Leveraging client-side DNS failure patterns to identify malicious behaviors

Leveraging client-side DNS failure patterns to identify malicious behaviors

A natural idea is to run multiple detection algorithms in parallel, each designed to identify a specific family of failure patterns. However, simultaneously applying these algorithms on the same DNS failure stream poses several challenging issues. In particular, among multiple *distinct* but *overlapping* clusters, each identified by a different algorithm, which one should be selected as the "true" suspicious cluster(s)?

To address these issues, we develop a framework that leverages *evolutionary learning* to automatically detect diverse failure patterns. We let the temporary clusters identified by the various clustering algorithms evolve and compete over time. We associate each cluster with a quality measure or "fitness" score determined by a number of factors such as the cluster cohesiveness, size, temporal closeness, etc., and update the measure as each cluster evolves over time. The clusters with good fitness scores "survive" and are selected as the output of our framework.

Our main contributions are as follows: (i) We perform a systematic study of DNS failures using large ISP datasets. (ii) We discover various failure patterns that are diverse and stealthy, which shows that relying on single detection algorithms may fail at detecting more complex attacks that abuse DNS failures. (iii) We propose a comprehensive framework that leverages *evolutionary learning* to detect diverse clusters of suspicious DNS failures, using both *syntactic* and *temporal* features. (iv) Our framework detects DNS failure clusters on a per-client basis and different from previous work [5], [6], does not hinge on the existence of multiple clients infected by the same malware. Therefore, we can operate even on a single host or in a small edge network. (v) Evaluation results demonstrate that our framework effectively detects diverse, stealthy DNS failure patterns, where at least 97% clients with suspicious DNS behaviors are detected with over 81% precision.

II. Related Work

There has been a large amount of work on analyzing DNS traffic for cyber security. Several papers have focused on studying successfully resolved, malicious domain names. Notos [7] and EXPOSURE [8] are DNS reputation systems that employ a variety of features to identify potentially malicious domains. Kopis [9] monitors DNS query patterns from the vantage point of authoritative nameservers and TLD (Top Level Domain) servers. Using spamtraps, it was shown that the initial DNS behavior of malicious domains differs from that of benign ones [10]. Gao et al. [4] empirically re-examined the global DNS behavior, and proposed to detect malicious domain groups using only temporal correlation in DNS queries.

Detecting domain-flux botnets has also been a popular topic. A clustering algorithm was developed to identify fast-flux domains based on the similarity among successfully resolved IPs [11], while several *syntactic* metrics in the successfully resolved domain names were used to identify groups of DGA generated, mostly random-looking domains [3]. Our work is complementary to these work as we leverage DNS failures to identify malicious clients and expand the coverage of domain name syntactic patterns beyond random-looking cases [3], [8].

TABLE I
Overview of the two DNS datasets.

Items	Aug2011	Apr2012
Total A:IN DNS sessions w/ eTLDs	12,816,150	24,039,008
Failed DNS sessions	335,588 (2.62%)	516,047 (2.15%)
Queried names	892,255	1,113,073
Clients	12,272	15,911

Another group of studies has focused on analyzing DNS failures [5], [6], [12], [13]. In particular, Pleiades [6] collects all the failed queries from clients in an ISP and clusters them based on syntactic features (mostly targetting random domains) and coocurrence of failures across multiple clients. In contrast, we show that there are diverse and stealthy failure patterns that require multiple techniques for syntactic analysis (for non random-looking domains), as well as temporal correlation. In addition, our approach does not hinge on the existence of multiple clients infected by the same malware, and thus can be used on a single host or in a small edge network with a limited vantage point. Finally, our approach is completely unsupervised as we do not require models of known bot DNS behavior ahead of time.

III. Datasets Overview and Preliminary Analysis

We use network traces collected at a vantage point within a large European ISP. The monitored network covers over 15000 unique (and mostly residential) client IP addresses. We collected traces in August 2011 and April 2012, each spanning 24 hours. All incoming and outgoing TCP connections and UDP flows of the network were captured. We extract all the DNS queries and responses from the traces and produce two 24-hour long DNS datasets. We refer to the two datasets as `Aug2011` and `Apr2012`, respectively. We also use the relevant TCP/UDP flows for investigating certain suspicious or malicious activities uncovered in the DNS datasets. Finally, we identify suspicious clients as those that have resolved domains present in popular blacklists [14]–[16] or that have a bad reputation from Web-of-Trust (WoT) [17] or that have generated TCP/UDP flows labeled as malicious by a commercial IDS. These client set will be the basis of our analysis.

A. Data Preprocessing and DNS Failures

We match DNS queries with the corresponding responses and refer to the resulting query-response pair as a *DNS session*. We focus only on *A:IN* type DNS sessions (namely, queries for the IPv4 address using an Internet domain name) since they are the most predominant in our dataset, and remove other types of sessions. Although our framework is agnostic to the DNS session type (e.g. TXT, etc.), we leave this to future work. In addition, our analysis mainly focuses on failed DNS sessions, whose RCODE (server response) is 3 (Name Error) or 2 (Server Failure). In some cases we also analyze successful DNS sessions (RCODE = 0) when the domain name queried can be correlated with suspicious failed DNS sessions. Table I summarizes key statistics of the two datasets.

