Received: 13 March 2018 | Revised: 26 June 2018 | Accepted: 20 July 2018

DOI: 10.1002/ett.3505

SPECIAL ISSUE ARTICLE

WILEY

Thriving on chaos: Proactive detection of command and control domains in internet of things-scale botnets using DRIFT

Jeffrey Spaulding[1] | Jeman Park[1] | Joongheon Kim[2] | DaeHun Nyang[3] | Aziz Mohaisen[1]

[1]Computer Science Department, University of Central Florida, Orlando, Florida

[2]School of Software, Chung-Ang University, Seoul, South Korea

[3]Computer Science and Information Engineering, Inha University, Incheon, South Korea

Correspondence
Joongheon Kim, 84 Heukseok-ro, Dongjak-gu, Seoul 06974, Republic of Korea.
Email: joongheon@cau.ac.kr

Aziz Mohaisen, Computer Science Department, University of Central Florida, 4000 Central Florida Blvd, Orlando, FL 32816.
Email: mohaisen@ucf.edu

Funding information
National Science Foundation, Grant/Award Number: CNS-1809000; Korea National Research Foundation, Grant/Award Number: 2016K1A1A2912757; Chung-Ang University Research Grant (2018)

Abstract

In this paper, we introduce DRIFT, a system for detecting command and control (C2) domain names in Internet of Things–scale botnets. Using an intrinsic feature of malicious domain name queries prior to their registration (perhaps due to clock drift), we devise a difference-based lightweight feature for malicious C2 domain name detection. Using NXDomain query and response of a popular malware, we establish the effectiveness of our detector with 99% accuracy and as early as more than 48 hours before they are registered. Our technique serves as a tool of detection where other techniques relying on entropy or domain generating algorithms reversing are impractical.

1 | INTRODUCTION

With the number of connected Internet-of-Things (IoT) devices expecting to surpass 20.4 billion by 2020 (*not including computers and phones*),[1] the main threat looming on the horizon for these IoT devices is the scourge of botnets. According to the OWASP IoT project,[2] IoT systems are at a high security risk due to vulnerabilities such as the lack of adopting well-known security techniques like encryption, authentication, and role-based access control. The Mirai botnet, for example, infected thousands of IoT devices and wreaked havoc across the Internet by launching some of the largest and most disruptive distributed denial-of-service (DDoS) attacks against sites like Krebs on Security, OVH,[3] and the domain name system (DNS) provider Dyn.[4]

The typical botnet consists of various infected hosts, command and control (C2) channels, and a botmaster. The infected hosts ("zombies"), as shown in Figure 1, are often massively distributed, whereas the C2 is a channel used by a mastermind (the "botmaster") to instruct bots to perform various forms of malice, eg, launching DDoS attacks.[5] To communicate with

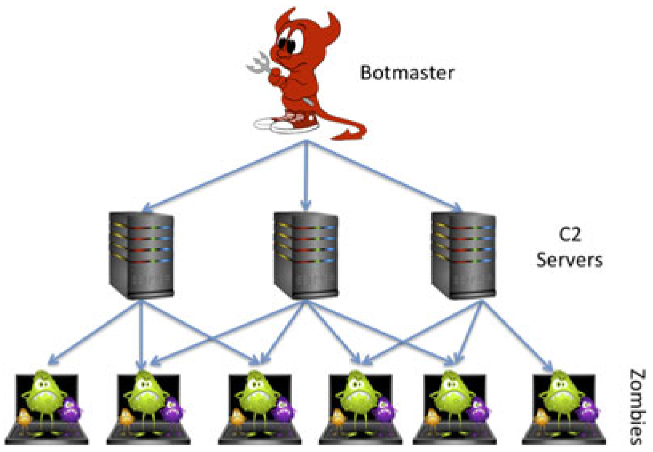

FIGURE 1 An illustration of a botnet infrastructure

bots, there are several potential ways utilized by botmasters, and domain names as a C2 channel are one of the most common and preferred methods because they are easy to acquire and recycle.[6] To generate such domain names, domain generation algorithms (DGAs) are widely used today by botmasters. Usually, DGAs use time as a seed to dynamically and automatically generate potentially pseudorandom domain names that are registered by botmasters and used by bots for C2 communication. Variants of the Mirai botnet, for instance, started to adopt DGAs to better avoid detection and keep a constant contact with C2 servers.[7] One way of mitigating these botnets is to prevent them from registering their C2 domains in the first place or by taking such domain names down.[8,9]

To address DGAs by detection, there have been two schools of thought either (1) relying on reverse engineering of the bot software[10] or (2) using the intrinsic features of the generated domains.[11] The first method, while powerful in generating all domain names to be potentially used by a malware family (even in the future and can thus be used to proactively block those domains by preregistering them), is very expensive. This method requires obtaining samples of the malware family that utilizes such DGAs. However, obtaining such malware is not the biggest hurdle. Many of today's malware families employ obfuscation techniques that make their analysis a difficult task.

The second school of thought uses pseudorandomness of algorithmically-generated domains and exploits the fact that those domains have a high entropy for their detection.[12] Registered domain names that are queried by infected hosts are evaluated, a measure of their pseudorandomness using their entropy is calculated, and the likelihood of them being malicious based on their entropy score is assigned. While shown to identify malicious domains reasonably well, such techniques suffer from various drawbacks. First, domain names need to be registered for a monitor to be able to measure such entropy and determine if a domain is malicious or not. Thus, such techniques cannot be used proactively to detect malicious domains. Second, those techniques assume that randomly generated domain names are only used in malicious activities. It is not far fetched to imagine that domain names with high entropy are utilized for domain name parking and nonpublic facing domains (eg, content delivery network addressing) among others.

1.1 | Key idea

To this end, this paper addresses the problem of proactive malicious domain name identification focusing on DGAs. We aim to identify such domain names before they are registered using our detection system known as DRIFT. Our main source of inference is the DNS query and resolution of domain names. We motivated our study by a large-scale analysis of domain names and their queries. We find that domains that are used for malicious activities, and especially those generated algorithmically, tend to have unique and distinguishing patterns. In particular, domain names that are algorithmically generated tend to have a large number of DNS queries even before their registration, typically resulting in NXDomain (nonexistent domain) responses. This trend persists, and the number of queries increases and peaks at the time of registration, then declines gradually, indicating the ephemeral use of those domain names for their major purpose. On the other hand, domain names that are being used for benign applications tend to have significantly less queries before registration, whereas their post-registration query volumes (which may fluctuate over time) do not have a single declining curve, thus highlighting a fundamentally different use model.

1.2 | Contributions

The contribution of this paper is as follows. First, we highlight a fundamental difference between the query patterns for domain names that are used by botnets, often generated using DGAs, and those by benign ones. We use this insight to differentiate between those domains using a simple classification algorithm for proactive detection of malicious domains.

1.3 | Organization

The organization of the rest of this paper is as follows. Section 2 provides an overview of the DNS resolution and registration process, which is then followed by Section 3 that describes the threat model and system overview of DRIFT. Section 4 presents our online detector and Section 5 reviews our datasets and high-level characteristics. In Section 6, we evaluate our system and provide a discussion in Section 7. Section 8 highlights the related work of detecting DGA domains and we provide concluding remarks in Section 9.

2 | PRELIMINARIES

2.1 | DNS overview

In this section, we provide an overview of how the DNS functions and the domain name resolution process once a user initiates a DNS query. In Section 2.1.3, we discuss the process of how a domain name is registered in the DNS.

