGA discovery module produced over the duration of our study.

7.2.1 Correlation Thresholds

In order to set the thresholds θ_{maj} and θ_{σ} defined in Section 4.2, we spent the first five days of November 2010 labeling the 213 produced clusters as DGA related (Positive) or noisy (Negative). For this experiment, we included all produced clusters without filtering out those with θ_{μ}=98% (or higher) "similarity" to an already known one (see Section 4.2). In Figure 3, we can see in the Y-axis the percentage values for the dominant (non-benign) class in every cluster produced during these five days. In the X-axis we can see the variance that each dominant class had within each cluster. The results show that the Positive and Negative assignments had a clear cut, which we can achieve by setting the thresholds as $\theta_{maj} = 75\%$ and $\theta_{\sigma} = 0.001$. These thresholds gave us very good results throughout the duration of the experiments. As we will discuss in Section 7.2.3, the DGA discovery module falsely reported only five benign clusters over a period of 15 months. All falsely reported clusters had variance very close to 0.001.

7.2.2 New DGAs

Pleiades began clustering NXDomain traffic on the first day of November 2010. We bootstrapped the DGA modeler with domain names from already known DGAs and also a set of Alexa domain names as the benign class. In Table 2, we present all unique clusters we discovered throughout the evaluation period. The "Malware Family" column simply maps the variant to a known malware family if possible. We discover the malware family by checking the NXDomains that overlap with NXDomains we extracted from traffic obtained from a malware repository. Also, we manually inspected the clusters with the help of a security company's threat team. The "First

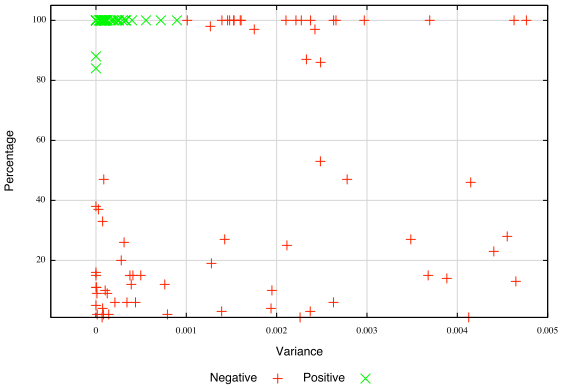

Figure 3: Thresholds θ_{maj} and θ_{σ} from the first five days of November 2010.

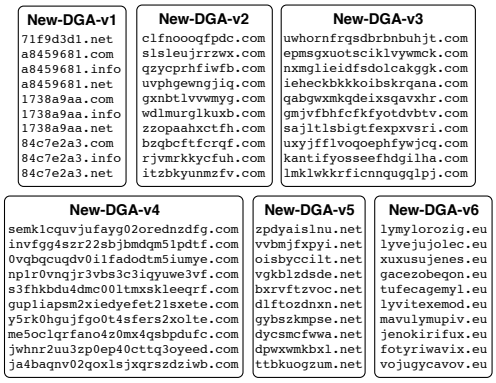

Figure 4: A sample of ten NXDomain for each DGA cluster that we could not associate with a known malware family.

Seen" column denotes the first time we saw traffic from each DGA variant. Finally, the "Population on Discovery" column shows the variant population on the discovery day. We can see that we can detect each DGA variant with an average number of 32 "infected hosts" across the entire statewide ISP network coverage.

Table 2: DGAs Detected by Pleiades.

Malware Family	First Seen	Population on Discovery
Shiz/Simda-C [32]	03/20/11	37
Bamital [11]	04/01/11	175
BankPatch [5]	04/01/11	28
Expiro.Z [8]	04/30/11	7
Boonana [41]	08/03/11	24
Zeus.v3 [25]	09/15/11	39
New-DGA-v1	01/11/10	12
New-DGA-v2	01/18/11	10
New-DGA-v3	02/01/11	18
New-DGA-v4	03/05/11	22
New-DGA-v5	04/21/11	5
New-DGA-v6	11/20/11	10

As we see in Table 2, Pleiades reported six variants that belong to known DGA-enabled malware families [5,8,11,25,32,41]. Six more variants of NXDomains were reported and modeled by Pleiades but for these, to the best of our knowledge, no known malware can be associated with them. A sample set of 10 domain names for each one of these variants can be seen in Figure 4.

