Scalable analytics to detect DNS misuse for establishing stealthy communication channels

D. L. Schales
J. Jang
T. Wang
X. Hu
D. Kirat
B. Wuest
M. Ph. Stoecklin

The Domain Name System (DNS) protocol is one of the few application protocols that are allowed to cross network perimeters of organizations. However, comprehensive monitoring of DNS traffic has been often overlooked in many organizations' cybersecurity strategies. As such, DNS provides a highly attractive channel for advanced threat actors and botnet operators to establish hard-to-block and stealthy communication channels between infected devices and command-and-control (C&C) infrastructures. Fast-fluxing (FF) and domain name generation algorithms (DGAs) are two well-known public DNS exploitation techniques to build agile C&C infrastructures. The detection of FF and DGA domain names is a big data problem, as it requires analyzing millions of DNS queries and replies over extended time periods. In this paper, we propose two algorithms to perform DNS analytics and effectively detect FF and DGA domain names. More importantly, we describe how the algorithms are implemented using two big data processing models: MapReduce and Feature Collection and Correlation Engine. The algorithms and implementation proposed are iterative and scale over long analysis periods. We describe the implementations and provide an evaluation complemented with case studies on 50 days of real-world DNS data consisting of more than 40 billion events, collected within a large corporate network.

1. Introduction

A botnet is a network of many malware-infected machines (bots) that can be remotely controlled by their operator (botmaster) through command-and-control (C&C) channels. Being the workhorse of varied large-scale attacks (e.g., denial-of-service, e-mail spamming, key logging, and click fraud), botnets represent a major threat to Internet security. A fundamental aspect of any botnet is that of coordination, i.e., how the bots identify and communicate with their botmasters (C&C servers). Conventionally, bots locate C&C servers using their IP (Internet Protocol) addresses, Domain Name System (DNS) domain names, or node IDs in peer-to-peer overlays.

Over the recent years, we have witnessed a rampant use of the DNS infrastructures to establish C&C channels in major botnet crimewares (e.g., Srizbi [1], Ranbyus [2], Torpig [3], Conficker [4], Bobax [5], and Gameover Zeus [6]) because the DNS protocol is one of the few application protocols that are typically allowed to cross network perimeters of organizations with almost no monitoring. In particular, two major classes of DNS-based coordination strategies have been discovered. In *fast-fluxing*, the botnet operator associates multiple possible IP addresses with a single domain name and frequently changes the associated IP address. In *domain-fluxing*, bots employ *domain generation algorithms* (DGAs) to dynamically compute a list of pseudo-random domains. This list is generated independently by each bot and is often regenerated periodically. The bot attempts to contact domains on the list in a certain order until one succeeds (i.e., the domain resolves to an IP address, and the corresponding

Digital Object Identifier: 10.1147/JRD.2016.2557639

server provides a valid response according to the botnet's protocol) or aborts after a certain number of trials. The botmaster needs to register only a few domains on the list to serve as C&C servers. As an example, each Conficker.C [4] worm generates 50,000 random domains every day, among which it attempts to contact up to 500 domains, giving itself a 1% chance of being connected if the botmaster registers only one domain per day.

The use of fast-fluxing and domain-fluxing rather than a list of hard-coded domains significantly strengthens a botnet's capability to evade detection and takedown. First, the frequent change of IP addresses or the sheer amount of potential rendezvous points makes it extremely difficult to preemptively eliminate C&C channels (e.g., using blacklisting to restrict outbound communication, sink-holing domain names that bots will contact in the future by registering the names). Moreover, in domain-fluxing, even if the current C&C domains or IP addresses are captured and taken down, the bots will eventually identify the relocated C&C servers via looking up the next set of automatically generated domains. Furthermore, due to the use of public-key encryption, it is often infeasible to mimic communication from the botmasters, because the bots reject commands not signed by their botmasters.

Being one backbone component of the Internet, the DNS infrastructure of a large organization oftentimes generates a large amount of data every day. Thus the detection of fast-fluxing and domain-fluxing domain names is essentially a Big Data problem as it requires analyzing millions of DNS queries and replies over extended time periods.

In this paper, we propose two novel algorithms that perform DNS analytics and effectively detect fast-fluxing domain names and domain-fluxing. Our algorithms are behavior based and are thus not limited to known botnets, but can detect previously unseen types of fast-fluxing and domain-fluxing. In comparison with existing work (see Section 8 of this paper), the fast-fluxing detection makes use of a larger feature set to eliminate false positives originating from legitimate services, and the domain-fluxing detection attempts to cluster and identify the names that eventually resolve and are used to establish C&C communication. To satisfy the scalability demand, we implement both algorithms using two processing environments: the MapReduce environment and the Feature Collection and Correlation Engine (FCCE), big data analytics middleware tailored for security analytics. The algorithms and implementation proposed are iterative and scale over long analysis periods. We provide extensive evaluation of the proposed algorithms and implementations over very large real-world DNS datasets (> 250 billion events), collected within a large corporate network.

2. Misusing the DNS for establishing communication channels

In this paper, we consider two main techniques of how the DNS is used by malware and C&C infrastructures (such as botnets) to identify rendezvous points to establish communication channels: *fast-fluxing domain names* (FF) and *domain generation algorithms* (DGAs).

2.1. Fast-fluxing domain names

In this fast-fluxing technique, a bot establishes communication to its C&C using a single domain name that resolves to varying IP addresses over a short time. This frequent change of IP addresses circumvents the traditional IP-based traffic filtering because it takes some time to detect a new IP as malicious. In traditional security systems, blocking domain names is generally more complicated compared to blocking IP addresses (e.g., via a firewall or intrusion protection system rule). Fast-fluxing can be understood as a behavior of the domain name system, i.e., the IP associations returned by the DNS system change frequently for a given domain name. Fluxing may occur on a minute granularity as well as over multiple hours. **Figure 1** shows an example of the life-cycle of a fast-fluxing domain name: (1) the attacker chooses a domain name ("attack3r.com") and (2) embeds in the malware. In step (3), the attacker can now use the public DNS system to register the domain and frequently update its resolving IP addresses that point to C&C servers under the attacker's control. After infection (4) and execution of the malware on a victim's machine, the malware (5) starts to query the DNS system for associated IP addresses and connects to the resolved IP addresses. While some IP addresses can be blocked (e.g., by means of blacklists), in step (6) the malware continues to query until it eventually succeeds.

One key limitation of fast-fluxing domain names is that once the domain name is known—and the security infrastructure allows blocking of DNS lookups for a given name (or sink-holing)—the malware's communication channel is interrupted.