2.1.1 | Functionality

Domain names are integral to the DNS, which allows us to map human-readable strings to machine-readable IP addresses. In the case of IPv4, the address on the network is composed of 4 bytes (32 bits) in total and can be represented by four number segments separated by dots as in `1.2.3.4`. IPv6, which has an address space that is four times larger than IPv4, has a total of eight segments in its address structure, with each segment being 2 bytes that are represented by hexadecimal numbers (eg, 2a03:2880:f10c:83:face:b00c:0:25de). It is a herculean task for users to remember all the numerical addresses of the Internet services they want to access. The DNS, which is designed to improve the usability, allows users to access them through the familiar natural language, such as `www.example.com`.

2.1.2 | DNS resolution

The DNS is a hierarchical and distributed database structure for resolving domain names. The string address entered by users is converted to an IP address through root, top-level domain (TLD), and authoritative name servers. Figure 2 shows the translation of the domain name. When users attempt to access a particular service through an Internet web browser such as Chrome or Firefox, the DNS resolution process begins.

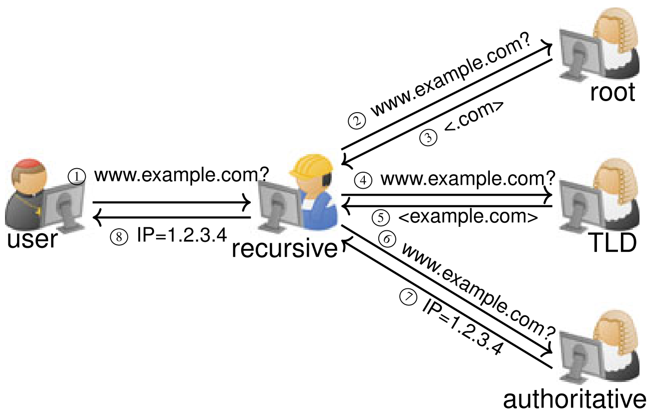

FIGURE 2 The steps of DNS resolution through the recursive, root, top-level domain, and authoritative name servers

Local cache and host table. As the first step of DNS resolution, the local resolver initially finds its local cache and host table. If there is no corresponding entry for the given domain, the local resolver generates and sends the DNS query to the recursive server for the DNS resolution. Step ① in Figure 2 shows the initial DNS query from the client to the recursive server. The query includes a special flag to indicate that is a recursive query. Once the recursive server receives the query from the client, it begins the recursive steps ② through ⑦ with the root, TLD, and authoritative name servers.

Root name server. In step ②, the recursive name server (which does not have a corresponding entry for the received query from the client) sends an iterative query to the root name server. The root name server is responsible for the translation of the root zone in the DNS system using stored data, such as the IP address and location of an authoritative TLD name server. Once the root name server receives the query of `www.example.com` in step ③, it returns the appropriate list of authoritative TLD servers for the `.com` TLD.

TLD name server. The next phase, ie, step ④, is querying for the translation of the TLD in the given domain. The resolver that found the TLD name server list through the previous step sends a query for `example.com` to one of the `.com` TLD name servers. The `.com` TLD name server searches for the record of the authoritative name server, corresponding to the queried domain and responds to the recursive server with a list, as shown in step ⑤. For example, if `example.com` has two authoritative name servers, namely, ns1 and ns2, the information about `ns1.example.com` and `ns2.example.com` would be included in the response to the recursive server.

Authoritative name server. The final step of recursive domain name resolution process goes through the authoritative name server. In step ⑥, the recursive name server contacts one of the authoritative name servers in the records received from the `.com` TLD name server. The authoritative name server, which knows the A address record for `www.example.com` (or AAAA address record in the case of IPv6), sends the result to the recursive server in step ⑦. At this point, the recursive name server determines the IP address of the requested domain name and forwards it to the local machine shown in ⑧. The local resolver, which finally knows the IP address of `www.example.com`, delegates it to a web browser. As a result, the web browser will be able to initiate the loading of a web page by sending a Hypertext Transfer Protocol (HTTP) request to the resolved IP address.

2.1.3 | Domain name registration process

In late 1998, the United States Department of Commerce named a private and newly-formed nonprofit organization called the Internet Corporation for Assigned Names and Numbers (ICANN) as the new entity to oversee the assignment of both IP addresses and domain names.[13] As outlined by ICANN, the process works as follows.[14] A domain name registrant (a person or organization) will usually apply online to an ICANN-accredited domain registrar (eg, *GoDaddy*) or one of their resellers. The registrar will check if the domain name is available and create a WHOIS record with the registrant's information. While registrars are contracted to conduct the day-to-day business of selling domain name registrations, registries (eg, *Verisign*) are responsible for maintaining the registry for each TLD. These registries are also responsible for accepting registration requests and maintaining a database of the necessary domain name registration data. Furthermore, they have to provide name servers to publish the zone file data (ie, information about the location of a domain name) throughout the Internet.

3 | SYSTEM OVERVIEW

In this section, we describe the objective and the approach taken by adversaries that DRIFT would like to address as well as an overview of the system.

3.1 | Threat model

With botnets quickly becoming the one of the most prevalent threats on the Internet,[15-17] the key threat we are most concerned with is ultimately the botmaster who commands a herd of bots. To communicate with their bots, botmasters typically use domain names as its C2 channel because they are easy to acquire and recycle.[6] Since using a single domain name for C2 communication can constitute a single point of failure (ie, law enforcement take downs), botnet authors (eg, Conficker, Torpiq, Kraken, etc.) began adopting algorithmically-generated domain names. These algorithms that generate domain names typically use a pseudorandom number generator (PRNG) that is seeded with a time value. As this algorithm is shared among bots participating in the botnet, using a synchronized seed value such as the current time

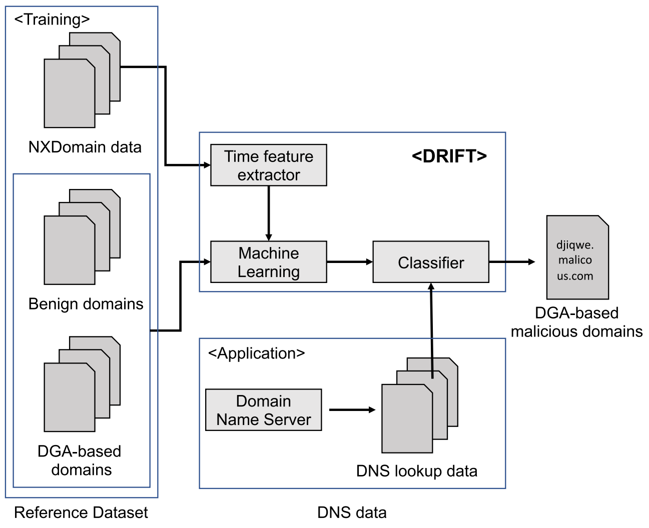

FIGURE 3 A system diagram for the detection of DGA-based malicous domains. DGA, domain generation algorithm; DNS, domain name system

will allow each bot to generate a similar list of DGA domain names to query in a sequential fashion. In the meantime, a botmaster, knowing full knowledge of the potential DGA domain names for any given time, can easily register one of those domain names in the DNS. Given that these DGAs can produce hundreds of domain names a day (250 in the case of Conficker A and B), it is becomingly difficult to deduce and reverse engineer these algorithms due to high entropy of its output. To this end, DRIFT aims to sever the C2 communication link between a botmaster and its bot herd by proactively identifying DGA domain names before they are registered.

3.2 | System overview

The structure of DRIFT is shown in Figure 3, which is a malicious domain detector based on the supervised learning technique discussed in Section 4.3. The system consists of two stages, a training phase and an application phase. In the training phase, feature extraction and learning are performed using precollected data. Afterwards, the application phase classifies DGA-based domain names through the learned classifier.

Training phase. The data elaborated on earlier in Section 5 is used for training as ground-truth data. As mentioned previously, a NXDomain response is returned if there is a connection attempt from a bot to a DGA domain that is not yet registered in the DNS. The time features of the collected NXDomain responses are extracted and used for supervised learning along with the labels of the domain name (benign or malicious). By analyzing the change of the NXDomain volume over time, the DRIFT system learns about the unique pattern evident of DGA-based botnets.