In the 15 months of our observations we observed an average population of 742 Conficker infected hosts in the ISP network. Murofet had the second largest population of infected hosts at 92 per day, while the Boonana DGA comes third with an average population of 84 infected hosts per day. The fastest growing DGA is Zeus.v3 with an average population of 50 hosts per day, however, during the last four days of the experiments the Zeus.v3 DGA had an average number of 134 infected hosts. It is worth noting the New-DGA-v1 had an average of 19 hosts per day, the most populous of the newly identified DGAs.

7.2.3 False Reports on New DGAs

During our evaluation period we came across five categories of clusters falsely reported as new DGAs. In all of the cases, we modeled these classes in the DGA modeler as variants of the benign class. We now discuss each case in detail.

The first cluster of NXDomains falsely reported by Pleiades were random domain names generated by Chrome [16,45]. Each time the Google Chrome browser starts, it will query three "random looking" domain names. These domain names are issued as a DNS check, so the browser can determine if NXDomain rewriting is enabled. The "Chrome DGA" was reported as a variant of Bobax from Pleiades. We trained a class for this DGA and flagged it as benign. One more case of testing for NXDomain rewriting was identified in a brand of wireless access points. Connectify[3], offers wireless hotspot functionality and one of their configuration option enables the user to hijack the ISP's default NXDomain rewriting service. The device generates a fixed number of NXDomains to test for rewriting.

Two additional cases of false reports were triggered by domain names from the `.it` and `.edu` TLDs. These domain names contained minor variations on common words (i.e. repubblica, gazzetta, computer, etc.). Domain names that matched these clusters appeared only for two days in our traces and never again. The very short lived presence of these two clusters could be explained if the domain names were part of a spam-campaign that was remediated by authorities before it became live.

The fifth case of false report originated from domain names under a US government zone and contained the

[3] `www.connectify.me`

Table 3: TPs (%) for C&C detection (1,000 training sequences).

	FPs (%)					
botnet	*0.1*	*0.5*	*1*	*3*	*5*	*10*
Zeus.v3	99.9	99.9	99.9	99.9	99.9	99.9
Expiro.Z	33.03	64.56	78.23	91.77	95.23	98.67
Bamital	100	100	100	100	100	100
Shiz	0	1.64	21.02	96.58	100	100
Boonana	3.8	10.69	15.59	27.67	35.05	48.43
BankPatch	56.21	70.77	93.18	99.9	99.91	99.94

string `wpdhsmp`. Our best guess is that these are internal domain names that were accidentally leaked to the recursive DNS server of our ISP. Domain names from this cluster appeared only for one day. This class of NXDomains was also modeled as a benign variant. It is worth noting that all falsely reported DGA clusters, excluding the Chrome cluster, were short lived. If operators are willing to wait a few days until a new DGA cluster is reported by Pleiades, these false alarms would not have been raised.

7.3 C&C Detection

To evaluate the effectiveness of the *C&C Detection*, we proceeded as follows. We considered the six new DGAs which we were able to attribute to specific malware, as shown in Table 3. Let NX_i be the set of NXDomains collected by the *DGA Discovery* (Section 4) and *DGA Modeling* (Section 5.1) modules for the *i*-th DGA. For each DGA, we set aside a subset $NX_i^{train} \subset NX_i$ of NXDomains to train an HMM_i model. Then we use the remaining $NX_i^{test} = NX_i - NX_i^{train}$ to compute the true positive (TP) rate of HMM_i, and a set A that consists of 602,969 unique domain names related to the consistently popular domain names according to `alexa.com` to compute the false positive (FP) rate. To obtain A we first consider all domain names that have been consistently ranked in the top 100,000 popular domains by `alexa.com` for approximately one year. This gave us a set T of about 60,000 "stable" popular domain names, which we consider as *legitimate* domains. Then, we monitored the stream of successful DNS queries in a large live network for a few hours, and we added to A all the domain names whose effective 2LD is in T.