2.2. Domain-fluxing: domain generation algorithms

In this technique, instead of a fixed embedded domain name, malware is equipped with an algorithm, known as DGA, to generate domain names at runtime, which are used to establish the communication channel. The use of DGA—instead of a list of hard-coded domains—significantly strengthens the capability to evade detection and/or take down. Foremost, the unlimited number of potential rendezvous points makes it extremely difficult to preemptively eliminate communication channels (e.g., using blacklisting to restrict outbound communication or pre-registering domains that bots may contact in the

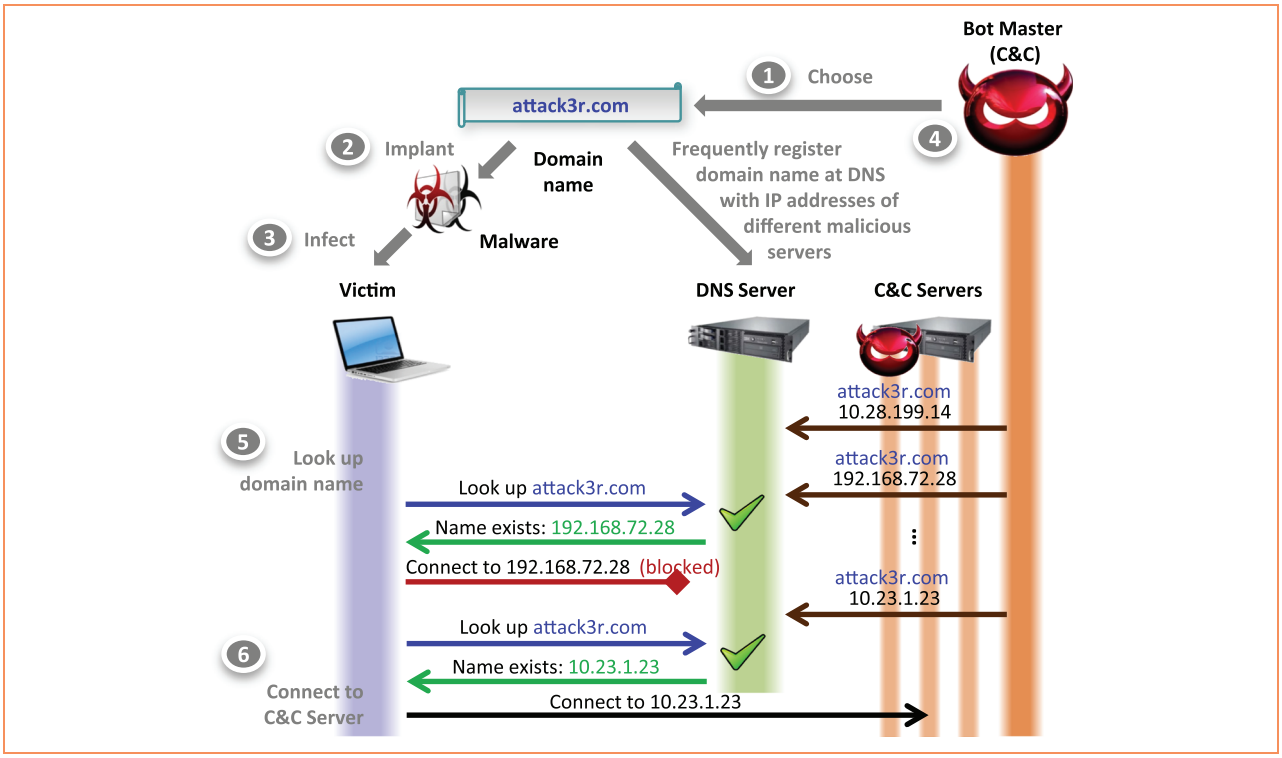

Figure 1

Establishment of a communication channel using fast-fluxing (FF) domain names.

future). Moreover, even if the current communication domains or IP addresses are detected and taken down, bots will eventually identify IP addresses of the relocated C&C servers by looking up the next set of automatically generated domains.

Figure 2 shows an example of the life-cycle of a DGA-based communication channel: Instead of the domain name, the attacker (1) configures and (2) implants the algorithm in the malware. Upon infection (3), the malware starts to generate domain names and tries to resolve those using DNS (4). Many names will result in no finding (NXDOMAIN), as the name has not been registered. However, once the attacker decides to establish a communication channel, in step (5) he may predict and register a future domain name and point it to an IP address of a server under the attacker's control. Eventually, the malware will look up this name (6) and connect to the corresponding IP address.

From the earliest documented DGA case used by the Srizbi botnet [1], to the most recently discovered Ranbyus [2], an alarming number of botnet crimewares now employ DGA technique as their primary communication strategy. Compared with alternatives (e.g., peer-to-peer C&C infrastructure), this strategy provides a tremendous level of evasiveness and agility. Well-known instances of DGA-bots include Torpig [3], Conficker [4], Murofet [7], Kraken [5], Bobax [5], Necurs [8], and Gameover Zeus [6].

The simplest DGA algorithm is a time-independent and deterministic DGA, which uses static hard-coded seeds and produces the same set of domain names every time they are executed (e.g., Kraken). Another common algorithm is a time-dependent and deterministic DGA, in which seeds are changing on a regular basis. For example, Conficker uses the current date as a seed. It is possible to deterministically pre-compute the domain names of these types of DGAs, and these domain names can be registered beforehand. The more advanced type of algorithm is a time-dependent and non-deterministic DGA, wherein the seeds cannot be anticipated; therefore, pre-computation is not possible. For example, Torpig [3] and Necurs [7] use popular trending topics of the Twitter service and foreign exchange rates as seeds, respectively.

2.3. Evolution of methods to establish C&C channels

Since the security industry continuously works toward taking down botnet and C&C infrastructures, attackers need to innovate and look for novel methods on how their channels can be established reliably. In **Figure 3**,

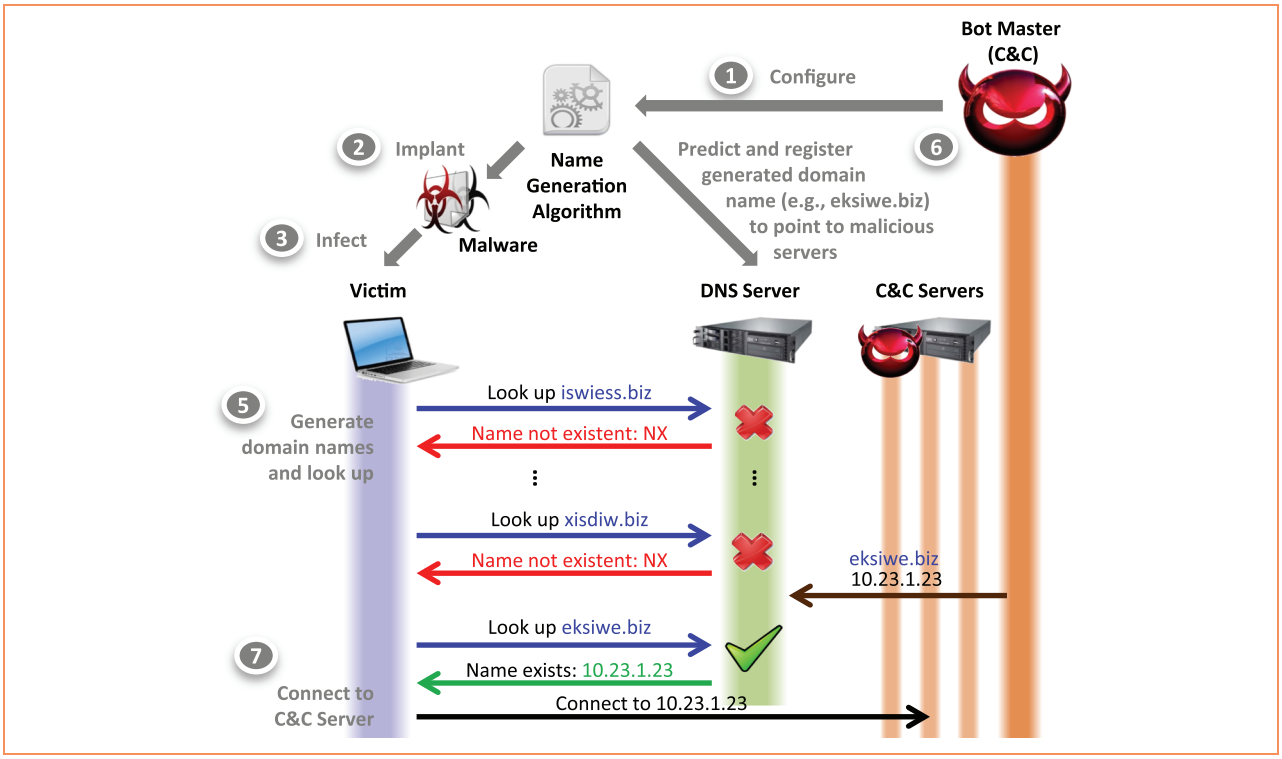

Figure 2

Establishment of a communication channel using domain generation algorithms (DGA).