Application phase. Through the training process, the system has a general understanding of the volume of changes in NXDomain responses caused by the clock drift between systems, as shown in Figure 6. Based on the learned result, it becomes possible to proactively distinguish the queries for the DGA domain names among all queries coming into the DNS in real time. In other words, at this point, the DRIFT system connected to the DNS is able to detect the suspicious domain names, which could potentially be used for C2 channel communication in botnets.

4 | ONLINE DETECTION ALGORITHM

Compared to the whole population of the NXDomain traffic, the volume and daily reoccurrence make DGA domains statistically abnormal. Significant traffic uptick for a given DGA domain occurs both one day prior and postgeneration date. Traffic volumes on the exact generation date soar several magnitudes higher than the ±5 days baseline to 42 887 unique 24 recursive name servers consisting of 199 097 total NXDomain requests from 211 unique countries for an average DGA domain.

```python
def generate_domain(y, m, d):
    domain = ""
    for i in range(16):
        y=((y^8*y)>>11)^((y&0xFFFFFFF0)<<17)
        m=((m^4*m)>>25)^16*(m&0xFFFFFFF8)
        d=((d^(d<<13))>>19)^((d&0xFFFFFFFE)<<12)
        dm+=chr(((y^m^d)%25)+97)
    return domain
```

FIGURE 4 Example domain generation algorithm used by CryptoLocker

4.1 | Rationale of feature

There are potentially various explanations for why domain names get queried before their registration. First, domain names intended for benign usage might get queried by interested registrants who want to acquire them, thus explaining the small number of queries, some of them receive before registration. On the other hand, the large number of queries that DGA domains receive might have to do with the fact that those domain names are time dependent. As highlighted in Figure 4, for the DGA domain used initially by CryptoLocker, a bot calculates a domain name using time features as the main inputs. A large drift in the time or clock misconfiguration would result in bots contacting the domain name before or after its registration.

We use such verified observation as the main feature for identifying malicious domain names. Using this a priori knowledge captured in a training model, we devise a method that can detect those domains even before they are registered with a high accuracy rate. The proposed approach based on this feature has various plausible benefits over the state-of-the-art ones. Among the others, the proposed technique is robust to the behavior of the adversary.

4.2 | Online detection

To be able to tell whether a domain name is malicious or not, we use the notion of the difference function. Given a function $y = f(x)$ that is defined on an interval $[x, x+h]$, the average rate of change of the function on the interval $[x, x+h]$ is as follows:

$$\frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}. \tag{1}$$

For an interval of $[x-a, x+a]$ (*a is some constant*), we have the following:

$$\frac{f(x+a) - f(x-a)}{(x+a) - (x-a)} = \frac{f(x+a) - f(x-a)}{2a}. \tag{2}$$

Using this simple concept, in the following, we outline how to build a feature vector, how to build a model, and how to label various domain names as malicious (used for C2) or benign.

4.2.1 | Building feature vectors

For a window of size $2a$ (a from the left and right of point x; note that x here can be any point in time), we calculate the difference as the change in the number of NXDomain responses to a given domain, normalized by the total time units, corresponding to $2a$. Parameter a is used based on the desired performance, and x is used for all values of the observed traffic. We highlight the operation of the basic feature with an example. Let us consider an observation of $[o_1^j, o_2^j, \dots, o_k^j]$ (for domain j), where each observation is the total number of NXDomain responses for a domain j over a fixed period of time (eg, hour). If we are to consider $a = 1$ in Equation (2), we calculate f_i^j as $|o_{i+1}^j - o_i^j|/2$, for all i (resulting in a vector of values, representing the use of the given domain, ie, $[f_i^j]^{1\times k}$). For a unified treatment of the vector, we normalize each element in it by the sum of all of its elements; this is $f_i^j / \sum_{\forall i} f_i^j$. Our detection algorithm then uses the same idea as earlier, over a sliding window of observations. As time goes, the window slides by forgetting the oldest observations of NXDomain responses for the given domain. Additionally, the detector updates the count vector of the NXDomain responses for the domain and calculates our feature vector as the difference function.

4.2.2 | Building a model

For a set of domains $d_1, \ldots, d_t$ that are known to be malicious, we create model $\mathcal{M}$ that is calculated as a centroid feature vector, corresponding to the average of the feature values of the different domains. As such, we define $\mathcal{M}$ as follows:

$$\mathcal{M} = [m_1, \ldots, m_k] : m_i = 1/t \sum_{j=1}^{t} f_i^j. \tag{3}$$

As earlier, for a unified treatment, we normalize each element in it by the sum of all of its elements; this is $m_i / \sum_{\forall i} m_i$.

4.2.3 | Learning labels of domains

The labeling of the domains is divided into two learning types, ie, 1-class, which is concerned with, given the associated label, if the distance between the feature vector and a reference is at most Δ; and 2-class learning, which is concerned of associating a domain name with the label that is closest to it (based on the comparison of the feature vector corresponding to each of them). In the following, we will describe both approaches formally.

1-class learning. For a candidate domain x defined by its difference function f^x as earlier, we determine the label of the domain by conducting the following. We calculate the Manhattan distance between $\mathcal{M}$ and f^x. That is, we calculate

$$D(\mathcal{M}, f^x) = \sum_{\forall i} \left| m_i - f_i^x \right|. \tag{4}$$

Then, we label the domain as malicious if $D(\mathcal{M}, f^x) > \Delta$ and as benign otherwise. Δ is determined through measurements and tuning based on the learning of the underlying distribution of the NXDomain queries and their difference functions of malicious domains.

2-class learning. Alternatively, we create a model for a set of known benign domains, namely, $\mathcal{B}$, where $\mathcal{B} = [b_1, \ldots, b_k]$, and assign the label of a sample x based on the following:

$$\text{Label} = \begin{cases} \text{Malicious} & : D(\mathcal{B}, f^x) > D(\mathcal{M}, f^x) \\ \text{Benign} & : D(\mathcal{B}, f^x) \le D(\mathcal{M}, f^x). \end{cases} \tag{5}$$

We note that our scheme is less aggressive because it prioritizes benign over malicious, as shown earlier. Depending on the detector objective, we might also be more aggressive by assigning a malicious label to a domains when the two quantities are equal; this is, alternatively,

$$\text{Label} = \begin{cases} \text{Malicious} & : D(\mathcal{B}, f^x) \ge D(\mathcal{M}, f^x) \\ \text{Benign} & : D(\mathcal{B}, f^x) < D(\mathcal{M}, f^x). \end{cases} \tag{6}$$

4.3 | Detection algorithm

The approach behind DRIFT in classifying domains into "malicious" or "benign" is based on the supervised learning technique of the nearest centroid neighborhood (NCN) classifier introduced by Chaudhuri in 1994.[18] The key idea behind this algorithm is that it assumes that a target class is represented by a cluster and uses its mean (ie, centroid) to determine the class of a new sample point based on its distance. Typically, the Euclidean distance calculations are used, but this can be any distance function. In DRIFT, we calculate the Manhattan distance between feature vectors, as shown in Equation 4. For a sample with an unknown class, the NCN classifier chooses a class with the closest centroid for the given sample. Despite this simple approach, Chaudhuri emphasizes that the NCN classifier can obtain high accuracy.[18] As shown in a study by Han and Karypis,[19] the centroid-based classifier consistently and substantially outperforms other algorithms, such as Naive Bayesian, k-nearest neighbors, and C4.5, on a wide range of datasets. The k-nearest neighbors algorithm, for example, requires computing and sorting every distance between the unknown sample and all others in the dataset, which is computationally expensive when the dataset is very large.