We performed experiments with a varying number $c = |NX_i^{train}|$ of training samples. Specifically, we set c equal to 100, 200, 500, 1,000, 2,000, 5,000, and 10,000. We then computed the trade-off between TPs and FPs for different detection thresholds. In the interest of space, we report only the results for c=1,000 in Table 3. In general, the results improve for increasing numbers of training instances. We set the detection threshold so as to obtain an FP rate equal to 0.1%, 0.5%, 1%, 3%, 5%, and 10%. As we can see, at FP=1% we obtained a high ($> 93\%$) TP rate for three out of six DGAs, and relatively good results ($> 78\%$) in five out of six cases. At FP=3% we have high TP rate ($> 91\%$) in five out of six cases.

As mentioned in Section 3, the *C&C Detection* module reduces the set of domain names successfully resolved by a host h that have been labeled as compromised with DGA-malware to a smaller set of good *candidate C&C domains* generated by the DGA. The results in Table 3 show that if we rank the domains resolved by h according to the likelihood assigned by the HMM, in most cases we will only need to inspect between 1/100 to 3/100 of the active domains queried by h to discover the C&C.

7.4 Case Studies

7.4.1 Zeus.v3

In September 2011, Pleiades detected a new DGA that we linked to the Zeus.v3 variant a few weeks later. The domain names collected from the machines compromised by this DGA-malware are hosted in six different TLDs: `.biz`,`.com`,`.info`,`.net`,`.org` and `.ru`. Excluding the top level domains, the length of the domain names generated by this DGA are between 33 and 45 alphanumeric characters. By analyzing one sample of the malware[4] we observed that its primary C&C infrastructure is P2P-based. If the malware fails to reach its P2P C&C network, it follows a contingency plan, where a DGA-based component is used to try to recover from the loss of C&C communication. The malware will then resolve pseudo-random domain names, until an active C&C domain name is found.

To date, we have discovered 12 such C&C domains. Over time, these 12 domains resolved to five different C&C IPs hosted in four different networks, three in the US (AS6245, AS16626 and AS3595) and one in the United Kingdom (AS24931). Interestingly, we observed that the UK-based C&C IP address remained active for a very short period of time of only a few minutes, from Jan 25, 2012 `12:14:04` EST to Jan 25, 2012 `12:22:37` EST. The C&C moved from a US IP (AS16626) to the UK (AS24931), and then almost immediately back to the US (AS3595).

7.4.2 BankPatch

We picked the BankPatch DGA cluster as a sample case for analysis since this botnet had been active for several months during our experiments and the infected population continues to be significant. The C&C infrastructure that supports this botnet is impressive. Twenty six different clusters of servers acted as the C&Cs for this botnet. The botnet operators not only made use of a DGA but also moved the active C&Cs to different networks every few weeks (on average). During our C&C

[4]Sample MD5s: 8f60afa9ea1e761edd49dfe012c22cbf and ccec69613c71d66f98abe9cc7e2e20ef.

discovery process, we observed IP addresses controlled by a European CERT. This CERT has been taking over domain names from this botnet for several months. We managed to cross-validate with them the completeness and correctness of the C&C infrastructure. Complete information about the C&C infrastructure can be found in Table 4.

The actual structure of the domain name used by this DGA can be separated into a four byte prefix and a suffix string argument. The suffix string arguments we observed were: `seapollo.com`, `tomvader.com`, `aulmala.com`, `apontis.com`, `fnomosk.com`, `erhogeld.com`, `erobots.com`, `ndsontex.com`, `rtehedel.com`, `nconnect.com`, `edsafe.com`, `berhogeld.com`, `musallied.com`, `newnacion.com`, `susaname.com`, `tvolveras.com` and `dminmont.com`.

The four bytes of entropy for the DGA were provided by the prefix. We observe collisions between NXDomains from different days, especially when only one suffix argument was active. Therefore, we registered a small sample of ten domain names at the beginning of 2012 in an effort to obtain a glimpse of the overall distribution of this botnet. Over a period of one month of monitoring the sink-holed data from the domain name of this DGA, this botnet has infected hosts in 270 different networks distributed across 25 different countries. By observing the recursive DNS servers from the domain names we sinkholed, we determined 4,295 were located in the US. The recursives we monitored were part of this list and we were able to measure 86 infected hosts (on average) in the network we were monitoring. The five countries that had the most DNS resolution requests for the sinkholed domain names (besides the US) were Japan, Canada, the United Kingdom and Singapore. The average number of recursive DNS servers from these countries that contacted our authorities was 22 — significantly smaller than the volume of recursive DNS servers within the US.