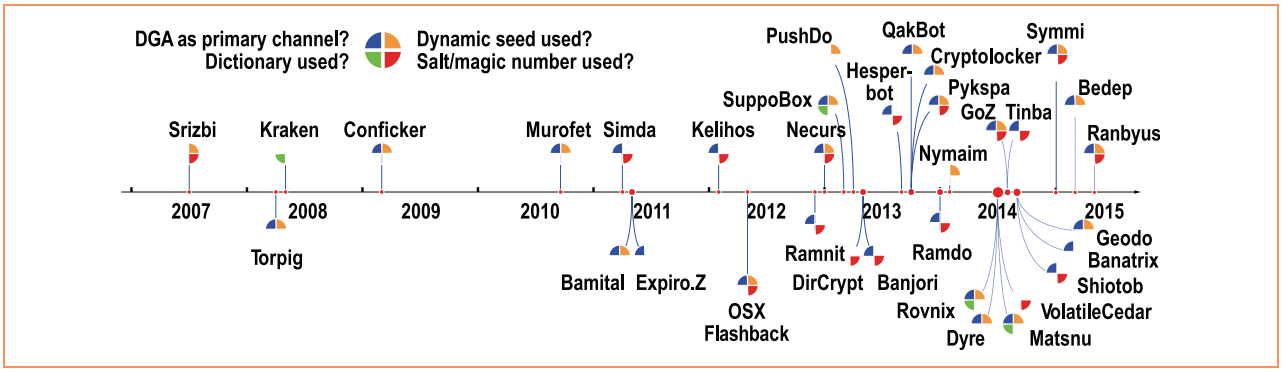

Figure 3

Evolution of DGA-based C&C channel establishment methods used by botnets between 2007 and 2015.

we plot the evolution of the methods found in large and well-studied botnets between 2007 and 2015. We assessed four characteristics of the methods (going clockwise around the small circular symbol in Figure 3), and these characteristics involve (1) whether domain-fluxing (or DGAs) was used as the primary channel to establish the communication, (2) whether a dynamic seed was used to configure the DGA, (3) whether a salt or magic number was used to initialize the algorithm, and (4) whether a dictionary was used to generate domain names.

It can be seen that the overall number of botnets appearing per year increases throughout the time interval. At the same time, the complexity of their methods increases over the years: the vast majority of botnets

employ DGAs along with one or more "enhancements" to complicate the detection of their activity (e.g., a dictionary), to run multiple campaigns in parallel (e.g., salt), or to prevent prediction of the domain names (e.g., dynamic seeds). It is particularly remarkable to observe that dynamic seeds have become increasingly popular starting from 2013. This technique adds an additional overhead for attackers as domain names cannot be registered ahead of time, since the attacker is unable to predict the names looked up, beyond a short time frame.

3. Processing environment overview

Our corporate network dataset contains more than 1 billion DNS replies (204 GB, uncompressed) per day on average, i.e., more than 12,000 messages per second. The processing of such huge data volumes pose significant challenges such as the scalability in terms of analyzing and storing the incoming DNS messages. We employ two big data analytics platforms to efficiently process the DNS messages and to perform analytics over long time periods: MapReduce in Apache Hadoop[**] and FCCE.

3.1. MapReduce

In order to store DNS messages, we utilize the Hadoop Distributed File System (HDFS). HDFS provides high reliability and availability of data, and offers highly scalable data access across multiple nodes. While having DNS data stored on HDFS, we design our distributed analytics algorithms for detecting fast-fluxing domain names and DGA-generated domain names using the MapReduce processing model in Apache Hadoop. MapReduce is a distributed computing model used to perform large-scale data analysis using a cluster of multiple nodes [9]. In the MapReduce framework, the Map and Reduce functions are defined with respect to $\langle key, value \rangle$ pairs. In the Map phase, the master Hadoop node takes a list of input pairs $\langle key_i, value_i \rangle$, splits them into smaller chunks of work, and distributes the chunks to worker nodes. In parallel, each worker node applies Map function to process the chunk of input data, and passes the output pairs $\langle key_o, value_o \rangle$ to the Reduce phase. Then, in the Reduce phase, the list of outputs $\langle key_o, value_o \rangle$ from multiple worker nodes are combined and assembled to generate the final output. We will discuss how we implement our fast-fluxing analytics in Section 4.2, and we discuss DGA analytics in Section 5.5 by leveraging the MapReduce framework.

3.2. Feature collection and correlation engine

As mentioned, we also use a scalable and distributed analysis engine, called Feature Collection and Correlation Engine (FCCE) [10]. FCCE is designed (1) to collect and extract features from geographically distributed large data sets, and (2) to find correlations across a diverse set of data types spanning over large time windows with very small latency and with minimal access to raw data. As a distributed key-value data management system, FCCE is optimized to extract, normalize, store, retrieve, and correlate features from diverse data sources. FCCE is also resilient with respect to node failures within the distributed engine architecture. Furthermore, FCCE provides a unified interface for both historical and real-time data analysis, which is critical in cyber security analytics to perform forensic analysis and real-time detection. Extracting features from data and creating their corresponding key/value pairs in FCCE essentially serves as the Map phase of the MapReduce computation. All the values associated with a particular key are aggregated in the same key, and FCCE offers quick data retrieval at the Reduce phase.

In our setup, eight distributed nodes are used for ingesting live raw data from geographically distributed data sources (taps installed next to the DNS servers), and extracted features are forwarded to six nodes for persisted storage and query services. To give a sense of scale to the amount of data processed in our deployment, the insertion rate of DNS requests and replies exceeds 12,400 messages per second on average, resulting in over 17,550 features extracted and inserted per second, with regular burst rates as high as 312,000 per second. We describe how we use the scalable and distributed feature collection and correlation capability of FCCE in Sections 4.2 and 5.5.

4. Fast-fluxing analytics

4.1. Detecting fast-fluxing domain names

At the high level, (fast) fluxing domain names are detected by monitoring the number of IP addresses associated with a particular domain name. More specifically, for the given time period, all IP addresses for a fully qualified domain name (FQDN) are aggregated with the name. If the number of IP addresses associated with the name exceeds a certain threshold, then the name is examined as a possible candidate for fluxing.