The first step in DRIFT is to build a model based on a set of domains that are known to be malicious, which is outlined in Section 4.2.2. Using a set of NXDomain observations for each malicious domain, the feature vectors are built using the function shown on Line 1 of Algorithm 1. The feature vectors are then fed into the function shown on Line 12, which produces a model representing a centroid feature vector. For the 1-class learning, we label the domain malicious if the return value of the function *distance*$(\mathcal{M}, f^x)$ is greater than the threshold value Δ and benign otherwise. The 2-class learning method is similar, but the threshold value Δ is substituted for a benign domain model.

In terms of computational complexity, the building of feature vectors is $O(k)$, where k is the number of observations for a given domain j. Building a model of centroid feature vectors, on the other hand, requires $O(kt)$, where t is the number of domains that the model represents. Note that building models of centroid feature vectors for the malicious and benign domains is only a preprocessing step prior to the actual learning phase, which runs in $O(k)$ time. Thus, the overall computational complexity of DRIFT is very low because the main learning phase is linear time.

Algorithm 1: DRIFT Algorithm

```
Input  : $O = [o^j_k]$ ($k$ observations for domain $j$), $a$ (window size)
Output: $F$ (vector of feature values $f^j_i$)
Function buildFeatureVector ($O, a$)
$sum = 0$
for $i = 1$ to $k$ do
    $f^j_i = |f^j_{i+a} - f^j_{i-a}|/2a$
    $sum = sum + f^j_i$
end
for $i = 1$ to $k$ do
    $f^j_i = f^j_i/sum$                                   /* (normalize) */
end
return $F$

Input  : $F$ (vector of $t$ feature values $f^j_i$)
Output: $\mathcal{M}$ (vector of $k$ centroids $m_i$)
Function buildModel ($F$)
$sum_m = 0$
for $i = 1$ to $k$ do
    $sum_f = 0$
    for $j = 1$ to $t$ do
        $sum_f = sum_f + f^j_i$
    end
    $m_i = (1/t) \times sum_f$
    $sum_m = sum_m + m_i$
end
for $i = 1$ to $k$ do
    $m_i = m_i/sum_m$                                     /* (normalize) */
end
return $\mathcal{M}$

Input  : $\mathcal{M}$ (model), $f^x$ (feature for candidate domain $x$)
Output: $D$ (distance)
Function distance ($\mathcal{M}, f^x$)
$sum = 0$
for $i = 1$ to $k$ do
    $sum = sum + |m_i - f^x_i|$
end
return $sum$
```

5 | DATASET AND CHARACTERISTICS

Our dataset was originally used by Thomas and Mohaisen in their work on detecting and clustering botnet domains using DNS traffic.[20] The dataset was collected during July of 2012 from a large DNS operator's authoritative name servers for the COM, NET, TV, and CC TLD authoritative name servers. As the registry of large TLDs, this DNS operator has a global view of DNS traffic, giving a unique observation of malware-associated DNS traffic.

TABLE 1 Conficker domain generation algorithm by variant and domains per day

Variant	Domains	TLDs
A	250	biz, info, org, net, and com
B	250	biz, info, org, net, com, … , cn
C	50k	110 ccTLDs not tv or cc

Abbreviations: TLD, top-level domain.

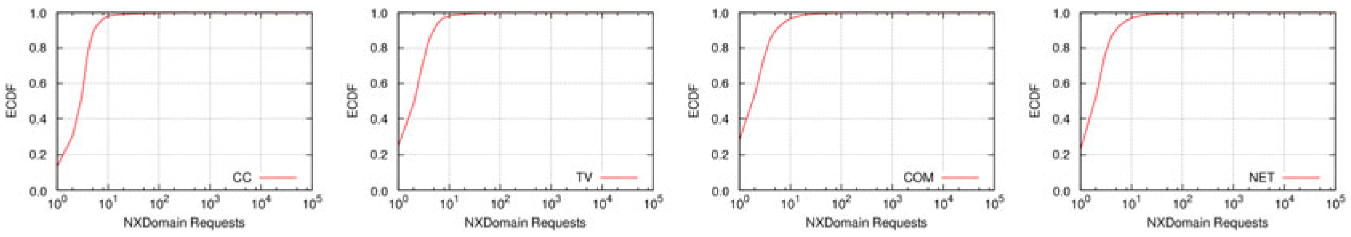

FIGURE 5 Cumulative distribution of NXDomain traffic volumes to major top-level domains, ie, .cc, .tv, .com, and .net, respectively. Notice that the majority of the domains receives small number of queries and a small percentage (∼3%) receives more than 10 queries

5.1 | Malware data

Conficker is one of the most well-known malware samples that employed the use of DGAs.[21] The family was originally discovered in 2008 and has been active for the past several years by infecting many hosts worldwide and by mutating into multiple variants, namely, Conficker A, B, C, D, and E.[22] The Conficker Working Group, a consortium of researchers and security professionals, has successfully reverse engineered the DGA and precalculated domains to be generated each day for variants A, B, and C.[23] Table 1 shows the various TLDs for variants A, B, and C and the domains to be generated on a daily basis. Variants A and B utilize the COM, NET, and CC TLDs, allowing us to analyze the DNS traffic for them. By April 2009, all domains generated by variant A were successfully locked or preemptively registered to mitigate the proliferation capabilities of the variant. Of the 15 500 domains to be generated by variants A and B in July of 2012 (corresponding to 500 domains per day over 31 days), 30 of the domains were registered in either COM or NET with active name servers resulting in the YXDomain (name exists when it should not) traffic, whereas the remaining DGA domains resulted in the NXDomain traffic.

5.2 | NXDomain data

NXDomain is the answer type for a domain name that is unable to resolve because, among other reasons, the domain name is not registered. The term was originally used to represent DNS response code 3 in RFC[24] and RFC 2308.[25] All of the data in the following corresponds to the state of the DNS resolution system by a large DNS operator in middle of 2012. We note that, while we are not able to use DNS traffic for some of the TLDs listed in Table 1, the ones using TLDs from the large DNS operator (CC, TV, NET, and COM) were captured, measured, and analyzed.[20]

In our dataset, a typical day in the COM zone has 2.5 billion NXDomain requests for more than 350 million unique second-level domains, whereas NET receives about 500 million NXDomain requests for more than 60 million unique second-level domains. Smaller zones such as TV and CC receive several magnitudes less of NXDomain traffic than COM and NET. While daily volumes of requests and unique domains are extremely high, the vast majority of the individual NXDomains observed receives very few requests within a given epoch of time. Figure 5 (subplot for each zone) shows a cumulative distribution function of the number of requests, a given NXDomain receives in 24 hours. As depicted, more than 95% of the unique second-level NXDomains receive less than 10 requests within 24 hours.

5.3 | Conficker NXDomain DNS

The vast majority of the domains generated by the Conficker DGA falls into the NXDomain category. We analyzed various aspects of the DNS traffic before, during, and after the domain's generation date to understand the lifecycle of a DGA

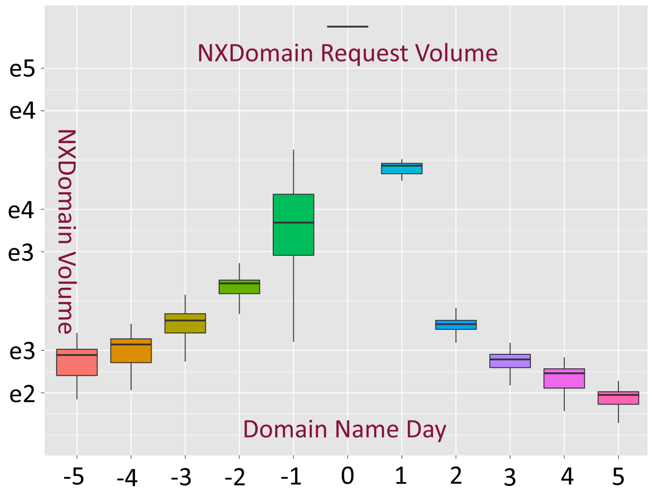

FIGURE 6 Conficker NXDomain domain name system lookups ±5 days from the domain generation date

domain with respect to the DNS traffic. Using the 2012 Conficker Domain list of precalculated DGA domains, we were able to group domains by their generation date and measure their DNS traffic.[20] Specifically, for a given domain to be generated on day x, we measured the domain's DNS traffic on days $x - 5$ to $x + 5$. Figure 6 depicts the pre-, during, and post-DNS traffic patterns for Conficker B with box plots to depict the range of DNS traffic observed on a given day. It is evident that, despite a specific generation date, DGA domains receive significant volumes of traffic during its pre and postgeneration date.