Table 4: C&C Infrastructure for BankPatch.

IP addresses	CC	Owner
146.185.250.{89-92}	RU	Petersburg Int.
31.11.43.{25-26}	RO	SC EQUILIBRIUM
31.11.43.{191-194}	RO	SC EQUILIBRIUM
46.16.240.{11-15}	UA	iNet Colocation
62.122.73.{11-14,18}	UA	“Leksim” Ltd.
87.229.126.{11-16}	HU	Webenlet Kft.
94.63.240.{11-14}	RO	Com Frecatei
94.199.51.{25-18}	HU	NET23-AS 23VNET
94.61.247.{188-193}	RO	Vatra Luminoasa
88.80.13.{111-116}	SE	PRQ-AS PeRiQuito
109.163.226.{3-5}	RO	VOXILITY-AS
94.63.149.{105-106}	RO	SC CORAL IT
94.63.149.{171-175}	RO	SC CORAL IT
176.53.17.{211-212}	TR	Radore Hosting
176.53.17.{51-56}	TR	Radore Hosting
31.210.125.{5-8}	TR	Radore Hosting
31.131.4.{117-123}	UA	LEVEL7-AS IM
91.228.111.{26-29}	UA	LEVEL7-AS IM
94.177.51.{24-25}	UA	LEVEL7-AS IM
95.64.55.{15-16}	RO	NETSERV-AS
95.64.61.{51-54}	RO	NETSERV-AS
194.11.16.133	RU	PIN-AS Petersburg
46.161.10.{34-37}	RU	PIN-AS Petersburg
46.161.29.102	RU	PIN-AS Petersburg
95.215.{0-1}.29	RU	PIN-AS Petersburg
95.215.0.{91-94}	RU	PIN-AS Petersburg
124.109.3.{3-6}	TH	SERVENET-AS-TH-AP
213.163.91.{43-46}	NL	INTERACTIVE3D-AS
200.63.41.{25-28}	PA	Panamaserver.com

8 Discussion and Limitations

Pleiades has some limitations. For example, once a new DGA is discovered, Pleiades can build fairly accurate statistical models of how the domains generated by the DGA “look like”, but it is unable to learn or reconstruct the exact domain generation algorithm. Therefore, Pleiades will generate a certain number of false positives and false negatives. However, the results we presented in Table 1 show that Pleiades is able to construct a very accurate *DGA Classifier* module, which produces very few false positives and false negatives for $\alpha = 10$. At the same time, Table 3 shows that the *C&C Detection* module, which attributes a single active domain name to a given DGA, and also works fairly well in the majority of cases. Unfortunately, there are some scenarios in which the HMM-based classification has difficulties. We believe this is because our HMM considers domain names simply to be sequences of individual characters. In our future work, we plan to experiment with 2-grams, whereby a domain name will be seen as a sequence of pairs of characters, which may achieve better classification accuracy for the harder to model DGAs.

For example, our HMM-based detector was unable to obtain high true positive rates on the Boonana DGA. The reason is that the Boonana DGA leverages third-level pseudo-random domain names under several second-level domains owned by *dynamic DNS* providers. During our evaluation, the hosts infected with Boonana contacted DGA-generated domain names under 59 different effective second-level domains. We believe that the high variability in the third-level domains and the high number of effective 2LDs used by the DGA make it harder to build a good HMM, thus causing a relatively low number of true positives. However, in a real-world deployment scenario, the true positive rate may be significantly increased by focusing on the dynamic DNS domains queried by the compromised hosts. For example, since we know that Boonana only uses dynamic DNS domains, we can filter out any other NXDomains, and avoid passing them to the HMM. In this scenario the

HMM would receive as an input only dynamic DNS domains, which typically represent a fraction of all active domains queried by each host, and consequently the absolute number of false positives can be significantly reduced.