For any name that is flagged as a candidate for fluxing, we collect relevant data for each of the IP addresses to which the name has resolved, such as Autonomous System Number (ASN), country code (CC), and network (determined either from ASN data and other external sources, or heuristically). In addition, we separately monitor the DNS data to learn if IP addresses belong to Internet service providers (ISPs), educational institutions, cloud service providers, and other well-known network entities. Based on the collected data related with the name, we filter the candidate names. For example,

Table 1 Features used in DGA detection. (ISO: International Organization for Standardization.)

Feature	*Result*	*Description*
stemCount	number	The number of "." separated stems (labels)
ccTLD	binary	The TLD is an ISO country code
extension	binary	TLD with file extension (.sh, .py, .pl, ...)
consonantRun	binary	The 2TLD stem contains a sequence of 4 or more consonants
vowelRun	binary	The 2TLD stem contains a sequence of 4 or more vowels
numericRun	binary	The 2TLD stem has a sequence of 4 or more digits
allDigits	binary	The 2TLD stem is all numeric digits
allVowels	binary	The 2TLD stem is all vowels
allConsonants	binary	The 2TLD stem is all consonants
triGram	number	Bucketized triGram score (250 buckets)
biGram	number	Bucketized biGram score (25 buckets)
length	number	The length of the 2TLD
lengthG	number	$\log_2$ of length
prefixLength	number	The length of the stems to the left of the 2TLD
ispunyCode	binary	Is the 2TLD stem punycode encoded?
dictWords	language	The language for any words found in the 2TLD stem
allWords	language	The language if all of the 2TLD stem consists of dictionary words

a fluxing name must have IP addresses in at least two of multiple networks, multiple ASNs or multiple CCs, or IP addresses belonging to an ISP or an educational institution.

In order to focus on detecting "new" fluxing names within the network, we exclude all domain names that have been observed prior to n days before the analytics date, based on a second-top-level domain (2TLD, e.g., example.com, example.co.uk), i.e., a 2TLD-based whitelist. This also allows us to effectively reduce the number of FQDNs to be tracked.

4.2. Implementation of fast fluxing analytic

The fast fluxing analytic is implemented using two different platforms, one based on the MapReduce processing model and another based on FCCE.

4.2.1. MapReduce

We design the fast fluxing analytics using a MapReduce processing model in Apache Hadoop as follows. In the Map phase, the domain name-to-IP address mappings are extracted. The key for the output is the domain name, and the value is the associated IP address: $\langle name, IP \rangle$. A historical 2TLD-based whitelist is also applied to sort out "new" fluxing candidate names. In the Reduce phase, the list of IP addresses associated with the name is aggregated based on the name as the key for the output of the Map phase. Then, the names which have more unique IP addresses than a threshold are examined as described in Section 4.1.

4.2.2. FCCE

FCCE allows us to operate the fast fluxing analytics in either a near real-time mode or an off-line batch mode. For the near real-time mode, the analytics subscribe to the domain-name-to-IP address mappings that are forwarded as being extracted from the live DNS data. In contrast, for the off-line batch mode, historical queries are made to FCCE to retrieve all of the domain-name-to-IP mappings for the certain date range. In addition, a historical query is made to retrieve all of the 2TLD domains that have been seen within the enterprise, which is used for excluding the names that are not "new." The aggregated name/IP data is then processed as described in Section 4.1.

5. DGA analytics

5.1. Identifying DGA instances from failed lookups

To detect potential DGA activity, the information about the number of failed lookups (i.e., NXDOMAIN) for unique 2TLDs is collected for every client. When the number of NXDOMAIN names exceeds a set threshold T_{NX} for a particular client, the names associated with these queries are examined.

The first step is to extract a set of features for each of the names. The feature set consists of a total of 43 different syntactic features derived from the name looked up; in **Table 1**, we list a selection of features used by our analytics. These features are then used to create a binary feature vector for each name. The vectors are used as the basis for creating clusters of

the names to assess syntactic similarity. To measure the distance between two vectors, we employ the Jaccard distance metric.

Because the number of clusters is unknown a priori, an unsupervised clustering algorithm (e.g., DBSCAN [11]), which divides the input space into clusters having similar density properties, is used. DBSCAN requires two parameters, *eps* to assess the density and *minPts* (a minimum cluster size). We set *minPts* to the value used for the NXDOMAIN count threshold (T_{NX}). The *eps* parameter is set to a fractional amount of the average distance between all vectors (if *eps* is not specified). Clustering is applied recursively to the resulting clusters with each attempt lowering the value of *eps* to $0.85 \times eps$. This is done to "tease apart" multiple distinct *DGA instances*, which may be associated with a client. This is particularly important when the client observed is a forwarding name server or a client is infected by multiple pieces of malware. Every cluster having a sufficient size (the configured threshold for NXDOMAINs), and having a sufficient number of shared features $(\geq \max(5, 0.30 \times totalFeatureCount))$, is considered as a candidate pool of failed names from a DGA instance.

5.2. Expanding to identify resolved names

For a candidate pool of names of a DGA instance of a client, the next step is to find one or more domain names that is resolved (i.e., non-NXDOMAIN query results). We take all of the successful lookups performed by the client within a given time frame and compute the same feature vector for these names described in Section 5.1. We then cluster these vectors individually with each of the candidate pools. If a name's vector is clustered with a candidate pool, we perform two sanity checks on the result: (1) if all of the candidate pool names are within a single TLD, we ensure that the resolved name is as well, and (2) if all of the names within the pool contain dictionary words, then the candidate word must be as well. If the name passes these tests, then we consider it a suspected *resolved DGA name* of the DGA instance.

5.3. Discussion

The DNS data often contains records originated from a forwarding name server. A challenge with the analytics is that there can be a large number of end-point clients behind the forwarding name server. Because we aggregate records based on the originating IP address, all of the data for the clients behind the forwarding name server appear to originate from the same IP and are aggregated. This data can contain names from multiple different DGA algorithms, which must be "teased apart." In order to achieve this, our analytics initially perform clustering with large *eps* and recursively runs with smaller *eps* until the clusters stabilize. This allows us to distinguish the various DGA algorithms which may exist within the data.

5.4. Confidence scoring

Once a resolved name has been identified as a potential part of a DGA instance's pool, we compute a confidence score for that name. Given that from resolved names, additional data may be extracted (such as shared IP addresses, name servers, or clients), we calculate confidence scores for each of those separately and eventually aggregate into the resulting scores.

Common IP address—One means of computing a confidence score of a candidate domain name, d, is by identifying additional names that share the IP address(es) with d. For each name sharing the IP, we cluster it with the pool of failed lookups the candidate name d clustered with. The number of these names that did cluster (Y_d) and did not (N_d) are counted. A confidence score C of the candidate domain name d is computed using the following formula:

$$C(d) = 1.0 - \left(1.0 - \frac{Y_d}{Y_d + N_d}\right)^{[\log_2(Y_d+1)]}$$

Common Name Server—Similar to the common IP address confidence score, a score may also be established for shared name servers.

Common Clients—Analogously, we consider shared clients: Y_d is set to the count of clients that resolved the candidate name and also queried some fraction of the failed lookups in the pool. N_d is set to the count of clients that resolved the candidate name, but did not query any of the failed lookups. Again, the same formula is used.