6 | EVALUATION

To evaluate the performance of our scheme, we use the dataset described in Section 5, with the head of the distribution of the dataset corresponding to malicious domains and the rest of the distribution corresponding to benign domains. With labels known in advance, we proceed to evaluate the labeling capability of our scheme. For 1-class learning and based on the distribution of the various malicious domains, we set $\Delta = 0.08$, which corresponds to 99% of the detection accuracy of all the domain name samples considered and included for building the baseline model $\mathcal{M}$. To build model $\mathcal{M}$ and to simulate a real-world scenario, we use 1000 domains. For unit a, we calculate the number of queries every hour and consider a sliding window size W as 8, 16, 24, 36, and 48 hours (thus, a window of size 24 would move a step of 1 hour at a time to simulate lazy learning of a new difference vector). We start "observing" responses for each five days (as highlighted in our dataset) before the registration of a domain. For our evaluation, we consider a variety of evaluation metrics as follows.

- **Standard metrics:** (1) True positives (T_p), ie, domains correctly identified as malicious. (2) False positives (F_p), ie, domains incorrectly marked as malicious. (3) True negative (T_n), ie, domains marked correctly as not malicious. (4) False negative (F_n), ie, domains incorrectly marked as not malicious. Using those outcomes, precision, recall, accuracy, and F1 score are $P = \frac{T_p}{T_p+F_p}$, $R = \frac{T_p}{T_p+F_n}$, $A = \frac{T_p+T_n}{T_p+T_n+F_p+F_n}$, and $F1 = 2 \times \frac{P \times R}{P+R}$.
- **Time:** We use how much in advance (before registration) a domain can be detected as a measure of "proactiveness."

The results for 2-class learning are shown in Table 2 and Figure 7 across multiple evaluation metrics. We notice that the performance of our scheme is quite promising, especially with a limited amount of knowledge (expressed in a small window size). As for **time** as an evaluation metric, we notice that our scheme can learn with an accuracy of more than 0.90 (on average) for more than 88 hours in advance ($= 24 \times 5 - 8 - 24$) and can achieve an accuracy of more than 0.99 (on average) for more than 48 hours in advance ($= 24 \times 5 - 48 - 24$).

The monitoring of domain names and their usage is a continuous process, often employed for understanding the health of the domain name state. Thus, our work does not require additional monitoring capabilities not in use today. However, for a larger window, our system would require obtaining a count of queries per domain name for a larger period of time (eg, four days). Even when we aggregate those counts per five minutes (hourly is a standard), the overall overhead per

TABLE 2 Standard measurements of performance: true positive, true negative, false positive, false negative for different windows size (average, over 24 slides for the given W size)

W	T_P	T_N	F_P	F_N	P	R	A	$F1$
8	91.3	89.4	10.6	8.7	0.89	0.91	0.90	0.90
16	97.4	92.7	7.3	2.6	0.93	0.97	0.95	0.95
24	98.1	94.5	5.5	1.9	0.95	0.98	0.96	0.96
36	99.3	95.5	4.5	0.7	0.96	0.99	0.97	0.98
48	99.4	98.3	1.7	0.6	0.98	0.99	0.99	0.99

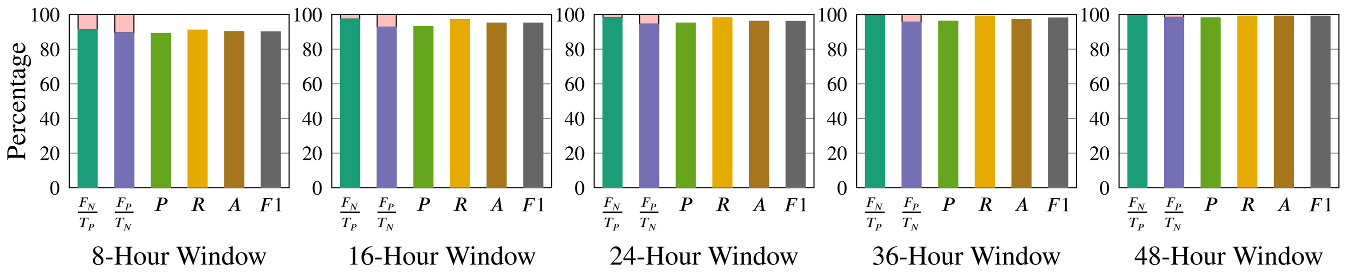

FIGURE 7 Plotting the standard measurements of performance: false negative/true positive, false positive/true negative for different windows size (average, over 24 slides for the given W size). Notice that an accuracy of 0.99 can be achieved for more than 48 hours in advance

monitoring a domain name will be 9 KB to extract our features if we are to store the entire curve (assuming 64-bit representation of a long integer). This overhead is linear in the number of days; one-day monitoring would have an overhead of 2.25 KB. A similar amount of overhead is required for storing the feature vector and the derivative. However, we notice that one does not need to store the curve and can calculate the derivative as a feature on-the-fly using the current and last count of queries. Furthermore, we notice that one can always aggregate the feature space to optimize the feature space, if storage is an issue.

7 | DISCUSSION

In this section, we discuss possible tactics that bots and botmasters can employ to evade the DRIFT system. As evident in the Conficker NXDomain DNS dataset, which clearly shows significant traffic volume before and after a Conficker-based DGA domain is registered (see Figure 6), we attribute this distinguishing pattern to a phenomena known as *clock drift*.

A drift is introduced by many factors, including network jitter, delays introduced by software, and even the environmental conditions in which the computer is operating.[26] Since DGAs tend to use time as a seed into a PRNG to ultimately generate a list of domains to query for C2 communication, any deviation from the actual time will cause the infected hosts to query domains prematurely (or late).

7.1 | Synchronizing to Internet time

Since DRIFT relies on the fact that DGA domains will be queried pre and post their registration date, one simple tactic botnet authors can employ to evade DRIFT is to ensure that the time value used for the PRNG seed originates from a *reliable* remote host, such as an Internet time server. For example, the botnet author can include code that initiates a HTTP request to a NIST (The National Institute of Standards and Technology) time server located at "http://nist.time.gov/actualtime.cgi", which returns the number of milliseconds since Jan 1, 1970.

Indeed, the aforementioned approach is the case for the Conficker family of malware, which randomly selects one of six search engines (w3.org, ask.com, msn.com, yahoo.com, google.com, and baidu.com) to obtain the time.[27] By initiating an HTTP GET request to one of these websites, the Conficker DGA can extract the date string GMT from the HTTP header. Since only the day, month, and year values are used, repeated queries on the same day would yield the same result.[27] As Leder et al pointed out, selecting such high-profile websites for time synchronization renders it impossible to simultaneously disable all target time sources in a coordinated effort.[28]

If Conficker queries one of the six high-profile websites mentioned previously for the actual time, *how is it possible that there were NXDomain queries prior to the registration of a Conficker DGA domain name on a given day?* This may indicate that an incorrect time was returned from one of those high-profile websites, which is highly unlikely. One plausible scenario could be an incorrect or Network Time Protocol (NTP) configuration, especially with virtual machines that typically run web servers. As evidenced on the official NTP.org known issues,[29] ie, “NTP server was not designed to run inside of a virtual machine. It requires a high resolution system clock, with response times to clock interrupts that are serviced with a high level of accuracy.” Another unlikely scenario is an attack on the NTP itself, which is highlighted in the work by Malhotra et al,[30] where attackers can exploit unauthenticated NTP traffic to alter the time on client systems.