As we mentioned in Section 3, detecting active DGA-generated C&C domains is valuable because their resolved IP addresses can be used to update a C&C IP blacklist. In turn, this IP blacklist can be used to block C&C communications at the network edge, thus providing a way to mitigate the botnet's malicious activities. Clearly, for this strategy to be successful, the frequency with which the C&C IP addresses change should be lower than the rate with which new pseudo-random C&C domain names are generated by the DGA. This assumption holds for all practical cases of DGA-based malware we encountered. After all, the generation of pseudo-random domains mainly serves the purpose of making the take-down of loosely centralized botnets harder. However, one could imagine "hybrid" botnets that use DGA-generated domains to identify a set of peer IPs to bootstrap into a P2P-based C&C infrastructure. Alternatively, the DGA-generated C&C domains may be flux domains, namely domain names that point to a IP fluxing network. It is worth noting that such sophisticated "hybrid" botnets may be quite complex to develop, difficult to deploy, and hard to manage successfully.

Another potential limitation is due to the fact that Pleiades is not able to distinguish between different botnets whose bot-malware use the same DGA algorithm. In this case, while the two botnets may be controlled by different entities, Pleiades will attribute the compromised hosts within the monitored network to a single DGA-based botnet.

One limitation of our evaluation method is the exact enumeration of the number of infected hosts in the ISP network. Due to the location of our traffic monitoring sensors (below the recursive DNS server), we can only obtain a lower bound estimate on the number of infected hosts. This is because we have visibility of the IP addresses within the ISP that generate the DNS traffic, but lack additional information about the true number of hosts "behind" each IP. For example, an IP address that generates DNS traffic may very well be a NAT, firewall, DNS server or other type of complex device that behaves as a proxy (or relay point) for other devices. Also, according to the ISP, the DHCP churn rate is relatively low, and it is therefore unlikely that we counted the same internal host multiple times.

In the case of Zeus.v3, the DGA is used as a backup C&C discovery mechanism, in the event that the P2P component fails to establish a communication channel with the C&C. The notion of having a DGA component as a redundant C&C discovery strategy could be used in the future by other malware. A large number of new DGAs may potentially have a negative impact on the supervised modules of Pleiades, and especially on the HMM-based C&C detection. In fact, a misclassification by the *DGA Classifier* due to the large number of classes among which we need to distinguish may misguide the selection of the right HMM to be used for C&C detection, thus causing an increase in false positives. In our future work we plan to estimate the impact of such misclassifications on the C&C detection accuracy, and investigate whether using auxiliary IP-based information (e.g., IP reputation) can significantly improve the accuracy in this scenario.

As the internals of our system become public, some botnets may attempt to evade both the DGA discovery and C&C detection process. As we have already discussed, it is in the malware authors' best interest to create a high number of DGA-related NXDomains in order to make botnet take-over efforts harder. However, the malware could at the same time generate NXDomains not related with the C&C discovery mechanism in an effort to mislead our current implementation of Pleiades. These *noisy* NXDomains may be generated in two ways: (1) randomly, for example by employing a different DGA, or (2) by using one DGA with two different seeds, one of which is selected to generate noise. In case of (1), the probability that they will be clustered together is small. This means that these NXDomains will likely not be part of the final cluster correlation process and they will not be reported as new DGA-clusters. On the other hand, case (2) might cause problems during learning, especially to the HMM, because the noisy and "true" NXDomains may be intermixed in the same cluster, thus making it harder to learn an accurate model for the domain names.

9 Conclusion

In this paper, we presented a novel detection system, called Pleiades, that is able to accurately detect machines within a monitored network that are compromised with DGA-based bots. Pleiades monitors traffic below the local recursive DNS server and analyzes streams of unsuccessful DNS resolutions, instead of relying on manual reverse engineering of bot malware and their DGA algorithms. Using a multi-month evaluation phase, we showed that Pleiades can achieve very high detection accuracy. Moreover, over the fifteen months of the operational deployment in a major ISP, Pleiades was able to identify six DGAs that belong to known malware families and six new DGAs never reported before.

References

[1] K. Aas and L. Eikvil. Text categorisation: A survey., 1999.

[2] abuse.ch. ZeuS Gets More Sophisticated Using P2P Techniques. http://www.abuse.ch/?p=3499, 2011.

[3] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for DNS. In *the Proceedings of 19th USENIX Security Symposium (USENIX Security '10)*, 2010.

[4] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting malware domains in the upper DNS hierarchy. In *the Proceedings of 20th USENIX Security Symposium (USENIX Security '11)*, 2011.

[5] BankPatch. Trojan.Bankpatch.C. http://www.symantec.com/security_response/writeup.jsp?docid=2008-081817-1808-99&tabid=2, 2009.