Neural Network—The three confidence scores above used additional data associated with the resolved name to determine a confidence score. Returning to only using the syntactic properties of the names, a fourth confidence score is computed using a neural network. Because the original clustering algorithm had no means to weight the features, the formed cluster may be incorrect. A neural network, when appropriately trained, will learn the weightings. To train the neural network, we need ground-truth datasets. For the malicious domain set, a subset of the NXDOMAINs is used. For the benign set, a subset of the Alexa top domains (http://www.alexa.com/) is used. Additional subsets from each are also used for verifying the training. Once training is completed, the candidate name is classified by the neural network, giving a value of [0.0, 1.0]. However, because the training phase is non-deterministic, the result can vary over time (if the name is reanalyzed). To mitigate such impact, the training/verification/check is repeated multiple times

Table 2 Weights used for aggregating the confidence scores.

Confidence score	*Weight*
Common IP address	20%
Common name server	20%
Common client	25%
Neural network	35%

(for a limited time). In order to obtain a more stable score, the average value of the scores is used as the final score.

Aggregated Confidence Score—To calculate an aggregated confidence score, a weighted average is computed using the weight configuration listed in **Table 2**. The weights are heuristically chosen to reflect the reliability of the scoring; for example, it is not uncommon that many similar names share an IP address (e.g., shared hosting); thus, we weigh the confidence score of the common IP address lower. In the event that a score is not present (e.g., if no other names share the IP, there will be no shared IP score), the weights are adjusted upward.

5.5. Implementation

We have implemented DGA detection on two platforms. One implementation is based on a MapReduce environment and another based on FCCE.

5.5.1. MapReduce

The programming model for MapReduce complicates the overall implementation of the DGA detection. The initial phase of identifying candidate names is straightforward. All resolved and NXDOMAIN names queried by a client are extracted from the input data during the Map phase. The output is keyed by the client, and the output is the resolved or NXDOMAIN name, with a tag indicating which type the name was. In the Reduce phase, each input record consists of all the resolved and NXDOMAIN names queried by a client. The names are analyzed and clustered as described in Section 5.1.

Computing confidence scores for the flagged names is complex in a pure MapReduce environment. We currently do not have the neural network method implemented for MapReduce. However, for the other three scores, the same general steps are required, exemplified in the following subsections.

Common IP Addresses—The original Reduce phase creates a work file. Each record contains a candidate name along with the NXDOMAIN names that it clustered with. The common IP scoring analytics read this at initialization. The DNS data is then reprocessed. For any name from the work file, any IP addresses that are found in the DNS data are extracted. The output of the Map phase is keyed by the name. The value is the IP address. The Reduce phase creates a new work file for the next stage (each stage is a separate job). The next stage again loads in a work file at initialization. Each record is an IP address along with the name that resolved to that IP address, as well as the NXDOMAIN names that clustered with it. The Map phase again reprocesses the DNS, and for any name found in the DNS data that resolves to one of these IP addresses, an output record is generated. The output is keyed by the original resolved name and contains the newly discovered name, the IP address, and the NXDOMAIN names. In the Reduce phase, the newly discovered names are analyzed and an attempt to cluster them with the NXDOMAIN names is made. The result is written out to a results file containing the name, the score, the NXDOMAIN names, the resolved IP addresses and the newly clustered names.

Common Name Server—The process for computing the confidence score using a common name server is identical to that used by Common IP Address. However, instead of using IP addresses to find related names, name server information from NS records is used.

Common Clients—The process for computing the confidence score using common clients is different from that used by Common IP Address and common Name Server, although it also requires multiple stages. All of the resolved and NXDOMAIN names in the work file are used to identify clients which queried one or more of the resolved and/or NXDOMAIN names. The output of the Map phase is keyed by the candidate DGA name and the value is the client ID and name queried. The Reduce phase examines the overlap between clients which resolved the candidate name and those which also queried some subset of the NXDOMAIN names. The confidence score is computed from the overlap.

Aggregate Score—The data from the results files is processed outside of the MapReduce environment, and the aggregate score is computed. However, because the MapReduce implementation currently does not have the neural network confidence score, the weights are adjusted to reflect the absence.

5.5.2. FCCE

Because FCCE provides both a subscription service and a high-speed historic query system, the analytic for DGA can be implemented in a very lightweight and efficient manner. For online, near-real-time monitoring, a subscription to the stream of NXDOMAIN and client tuples is used to count the number of unique NXDOMAIN 2TLDs that each client queries. For historical analytics, queries are performed to retrieve historical data. In either case, to avoid the large resources required for aggregating of the NXDOMAIN 2TLDs in large-scale deployments, HyperLogLog counters [12, 13] are used to generate a

Table 3 Data sources and volumes.

	Average Daily Volume	*DNS message rate*		*Feature creation rate*	
		Mean/sec	Max/sec	Mean/sec	Max/sec
Tap 1	139 GB	8,000	24,500	11,400	194,000
Tap 2	55 GB	3,100	8,200	5,500	103,00
Tap 3	10 GB	1,300	11,000	650	15,000
Total	204 GB	12,400	43,700	17,550	312,000

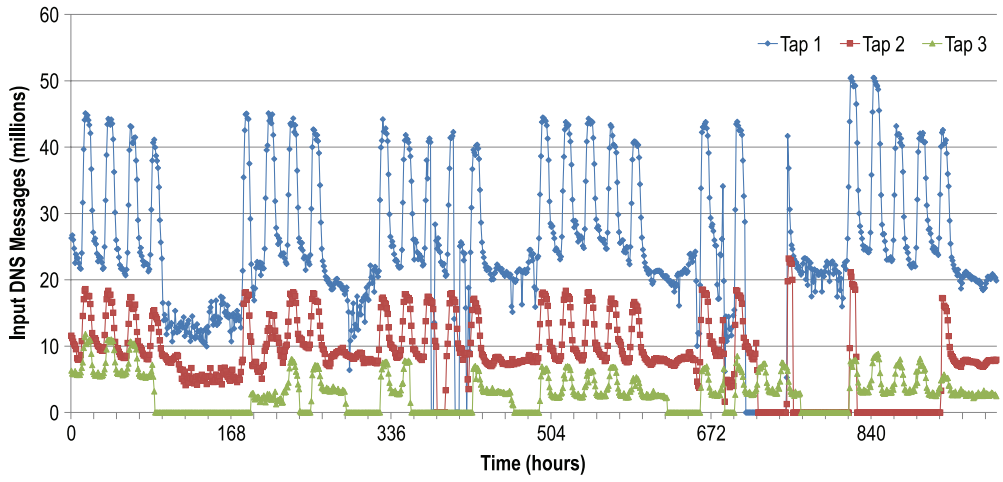

Figure 4

Number of incoming DNS messages per hour for 3 taps.

probabilistic count. When the probabilistic count for a particular client exceeds a threshold, a historic query is performed to retrieve the NXDOMAIN names and resolved names for the client. This data is then analyzed as described in Section 5.1.

For names that are identified as candidate DGA resolved names, the confidence scores are computed by performing historical queries to obtain the needed data.

6. Results

In this section, we provide an overview of the data sources and discuss the detection results generated by our fast fluxing analytics and DGA analytics.

6.1. Data sources

We evaluate the proposed analytics against DNS traffic collected by monitoring DNS messages from and to the main DNS resolvers over a 50-day period from September 1 to October 20, 2015. The DNS data are received as PCAP (packet capture) encapsulated raw data packets from 3 primary taps (i.e., vantage points).