The most likely scenario to explain the clock drift in Conficker, despite obtaining a valid time from a reliable internet source, is revealed in the work of Leder and Werner,[28] who illustrated an issue in the PRNG based on a reimplementation in the C programming language. Essentially, they discovered that a cross-compiled version (using MinGW) of the DGA drifts out of sync after a few hundred operations because the *log()* function used by MinGW differs slightly from the implementation in the *msvcrt.dll* used by Conficker. They noticed that a digit at position 10^{13} differed because of different roundings. Ultimately, this resulted in a single bit that was different and brought the PRNG and, therefore, the domain generation out of sync.

7.2 | Circumventing DNS

As stated previously, one of the main mechanisms of DRIFT to proactively identify DGA domain names is the use of the DNS protocol for C2 communication. As such, botnets can simply circumvent DNS (ie, use IP addresses directly) entirely or use another communication protocol (eg, peer-to-peer) that does not rely on domain names or DNS resolution to ultimately evade detection by the DRIFT system. In the following, we discuss these circumvention techniques and alternative C2 communication protocols and their implications.

Hardcoding C2 addresses. In the early days of botnets, bot authors would hardcode the IP address of the C2 server in the bot binary to eliminate the use of DNS from the picture, thus making their activities stealthier.[31] However, as Khattak et al pointed out, this is a rather primitive method because an obvious pitfall is reverse engineering the bot binary to potentially reveal the C2 server, thus leading to a C2 server hijack. Another disadvantage of this approach is that network administrators can easily blacklist C2 IPs at a network gateway using an access control list, thereby severing call-back channels of all the bots.

Along the same lines, bot authors can also hardcode the domain names for C2 servers into the bot binary, which is a better approach than IP address hardcoding from the botnet point-of-view.[31] This is due to that fact that, if the IP address associated with the domain name is taken down, the bot master can still carry out its malicious activities by mapping the domain name to a new IP address (while requiring no updates on the bot end). As mentioned earlier, using a single domain name for C2 communication can constitute a single point of failure, which gave rise to the adoption of DGAs. This in turn made it difficult for law enforcement agencies because taking down a domain is a complicated process involving several formalities.[32] By the time a suspected DGA domain is taken down, the botnet has typically already moved to a new domain (in a process known as bot-herding).[31]

DNS NXDomain hijacking. The NXDomain response from botnets querying DGA domains prematurely is the key mechanism behind the DRIFT system. With that being said, altering or "hijacking" this response would throw off the measurements utilized by DRIFT because it analyzes the number of NXDomain responses over a fixed period of time (eg, hour). As a case in point, a previous study[33] has shown that the DNS servers operated by certain ISPs may hijack such responses in an effort to “assist” users by sending them to a “search” page filled with advertisements, rather than returning the NXDomain response. Disconcerting as it sounds, a recent large-scaled study[34] concluded that NXDomain hijacking by ISPs is quite rare because it only affected 4.8% of the 1.2 M network nodes they measured (which were spread across 14 k autonomous systems (ASes) in 172 countries).

Another possible way that botmasters can mitigate NXDomain responses (thereby avoiding detection by DRIFT) is to employ caching DNS servers or “rogue” DNS servers on compromised systems. A typical caching DNS server does not contain any domain resource records, it simply resolves DNS queries from clients and caches the answer to respond quickly for future queries. As Heron[35] described, adversaries would only need a compromised machine or a server hosted in a country with lax cyber crime laws. Essentially, when a botnet-originated DNS query is accepted by one of these servers, it could avoid returning a NXDomain response and possibly provide a legitimate IP address under its control.

7.3 | Botnet topologies

Centralized botnet. A centralized botnet is a topology with a clear distinction between the botmaster and its participating bots, shown prominently in 1. In this topology, the botmaster prepares the C2 channel to issue commands so that bots accessing the channel can receive instructions to take future actions. In addition to HTTP, these commands can be sent through various protocols such as UDP, TCP, and IRC. In the case of using preassigned IP addresses for TCP or UDP, the approach by DRIFT in utilizing the characteristics of the DNS NXDomain responses cannot be directly applied here. However, if a centralized botnet does not use HTTP or DNS resolution, the botmaster can easily be thwarted simply by using the countermeasures we discussed previously to determine the C2 server's IP address. We highlight that DRIFT's approach focuses on DGA-based botnets, which have become increasingly difficult to take down due to the coordination of law enforcement and different agencies.[32]

Decentralized botnet. In contrast to a centralized topology, decentralized botnets have no obvious commander, which makes the C2 communication not concentrated around a specific node (ie, botmaster). Applying DRIFT's detection mechanisms here will likely be difficult, especially if the C2 communication protocol does not use DNS resolution. For example, a typical decentralized botnet uses peer-to-peer technology and works by allowing the participating bots to relay the commands to each other (rather than from a centralized node). The decentralized topology is free from the single-point-of-failure issues discussed previously because different parts of the botnet also functions as C2 servers. In summary, DRIFT will likely perform well for decentralized botnets that use DNS resolution because it takes advantage of patterns in NXDomain responses due to clock drift. However, since the role of the C2 server is distributed, the amount of NXDomain responses that occur will be significantly lower than that of the centralized botnet. In future work, we will use the same approach for the analysis of decentralized botnets using DNS resolution.

8 | RELATED WORK

8.1 | Domain gorithm

Primarily used for a botnet's C2 channel, DGAs are widely used in malware families such as Conficker, Kraken, and Torpiq. As the potential impact of DGAs continues to expand, a wide range of research has been well underway in the broad field of computer security. One of the first academic publications that introduced the concept of malware utilizing DGAs was in 2009 by Stone-Gross et al,[36] who described their experiences attempting to seize control of the Torpig botnet. They pointed out that, in the past, botnet authors would use IP fast-flux techniques where a certain domain is mapped to a set of frequently changing IP addresses to avoid take downs. Realizing that a single domain name constituted a single point failure, the authors discovered that the Torpig botnet started using a *domain flux* technique that employed the use of a DGA for locating its C2 server. Shin et al[22,37] conducted a large-scale survey of the distribution of Conficker (one the most notorious DGA-based malwares) and the effectiveness of the existing detection systems. The Kraken botnet, another notorious DGA-based malware, was closely analyzed by Royal,[38] who detailed its behavior and provided sample DGA domains as well as MD5 hashes. Recently, Fu et al[39] proposed two new DGAs based on hidden Markov models and probabilistic context-free grammars, respectively, which can avoid current detection systems.

8.2 | Detection of DGA-based botnet

To counter the prevalence of DGA-based malware, several approaches have been proposed over the years to identify algorithmically-generated malicious domain names by DGA-based botnets. As mentioned previously, all of these works usually fall into two schools of thought either (1) relying on reverse-engineering of the bot software or (2) using the intrinsic features of the generated domains. Table 3 summarizes the most recent studies on how to proceed with the detection of DGA-based botnets. As a result of the development of machine learning technology, the rate of research for identifying algorithmically-generated domains using various intrinsic features is high.