[6] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive dns analysis. In *Proceedings of NDSS*, 2011.

[7] R. Feldman and J. Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge Univ Pr, 2007.

[8] R. Finones. Virus:Win32/Expiro.Z. http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Virus%3AWin32%2FExpiro.Z, 2011.

[9] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, 1999.

[10] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proc. USENIX Security*, 2007.

[11] M. Geide. Another trojan bamital pattern. http://research.zscaler.com/2011/05/another-trojan-bamital-pattern.html, 2011.

[12] S. Golovanov and I. Soumenkov. TDL4 top bot. http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot, 2011.

[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *USENIX Security*, 2008.

[14] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[15] J. Hermans. MozillaWiki TLD List. https://wiki.mozilla.org/TLD_List, 2006.

[16] S. Krishnan and F. Monrose. Dns prefetching and its privacy implications: when good things go bad. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[17] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[18] M. H. Ligh, S. Adair, B. Hartstein, and M. Richard. *Malware Analyst's Cookbook and DVD*, chapter 12. Wiley, 2010.

[19] P. Mockapetris. Domain names - concepts and facilities. http://www.ietf.org/rfc/rfc1034.txt, 1987.

[20] P. Mockapetris. Domain names - implementation and specification. http://www.ietf.org/rfc/rfc1035.txt, 1987.

[21] M. Newman. *Networks: an introduction*. Oxford University Press, 2010.

[22] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances In Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[23] P. Porras, H. Saidi, and V. Yegneswaran. An analysis of conficker's logic and rendezvous points. http://mtc.sri.com/Conficker/, 2009.

[24] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[25] C. POLSKA. ZeuS P2P+DGA variant mapping out and understanding the threat.

http://www.cert.pl/news/4711/langswitch_lang/en, 2012.

[26] P. Porras. Inside risks: Reflections on conficker. *Communications of the ACM*, 52:23–24, October 2009.

[27] P. Porras, H. Saidi, and V. Yegneswaran. Conficker C analysis. Technical report, SRI International, Menlo Park, CA, April 2009.

[28] L. R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition. 1990.

[29] P. Royal. Analysis of the kraken botnet. http://www.damballa.com/downloads/r_pubs/KrakenWhitepaper.pdf, 2008.

[30] S. Shevchenko. Srizbi domain generator calculator. http://blog.threatexpert.com/2008/11/srizbis-domain-calculator.html, 2008.

[31] S. Shevchenko. Domain name generator for murofet. http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html, 2010.

[32] SOPHOS. Mal/Simda-C. http://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Mal~Simda-C/detailed-analysis.aspx, 2012.

[33] J. Stewart. Bobax trojan analysis. http://www.secureworks.com/research/threats/bobax/, 2004.

[34] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[35] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the storm and nugache trojans: P2P is here. In *USENIX ;login:, vol. 32, no. 6*, December 2007.

[36] T.-F. Yen and M. K. Reiter. Are your hosts trading or plotting? Telling P2P file-sharing and bots apart. In *ICDCS*, 2010.

[37] R. Villamarin-Salomon and J. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *5th Consumer Communications and Networking Conference*, 2008.

[38] Wikipedia. The storm botnet. http://en.wikipedia.org/wiki/Storm_botnet, 2010.

[39] J. Williams. What we know (and learned) from the waledac takedown. http://tinyurl.com/7apnn9b, 2010.

[40] J. Wolf. Technical details of srizbi's domain generation algorithm. http://blog.fireeye.com/research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html, 2008. Retreived: April, 10 2010.

[41] J. Wong. Trojan:Java/Boonana. http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Trojan%3AJava%2FBoonana, 2011.

[42] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual Conference on Internet Measurement*, IMC '10, pages 48–61, New York, NY, USA, 2010. ACM.

[43] S. Yadav and A. N. Reddy. Winning with dns failures: Strategies for faster botnet detection. In *7th International ICST Conference on Security and Privacy in Communication Networks*, 2011.

[44] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proc. International conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[45] B. Zdrnja. Google Chrome and (weird) DNS requests. http://isc.sans.edu/diary/Google+Chrome+and+weird+DNS+requests/10312, 2011.

[46] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo. Detecting stealthy P2P botnets using statistical traffic fingerprints. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Dependable Computing and Communication Symposium*, 2011.