Table 3 summarizes for each tap (1) the volume of DNS messages we received every day, (2) the number of DNS messages per second seen in the raw traffic, and (3) the rate at which the features are extracted from the DNS messages. In total, 44.7 billion DNS messages have been collected; the hourly incoming message rate for all 3 taps is illustrated in **Figure 4**.

On average, we collected around 204 GB of DNS traffic every day across all three taps. Among them, tap 1 has the largest traffic volume accounting for more than half (139 GB) of the total volume. The average incoming rate of DNS messages is 12,400 messages per second, with the peak rate almost quadruple at approximately 43,700 messages per second. Due to the distributed processing capabilities of FCCE engine, our analytics are able to process this large amount of traffic with very small latency and continuously extract features from the raw DNS messages to detect potential fast fluxing and name generation behaviors. The mean and burst feature creation rate is 14,550 per second and 312,000 per second. The discrepancies regarding mean feature per sec

Table 4 Example fast-fluxing domain names found in the dataset.

Domain name	*Time frame*	*IP addresses*	*ASNs*	*Countries*	*Confirmed*
ns2.flyopenvz.ru.	2015/10/05 – 2015/10/15 (10 days)	133	80	16	yes
731pro.pw.	2015/10/05 – 2015/10/23 (18 days)	180	101	14	yes
a.chikgarde.at.	2015/09/03 – 2015/10/26 (53 days)	30	26	11	yes
a.newdrty.org.	2015/09/04 – 2015/09/04 (1 day)	13	12	5	yes

and average DNS messages per sec concern the number of unique queries within an input stream as well as the number of features (related to the number of resource records) within those messages.

6.2. Detection results

In this section, we present the experimental results of our fast-fluxing and DGA analytics.

6.2.1. Fast-fluxing domain names

In this section, we present the experimental results of our fast-fluxing analytics. The input for the fast-fluxing analytics is 4.5 million (DNS name, IP address) pairs every day. Among them, approximately 200,000 DNS names are associated with multiple resolved IP addresses (i.e., a lookup returns more than one IP address), and, after filtering names with the 2TLD-based whitelist ($n = 30$ days), the remaining set is selected as candidates for further analysis. Our analytics flags 10 candidate fluxing names per day on average. Over the entire monitoring period, the analytics flagged in total 53 unique flux domains from 28 different 2TLDs. We manually verified all the flagged domains against external threat intelligence such as Virustotal and Spamhaus. We were able to confirm that 43 of them are fast fluxing domains (81.3% precision), many of which are used for spam-related activities. On the other hand, 4 flagged domains are false positives (7.5%). Further investigation showed that they are either newly created content distribution networks (CDNs) or crowd-sourced domains (such as pool.ntp.org) with very similar behaviors to fast fluxing domains. There were also 5 domains (9.4%) for which we were not able to find sufficient evidence to determine whether they are indeed fast-fluxing domains.

Table 4 shows several examples of confirmed fast-fluxing domains detected in our dataset. As we can see in the table, these domains exhibit typical fast fluxing behavior, i.e. resolving to a large number of geographically dispersed IP addresses. For example, 731pro.pw was associated with 180 different IP addresses spreading across 101 different Autonomous Systems and 14 countries during 18 days of our monitoring time. Evidence showed that it was involved in post-infection communication of a malware program distributed by the nuclear exploit kit.

6.2.2. DGA detection

We ran the DGA detection over the entire trace with at least $T_{NX} = 10$ failed lookups per client and clustering parameters $minPts = 5$ and $eps = 0.25$. In total, we observed about 250,000 distinct client addresses. On average, they queried 5.5 million unique NXDOMAIN names every day from 43,000 unique 2TLDs. On the other hand, they also queried about 7.3 million unique names that can be successfully resolved every day. These unique DNS names are from 888,000 unique 2TLDs. On average, each client queried 75 NXDOMAIN and 260 resolved domains, respectively. However, the distribution is heavily skewed. In fact, the median number of NXDOMAIN and resolved domains queried by the clients are only 15 and 75. The top client, however, queried as many as 118,000 NXDOMAINs and 87,000 resolved names. The analytics reports on average 19 candidate resolved DGA names per day. From the resolved DNS names, we traced back the client machines that were likely infected by a botnet and used these DNS queries as the C&C communication channel. **Figure 5** shows the number of unique IP addresses of candidate DGA clients (i.e., the number of NXDOMAIN 2TLDs exceeded the threshold) for each day during our entire monitoring period. As we can see from the figure, the number of candidate DGA clients shows a diurnal pattern. During a typical weekday, there are more than 60 DGA client machines querying various generated DNS domains, and the number drops to 20 to 30 during the weekend.

The DGA analytics achieves very high detection accuracy. Over the entire monitoring period, the analytics detected 21 unique *resolved* DGA domains with a weighted score of 0.75 or higher, among which 20 are verified to be malicious (95.2%). The remaining one domain is likely to be true positive as well, but we were not able to gather sufficient evidence. Notice that these 21 are "resolved" domains, meaning that they were active (for instance,

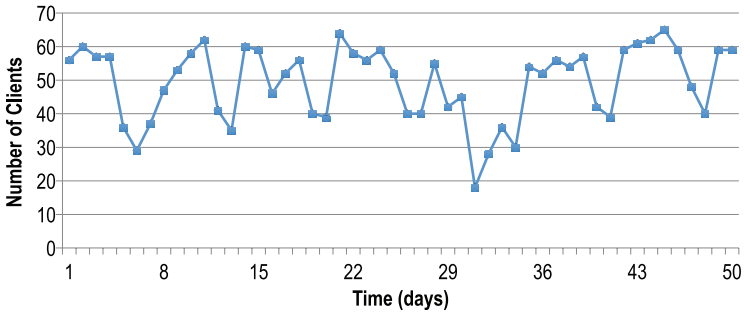

Figure 5

Number of unique IP addresses of candidate DGA clients for each day.

used as C&C servers for certain botnets) when they were "caught" by our analytics. Each of these domains was also clustered with a large number of NXDOMAINs. On average, 1,137 NXDOMAINs were associated with these active DGA domain names and the largest DGA cluster we have observed is "speulay.com," which contained 3,735 different domain names. **Table 5** presents a sample DGA name detected by our analytics.

In the weighted score range of 0.50 to 0.75, 180 names were identified as potential DGA names. Of the 180, a total of 31 were determined to be false positives, giving a false-positive rate of 17% for weighted scores in this range. Thus, while keeping only the names from the 0.75-to-1.0 range gives a near zero false-positive rate, a significant number of flagged names (83%) are in the 0.50-to-1.0 range and would be considered false negatives if names are clipped when the weighted score is less than 0.75.

The false positive rate rises quickly for names with weighted scores in the range 0.25 to 0.50. Of 56 candidate names, 43 were false positives, for a false-positive rate of 77%. For the remaining names, for which the weighted score is less than 0.25, there were 23 names identified, with all of them being false positives.

7. Limitations and future work

The DNS analytics approach presented in the paper is empirically proved effective and scalable. Yet, thus far our design and implementation have focused on detecting FF-generating and DGA-generating domain names by analyzing visible DNS query and response logs. The complexity of many environments may easily obscure our observations on DNS traffic. However, other data sources exist in these environments, and these can be used to enhance results by correlating the results of the DNS analytics with these data sources. This work thus can be enhanced in a number of directions that are worth further investigations.