Approach through reverse-engineering. The recent work by Plohmann et al[54] is the epitome of the first school of thought that relies on reverse-engineering malware. The authors performed a comprehensive measurement study of 43 DGA-based malware families and variants in a bottom-up fashion by reimplementing their algorithms to ultimately produce over 159 million unique DGA domains. Using a WHOIS dataset from DomainTools (containing over 9 billion WHOIS records spanning 14 years), the authors removed any DGA domain for families not in the WHOIS dataset, which

TABLE 3 Summary of domain generation algorithm domain identification approaches

Study	Approach	Malware Families
Stone-Gross et al[36]	1) Reverse-engineer	1 (Torpig)
Yadav et al[40]	2) Intrinsic	3 (Conficker, Torpig & Kraken)
Antonakakis et al[12]	2) Intrinsic	12 Botnets
Barabosch et al[41]	1) Reverse-engineer	4 Families
Guerid et al[42]	2) Intrinsic	2 (Conficker & Kraken)
Zhang et al[43]	2) Intrinsic	3 (Conficker, Torpig & Srizbis)
Haddadi and Zincir-Heywood[44]	2) Intrinsic	3 (Conficker, Kraken & Alexa)
Nhauo and Sung-Ryul[45]	2) Intrinsic	3 (Conficker, Torpig & Kraken)
Mowbray and Hagen[46]	2) Intrinsic	19 DGAs
Schiavoni et al[47]	2) Intrinsic	3 (Conficker, Torpig & Bamital)
Bilge et al[48]	2) Intrinsic	2 (Conficker, Torpig) & malicious domain lists
Sharifnya and Abadi[49]	2) Intrinsic	3 (Conficker, Kraken & Murofet)
Grill et al[50]	2) Intrinsic	6 families
Wang et al[51]	2) Intrinsic	1 (Conficker)
Kwon et al[52]	2) Intrinsic	26 Botnets
Zhang et al[53]	2) Intrinsic	2 (Conficker & Kraken)
Plohmann et al[54]	1) Reverse-engineer	43 Families
Wang et al[55]	2) Intrinsic	3 (newGoZ, Ramnit & Qakbot)

left them with a set of over 18 million unique DGA domains. By studying the registration status of these DGA domains, they were able to determine that their domain dataset can be used for both predictive blocking of C2 channels and the accurate identification of malware families and campaigns with virtually no false positives.

Approach employing intrinsic features. More research is being conducted to identify malicious domains through intrinsic features such as behavior and dynamics in DGA-based botnets. Yadav et al[40] conducted a study that computed the information entropy of the distribution of alphanumerics (unigram and bigram) within a group of domain names. Antonakakis proposed *Pleiades*, which focuses on NXDomain responses. Pleiades classifies the domains based on the similarity to detect malicious domain names based on known models of DGA. The approach suggested by Guerid et al[42] performs bot grouping based on NXDomain responses in privacy-preserving manner by using Bloom filters. Zhang et al[43] built the system, detecting the DGA-based malicious domain names by taking entropy, bigram, and length into account. Haddadi and Zincir-Heywood[44] proposed a stateful-SBB classifier based on the genetic programming, which takes string domain name as input to determine whether it is malicious. Nhauo and Sung-Ryul[45] implemented a support vector machine classifier, which only takes a domain name as input to reduce the burden of collecting and managing large amounts of metadata. Mowbray and Hagen[46] identified DGA domain names by analyzing client IP addresses with abnormal distributions of second-level strings lengths in their DNS queries. *Phoenix* presented by Schiavoni et al[47] labels the malicious domain names automatically using DNS and IP-related features, which can be applied to not only groups of domains but also a single domain. Bilge et al[48] proposed a system, ie, *Exposure*, which utilizes four time-based, four DNS answer-based, five TTL value-based, and two domain name-based features to detect DGA-based domains. *DFBotKiller* proposed by Sharifnya and Abadi[49] is a reputation-based system, which distinguishes malicious domains by considering suspicious activities as well as DNS query failures. Grill et al[50] presented a DGA-malware detection system only using NetFlow information, an aggregation of all packets between a source IP and port pair to a destination IP and port pair on the same protocol. Wang et al[51] employs social network analysis, which divides the hosts and NXDomains into clusters and identifies them as either benign or malicious. *PsyBoG* proposed by Kwon et al[52] is a malicious domain name detection system that uses a signal processing technique and power spectral density analysis. Zhang et al[53] presented *Botdigger*, which can detect an individual bot by gathering and analyzing the data from a single network. Wang et al[55] devised *BotMeter*, which assesses the populations of DGA-based botnets over large networks by analyzing the DNS lookups at upper-level DNS servers.

Comparison with previous work. While some of the previous works described earlier use NXDomain queries as a main feature in their detection systems (eg, the work of Antonakakis et al[12]), none have examined their frequency and timing in respect to when the target DGA domains are registered. Our approach is the only known method that relies on the fact that these NXDomain queries are attributed to clock drifts in which an infected machine's DGA becomes unsynchronized with their C2 server's DGA. As a result, our approach leverages the inherent patterns of NXDomain queries prior to the actual registration of the DGA domain, which allows us to detect them with a computationally-inexpensive classification algorithm.

9 | CONCLUSION

In this paper, we presented a system called DRIFT to proactively detect algorithmically-generated malicious domain names typically employed by botnets. We highlighted the fact that DGA domains tend to have a large number of DNS queries prior to registration, resulting in NXDomain responses, which is then followed by a gradual overall decline. We then devised a detection algorithm using the notion of the difference function over the number of NXDomain responses for a given domain with a sliding time window. Using the DNS traffic gathered from certain TLDs for the precalculated list of generated domains by the Conficker malware variants, our detection algorithm was able to achieve 99% accuracy (on average) as early as 48 hours prior to registration.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under grant CNS-1809000, by Korea National Research Foundation under grant 2016K1A1A2912757, and by Chung-Ang University Research Grant (2018).

ORCID

Jeffrey Spaulding http://orcid.org/0000-0003-0047-5156

REFERENCES

1. Middleton P. Forecast analysis: Internet of Things–endpoints, worldwide, 2016 update. 2017. http://gtnr.it/2oRo4aN
2. OWASP. OWASP Internet of Things (IoT) Project. http://bit.ly/1k0dSrD
3. OVH. The DDoS that didn't break the camel's VAC. http://bit.ly/2D36Ufm
4. Antonakakis M, April T, Bailey M, et al. Understanding the Mirai botnet. In: Proceedings of the 26th USENIX Security Symposium; 2017; Vancouver, Canada.
5. Wang A, Mohaisen A, Chen S. An adversary-centric behavior modeling of DDoS attacks. In: Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017; Atlanta, GA.
6. Kountouras A, Kintis P, Lever C, et al. Enabling network security through active DNS datasets. In: Proceedings of the 19th International Symposium on Research in Attacks, Intrusions and Defenses (RAID); 2016; Paris, France.
7. Holub A, Colford P. The future is here - assaulting the Internet with Mirai. 2017. http://bit.ly/2oSkJrU
8. Nadji Y, Antonakakis M, Perdisci R, Dagon D, Lee W. Beheading hydras: performing effective botnet takedowns. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS); 2013; Berlin, Germany.
9. Lever C, Kotzias P, Balzarotti D, Caballero J, Antonakakis M. A lustrum of malware network communication: evolution and insights. Paper presented at: 2017 IEEE Symposium on Security and Privacy (SP); 2017; San Jose, CA.
10. Mohaisen A, Alrawi O. Unveiling Zeus: automated classification of malware samples. In: Proceedings of the 22nd International Conference on World Wide Web (WWW); 2013; Rio de Janeiro, Brazil.
11. Yadav S, Reddy AKK, Reddy AL, Ranjan S. Detecting algorithmically generated malicious domain names. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC); 2010; Melbourne, Australia.
12. Antonakakis M, Perdisci R, Nadji Y, et al. From throw-away traffic to bots: detecting the rise of DGA-based malware. In: Proceedings of the 21st USENIX Conference on Security Symposium (Security); 2012; Bellevue, WA.
13. ICANN. Registrar accreditation: history of the shared registry system. 2015. http://bit.ly/1NWexTL
14. ICANN. Domain name registration process. 2017. https://go.icann.org/2ymkyN9
15. Chang W, Mohaisen A, Wang A, Chen S. Measuring botnets in the wild: some new trends. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS); 2015; Singapore, Singapore.
16. Wang A, Mohaisen A, Chang W, Chen S. Revealing DDoS attack dynamics behind the scenes. In: Proceedings of the International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA); 2015; Milan, Italy.
17. Wang A, Mohaisen A, Chang W, Chen S. Delving into Internet DDoS attacks by botnets: characterization and analysis. In: Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2015; Rio de Janeiro, Brazil.
18. Chaudhuri BB. A new definition of neighborhood of a point in multi-dimensional space. *Pattern Recogn Lett*. 1996;17(1):11-17.