Intermediate forwarding name servers—In DNS analytics, we have assumed direct observations over all the DNS traffic (including query, response, and client information). In many deployments (e.g., large-scale enterprise networks), this assumption may not hold for cost and scalability reasons. For example, the DNS infrastructure of a large-scale network is often organized hierarchically. A DNS lookup query issued by a client will be first processed by its local DNS server. For efficiency purposes, a caching-and-forwarding mechanism is often enabled. Concretely, the local server stores previously queried domains and their results (i.e., either A-type resource records or NXDOMAIN), with each record valid for a certain pre-defined time-to-live (TTL). Given an incoming DNS lookup query, the server will first attempt to answer it using its cached results. Only when a miss occurs, the query will be forwarded to an upper-level DNS server, which eventually connects to the ISPs' DNS servers. Due to management and operational costs, it is not uncommon that DNS traffic is visible and collectable only at certain vantage points. Thus, a critical challenge is how to apply DNS kinetics analysis over such highly aggregated, pre-filtered DNS traffic.

Botnet threat remediation—A plethora of techniques (including the ones in this paper) have since been proposed to detect FF-generating and DGA-generating domain names in DNS lookups, including analyzing algorithmic patterns of domains, reverse-engineering malware instances, clustering NXDOMAINs in DNS lookups, and even directly capturing C&C traffic. Nevertheless, the studies on how to make use of detected botnet-relevant domains to remediate practical botnet threats in large-scale networks are still fairly limited. At a first glance, this may seem a rather simple problem with trivial solutions, including: a) capturing C&C channels and tracing back to infected devices and b) detecting botnet-domains and vetting DNS behavior of each individual machine to identify positive matches. Unfortunately, directly applying

Table 5 Example DGA name found in the dataset.

Name	*exyndvwy.info.*
IP address	95.211.230.75
Clustered NXDOMAIN names	safeqlez.info., xyzzpmyl.info., kkhqdhjm.info., qbawpzmq.info., wsgftbuy.info., xyzzpmyl.info., ganloscd.info., niktwium.info., wfodxdxr.info., ozffyibz.info., ganloscd.info., safeqlez.info., wsgftbuy.info., jhzdjjrc.info., kkhqdhjm.info., wfodxdxr.info., qbawpzmq.info., suqqwijg.info., obnklcbf.info., kiuxdzyz.info., ozffyibz.info., niktwium.info., jhzdjjrc.info., hndcibcr.info., obnklcbf.info.
Common IP address score	0.95
Clustered Names Sharing IP address	aydzixef.info., dvnzfltn.info., ifpudfcx.info., ipzhzbaz.info., kzudxhbt.info., okrfidzj.info.
Common Name Server score	0.98
Clustered Names Sharing Name Server	aydzixef.info., dvnzfltn.info., ekuwvano.info., ifpudfcx.info., ipzhzbaz.info., kzudxhbt.info., okrfidzj.info.
Shared client score	1.00
Neural network score	0.87
Weighted score	0.94

either naïve solution faces major challenges in practice. First, the captured C&C may be valid only for a short time span, as the bots may quickly roll over to relocated C&C domains, which results in the extremely low coverage of solution a). Second, in large networks, hundreds of thousands of machines are often geographically distributed and probably managed by different entities. Applying solution b) exhaustively to every part of the networks can be prohibitively expensive in terms of operational costs. In face of rapidly emerging botnet threats, with limited resources or accesses, this may considerably delay addressing the threats in a timely manner. It is therefore of great interest to first quickly navigate the threat landscape of the networks and to prioritize the remediation efforts.

Correlation against other data—In deployment environments with other monitoring data available, our DNS kinetics approach can be significantly enhanced by correlating the results of the DNS analysis with these additional data sources. For example, network traffic logs (e.g., firewall logs, web proxy logs, NetFlow records) can be correlated with findings to assess whether IP addresses resolved for fast-fluxing or domain-fluxing domain names were contacted by infected devices; as a consequence, intelligence about the network port, protocol, and application usage can be gathered. Moreover, since the results of DNS lookups are typically cached on an end point, only the initial lookup can be monitored with DNS logs. In order to track the actual communication patterns over time between an end point and a C&C server, the network traffic logs need to be inspected. Separately, not all devices' DNS traffic may be collected (e.g., due to a device's configuration of DNS servers, data sampling, and network hand-over); therefore, the network traffic logs can be used as an additional basis to identify communication of devices with IP addresses (e.g., NetFlow logs, and firewall logs) or host names (e.g., proxy logs) of C&C servers detected by the DNS kinetics.

8. Related work

The importance of analyzing DNS traffic has been recognized widely in the past [3, 14, 15]. Many previous works studied the malicious uses of the public DNS service [14, 16–20]. The Honeynet Project first reported the fast-fluxing technique actively being used in the wild [21]. Holz et al. [14] performed an empirical study of this technique. Mitigating the fast-flux technique is relatively easy once the malicious domain is identified. Because of this, botnets started employing domain-flux techniques, which are relatively difficult to detect and take down. Stone-Gross et al. [3] were able to hijack Torpig

botnet C&C communications and performed the first detailed study of the DGA-based domain-flux technique.

Holz et al. [14] have proposed a technique to detect fast-flux domain names based on three features. Our approach extends their approach with additional features (including country codes and network classification). Several approaches were proposed to automatically detect DGA domains by using the domain's static attributes such as the distribution of alphanumeric characters and linguistic features [20, 22], as well as the dynamic attributes such as NXDOMAIN replies and query patterns [16, 23]. Many recent systems combined both types of attributes from various vantage points, including the domain registration process, to improve detection [17, 18, 24–26]. For example, Pleiades [17] and EXPOSURE [18] use passive DNS data to build features from TTL, DNS response, linguistic-based, and other attributes. Hao et al. [26] explored the time-of-registration features by observing the domain registration process of the .com TLD over five months. Recent works on parked domains [24, 25] studied their monetization mechanisms, where Vissers et al. [25] proposed automatic detection of such park domains using DNS and other features such as malicious redirections. In addition to identifying DGA domains from NXDOMAIN replies, our approach attempts to identify related lookups that resolved successfully (and thus used for C&C communication) and are thus not considered by existing work.

Although several approaches were studied for combating malicious uses of DNS service, to the best of our knowledge, there is no single system that combines those approaches and provides a scalable solution. Our proposed system is complementary to those previous works and consolidates many of the previously known features and provides a scalable solution that can be deployed to large enterprise networks.

9. Conclusion

The Domain Name System (DNS) has been increasingly misused by miscreants to maintain stealthy, takedown-resistant communication channels with infected hosts. The two commonly seen techniques are Fast-fluxing and Domain-fluxing, where the first technique varies the IP addresses, while the second technique varies the domain names that an infected host connects to over time. Such constant variation of IP addresses and domain names makes it difficult for traditional network defenses to detect and block malicious connections. In this work, we proposed scalable algorithms for detecting fast-fluxing and domain-fluxing domains by analyzing DNS logs and describe their implementation in two big data processing models. We evaluate our algorithms and implementations using a large dataset of real-world DNS logs containing 44.7 billion DNS events. Our algorithms were able to identify fast-fluxing and domain-fluxing resolved domains with precisions of 81.3% and 95.2%, respectively. Because of these algorithms and their implementation, (enterprise) network operators are able to reliably identify and take down botnet and malware infested machines in their network.