19. Han E-H, Karypis G. Centroid-based document classification: analysis and experimental results. In: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD); 2000; Lyon, France.

20. Thomas M, Mohaisen A. Kindred domains: detecting and clustering botnet domains using DNS traffic. In: Proceedings of the 23rd International Conference on World Wide Web (WWW); 2014; Seoul, Korea.

21. Mohaisen A, Alrawi O, Mohaisen M. AMAL: high-fidelity, behavior-based automated malware analysis and classification. *Comput Secur*. 2015;52:251-266.

22. Shin S, Gu G. Conficker and beyond: a large-scale empirical study. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC); 2010; Austin, TX.

23. The Conficker Working Group. 2012. http://bit.ly/1kAYsJA

24. Mockapetris P. Domain names: implementation and specification (November 1987). RFC 1035. 2004.

25. Andrews M. Negative caching of DNS queries (DNS NCACHE). RFC 2308. 1998.

26. Jackson J. Why computers still struggle to tell the time. 2015. http://bit.ly/2BOW5R2

27. Porras P, Saïdi H, Yegneswaran V. A foray into Conficker's Logic and Rendezvous Points. In: Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (LEET); 2009; Boston, MA.

28. Leder F, Werner T. Know your enemy: containing Conficker. 2009. http://bit.ly/2ESrRQ1

29. NTP.org. Known operating system issues. http://support.ntp.org/bin/view/Support/KnownOsIssues

30. Malhotra A, Brakke E, Goldberg S. Attacking the network time protocol. In: Proceedings of the Network and Distributed System Security Symposium (NDSS); 2016; San Diego, CA.

31. Khattak S, Ramay NR, Khan KR, Syed AA, Khayam SA. A taxonomy of botnet behavior Detection, and defense. *IEEE Commun Surv Tutor*. 2014;16(2):898-924.

32. Piscitello D. Guidance for preparing domain name orders, seizures & takedowns. Thought Paper. 2012. https://go.icann.org/2H1npLi

33. Dagon D, Lee C, Lee W, Provos N. Corrupted DNS resolution paths: the rise of a malicious resolution authority. In: Proceedings of the Network and Distributed System Security Symposium (NDSS); 2008; San Diego, CA.

34. Chung T, Choffnes D, Mislove A. Tunneling for transparency: a large-scale analysis of end-to-end violations in the Internet. In: Proceedings of the 2016 Internet Measurement Conference (IMC); 2016; Santa Monica, CA.

35. Heron S. Working the botnet: how dynamic DNS is revitalising the zombie army. *Netw Secur*. 2007;2007(1):9-11.

36. Stone-Gross B, Cova M, Cavallaro L. et al. Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS); 2009; Chicago, IL.

37. Shin S, Gu G, Reddy N, Lee CP. A large-scale empirical study of Conficker. *IEEE Trans Inf Forensics Secur*. 2012;7(2):676-690.

38. Royal P. Analysis of the kraken botnet. 2008. http://www.flatland.tuxfamily.org/repo/papers_malwares/KrakenWhitepaper.pdf

39. Fu Y, Yu L, Hambolu O, et al. Stealthy domain generation algorithms. *IEEE Trans Inf Forensics Secur*. 2017;12(6):1430-1443.

40. Yadav S, Reddy AKK, Reddy ALN, Ranjan S. Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. *IEEE/ACM Trans Netw*. 2012;20(5):1663-1677.

41. Barabosch T, Wichmann A, Leder F, Gerhards-Padilla E. Automatic extraction of domain name generation algorithms from current malware. In: Proceedings of NATO Symposium IST-111 on Information Assurance and Cyber Defense; 2012; Koblenz, Germany.

42. Guerid H, Mittig K, Serhrouchni A. Privacy-preserving domain-flux botnet detection in a large scale network. In: 2013 Fifth International Conference on Communication Systems and Networks (COMSNETS); 2013; Bangalore, India.

43. Zhang Y, Zhang Y, Xiao J. Detecting the DGA-based malicious domain names. In: Proceedings of the International Conference on Trustworthy Computing and Services (ISCTCS); 2013; Beijing, China.

44. Haddadi F, Zincir-Heywood AN. Analyzing string format-based classifiers for botnet detection: GP and SVM. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC); 2013; Cancun, Mexico.

45. Nhauo D, Sung-Ryul K. Classification of malicious domain names using support vector machine and bi-gram method. *Int J Secur Appl*. 2013;7(1):51-58.

46. Mowbray M, Hagen J. Finding domain-generation algorithms by looking at length distribution. In: Proceedings of the 2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW); 2014; Naples, Italy.

47. Schiavoni S, Maggi F, Cavallaro L, Zanero S. Phoenix: DGA-based botnet tracking and intelligence. In: Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA); 2014; Egham, UK.

48. Bilge L, Sen S, Balzarotti D, Kirda E, Kruegel C. Exposure: a passive DNS analysis service to detect and report malicious domains. *ACM Trans Inf Syst Secur*. 2014;16(4):1-28. Article No 14.

49. Sharifnya R, Abadi M. DFBOtKiller: domain-flux botnet detection based on the history of group activities and failures in DNS traffic. *Digit Investig*. 2015;12:15-26.

50. Grill M, Nikolaev I, Valeros V, Rehak M. Detecting DGA malware using NetFlow. In: Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM); 2015; Ottawa, Canada.

51. Wang T-S, Lin C-S, Lin H-T. DGA botnet detection utilizing social network analysis. In: Proceedings of the 2016 International Symposium on Computer, Consumer and Control (IS3C); 2016; Xi'an, China.

52. Kwon J, Lee J, Lee H, Perrig A. PsyBoG: a scalable botnet detection method for large-scale DNS traffic. *Comput Netw*. 2016;97:48-73.

53. Zhang H, Gharaibeh M, Thanasoulas S, Papadopoulos C. BotDigger: Detecting DGA bots in a single network. In: Proceedings of the IEEE International Workshop on Traffic Monitoring and Analaysis (TMA); 2016; Louvain-la-Neuve, Belgium.

54. Plohmann D, Yakdan K, Klatt M, Bader J, Gerhards-Padilla E. A comprehensive measurement study of domain generating malware. In: Proceedings of the 25th USENIX Security Symposium; 2016; Austin, TX.

55. Wang T, Hu X, Jang J, Ji S, Stoecklin M, Taylor T. BotMeter: charting DGA-botnet landscapes in large networks. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS); 2016; Nara, Japan.

How to cite this article: Spaulding J, Park J, Kim J, Nyang D, Mohaisen A. Thriving on chaos: Proactive detection of command and control domains in internet of things-scale botnets using DRIFT. *Trans Emerging Tel Tech*. 2018;e3505. https://doi.org/10.1002/ett.3505