[**]Trademark, service mark, or registered trademark of Apache Foundation in the United States, other countries, or both.

References

1. S. Shevochenko, *Srizbi's Domain Calculator*, 2008. [Online]. Available: http://blog.threatexpert.com/2008/11/srizbis-domain-calculator.html
2. SonicALERT, *Ranbyus Banking Trojan, Cousin of Zbot*, 2015. [Online]. Available: https://www.mysonicwall.com/sonicalert/searchresults.aspx?ev=article&id=693
3. 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 *Proc. 16th ACM Conf. CCS*, 2009, pp. 635–647.
4. F. Leder and T. Werner, *Know Your Enemy: Containing Conficker, To Tame a Malware*, 2009. [Online]. Available: http://www.honeynet.org/files/KYE-Conficker.pdf
5. P. Royal, *On the Kraken and Bobax Botnets*, 2008. [Online]. Available: https://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf
6. Malcovery Security, "*GameOver Zeus Mutates, Launches Attacks*, 2014. [Online]. Available: http://blog.malcovery.com/breaking-gameover-zeus-returns
7. S. Shevochenko, *Domain Name Generator for Murofet*, 2010. [Online]. Available: http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html
8. TrendMicro, *NECURS: The Malware That Breaks Your Security*, 2014. [Online]. Available: http://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/3133/necurs-the-malware-that-breaks-your-security
9. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Operating Syst. Des. Implement.*, 2004, vol. 6, pp. 1–13.
10. D. L. Schales, X. Hu, J. Jang, R. Sailer, M. P. Stoecklin, and T. Wang, "FCCE: Highly scalable distributed feature collection and correlation engine for low latency big data analytics," in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 1316–1327.
11. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.
12. O. G. P. Flajolet, É. Fusy, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. AOFA*, 2007, pp. 127–146.
13. S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. EDBT Conf.*, 2013, pp. 683–692.
14. T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *Proc. Symp. NDSS*, 2008, pp. 1–12.
15. S. Marchal and T. Engel, "Large scale DNS analysis," in *Proc. 6th IFIP WG 6.6 Int. AIMS Conf. Depend. Netw. Services*, 2012, pp. 151–154.
16. M. Antonakakis, R. Perdisci, W. Lee, N. V. Ii, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy," in *Proc. USENIX*, 2011, pp. 1–16.
17. M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *Proc. USENIX*, 2012, pp. 491–506.

18. L. Bilge, E. Kirda, C. Kruegel, M. Balduzzi, and S. Antipolis, "EXPOSURE: Finding malicious domains using passive DNS analysis," in *Proc. Symp. NDSS*, 2011, pp. 1–17.
19. R. Villamarin-Salomon and J. C. Brustoloni, "Identifying botnets using anomaly detection techniques applied to DNS traffic," in *Proc. IEEE CCNC*, 2008, pp. 476–481.
20. S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proc. ACM SIGCOMM IMC*, 2010, pp. 48–61.
21 T. H. Project, *Know Your Enemy: Fast-Flux Service Networks*. [Online]. Available: http://www.honeynet.org/papers/ff
22. S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-based Botnet tracking and intelligence," in *Proc. 11st Int. Conf. DIMVA*, 2014, pp. 192–211.
23. M. Thomas and A. Mohaisen, "Kindred domains: detecting and clustering botnet domains using DNS traffic," in *Proc. Int. Conf. WWW Companion*, 2014, pp. 1–29.
24. S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang, "Understanding the dark side of domain parking," in *Proc. 23rd USENIX Security*, 2014, pp. 207–222.
25. T. Vissers, W. Joosen, and N. Nikiforakis, "Parking sensors: Analyzing and detecting parked domains," in *Proc. NDSS*, 2015, pp. 1–14.
26. S. Hao, M. Thomas, V. Paxson, N. Feamster, C. Kreibich, C. Grier, and S. Hollenbeck, "Understanding the domain registration behavior of spammers," in *Proc. ACM SIGCOMM IMC*, 2013, pp. 63–76.

Received October 31, 2015; accepted for publication November 26, 2015

Douglas L. Schales *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (schales@us.ibm.com)*. Mr. Schales is a Senior Technical Staff Member in the Security Services Group (formerly Global Security Analysis Lab) at the IBM T. J. Watson Research Center. He received his master's degree in computer science from Louisiana Tech University in 1990. His focus is primarily in cybersecurity, with an emphasis on vulnerabilities and vulnerability analysis.

Jiyong Jang *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jjang@us.ibm.com)*. Dr. Jang is a Research Scientist in the Security Services group (formerly Global Security Analysis Lab) at the IBM Thomas J. Watson Research Center. He received his Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University in 2013. His research interests include most areas of computer security, with an emphasis on software and network security. His current research focuses on big data security analytics and its applications to malware analysis, network security, and Web security.

Ting Wang *IBM Security Division, CTO Security Intelligence, Fredericton, NB E3C 1B2 Canada (ting@cse.lehigh.edu)*. Dr. Wang is currently an Assistant Professor in the Computer Science and Engineering Department at
Lehigh University. Before joining Lehigh, he was a Research Staff Member at IBM Thomas J. Watson Research Center. He received a Ph.D. degree from Georgia Institute of Technology in 2011. His research interests include data mining, machine learning, and privacy and security. His current research focuses on designing privacy-preserving and secure systems, quantitative analysis of privacy and security in emerging computing systems, and theoretical foundations of computational privacy.

Xin Hu *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jjang@us.ibm.com)*. Dr. Hu is a Research Scientist in the Security Services Group (formerly Global Security Analysis Lab) at the IBM T. J. Watson Research Center. He received a Ph.D. degree in computer science and engineering from University of Michigan, Ann Arbor, in 2013. His research focuses primarily on cybersecurity, with an emphasis on big data security analytics and network security.

Dhilung Kirat *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (dkirat@us.ibm.com)*. Dr. Kirat is a Research Scientist in the Security Services Group (formerly Global Security Analysis Lab) at the IBM T. J. Watson Research Center. He received a Ph.D. degree in computer science from University of California, Santa Barbara, in 2015. His research interests revolve around areas of computer security, in particular malware analysis and security analytics.

Ben Wuest *IBM Security Division, CTO Security Intelligence, Fredericton, NB E3C 1B2 Canada (benwuest@ca.ibm.com)*. Mr. Wuest is a Senior Technical Staff Member and CTO Security Intelligence in the IBM Security Division. He holds a B.Sc. C.S. degree (1997) and an M.Sc. Eng. degree (2008) from the University of New Brunswick, Canada. His primary focus is the technical leadership of the Security Intelligence Platform at IBM.

Marc Ph. Stoecklin *IBM Research Division, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (mpstoeck@us.ibm.com)*. Dr. Stoecklin is the Manager of the Security Services Group (formerly Global Security Analysis Lab), where he works on cyber security analytics with a particular focus on network, device, and industrial control system security, as well as big data analytics and visualization. He is an expert in behavior modeling and anomaly analytics based on machine learning and data mining techniques. Dr. Stoecklin holds an M.Sc. degree (2007) and Ph.D. degree (2011) in computer, communication, and information sciences from École Polytechnique Fédérale de Lausanne (EPFL). Dr. Stoecklin is a member of the Institute of Electrical and Electronics Engineers.