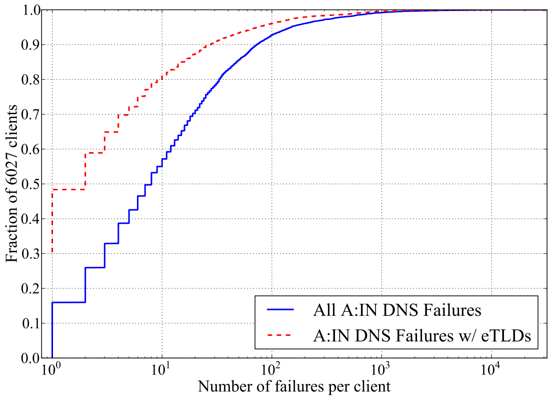

Fig. 1. DNS failures of clients with at least one DNS failure.

B. Manual Categorization of DNS Failures

We aim at finding malicious behavior out of DNS failures traffic. Our motivation is twofold: (i) to systematically analyze the various patterns of malicious behavior found in our dataset and (ii) to create ground truth to evaluate the system proposed in Section V.

We begin by removing those cases that are benign. In particular, we eliminate all DNS failures that do not contain an *effective top-level domain* (*eTLD*)[1] and thus cannot be resolved by a public resolver. They are typically a product of typos, misconfigurations or malformed DNS queries generated by some browsers. Fig. 1 plots the CDF of the DNS failures generated by all residential clients in `Aug2011`. The solid blue line represents all DNS failures while the dashed red line shows failures to domain names containing eTLDs. We observe that after removing these benign DNS failures, we still have over 1,200 clients that generate more than 10 failures. As we see later, many of these clients turned out to be malicious.

In order to separate *suspicious* DNS failures from benign ones, we conduct an analysis of the failed domain names. We first identify clients that satisfy any of the following conditions: (i) at least one failed domain name is present in a blacklist, (ii) at least one successful DNS query is flagged by a commercial IDS, (iii) generating more than 100 DNS failures, or (iv) generating failed queries containing at least five distinct eSLDs, as this correlates with higher chances of a client being malicious in our dataset.

Once we have distinguished such clients, we look for syntactic failure patterns as well as successful DNS queries related to those suspicious patterns. In addition, we look for any suspicious activity after a DNS query, such as contacted webpages or IP addresses that appear in blacklists and Web of Trust. We also search WHOIS databases to check the dates and owners of registered domain names. Newly registered domain names may raise more suspicion than names that have existed for a long time. In addition, we look for suspicious-looking domain names that were alive in our traces but are no longer registered, which provides a hint of suspiciousness.

TABLE II
CAT-R EXAMPLES.

Conficker	Torpig	Simda-E
arjynor.net	bfejhvfe.com	fobiqab.su
bdjcuenagtq.ws	dibxfhci.com	qedihyp.su
clrkknzxm.cc	gwubvjue.com	kyjiluj.su
zumxknrjcy.net	xxjgwbwd.com	xuxukyd.su

We find that the most common *benign* failure cases are: (i) clients that generate a small number of DNS failures over the 24-hour period, mostly caused by typos and temporary network issues or misconfigurations, (ii) clients that generate a large number of failures to a few distinct domain names, caused by network misconfigurations (e.g., running applications with disconnected VPN connections), and (iii) clients that generate a large number of distinct failures to a few eSLDs. These cases are mostly caused by DNS overloading [19] (i.e., applications leverage DNS for their own purposes, such as mail servers communicating through DNS with anti-spam services for spam filter) and DNS suffix appending (often configured in machines to handle non-FQDNs).

Finally, we create the *grey* category for those DNS failures that we cannot categorize as benign or malicious. In this category we have clients with DNS failures to domain names used for P2P activities (e.g. `piratebay.com` and domain names containing keywords such as `publisher` and `torrent`). We also find many failed names containing adult-themed keywords that belong to websites that are no longer in business. Finally, there are two clients for which the failed domain names are related to mail servers (i.e., the names contain strings `mx` or `mail`).

IV. SUSPICIOUS FAILURE PATTERNS

We look in-depth at suspicious DNS failure patterns and classify them based on their most distinct syntactic patterns. We discuss the challenges posed by the diversity and stealthiness of these suspicious failure patterns in automatic detection, which motivates our evolutionary framework in Section V.

Highly Random Domain Name Failure Patterns (Cat-*R*): This is the most dominant failure pattern category in our dataset. It is often associated with various families of domain-flux malware that generate random domains (as defined in Section V-B) to query in a short period of time. As only few of the domains might be registered by bot handlers, the malware activities lead to a large number of query failures. The most dominant domain-flux malware belongs to the `Conficker` family [2], and the second most dominant to the `Torpig` family [1]. Other random domain failure patterns are produced by malware of the `Sality` family, `Simda-E`, etc. We provide a few samples of the failed domains of this group in Table II.

"Partially Random" Domain Name Failure Patterns with *Limited Character Set* (Cat-*C*): This category is the second most popular and contains several sub-categories. They share some characteristics with Cat-R in that the failed domain

[1] An *effective* top-level domain (eTLD) separates the responsibility of the registrant from registrars. For example, `.com`, and `.co.uk` are eTLDs, in that domains such as `foo.com`, and `bar.co.uk` can be directly registered. Here `foo` or `bar` is termed an *effective* second-level domain (eSLD). We use a list of public suffixes [18] to extract eTLDs and eSLDs.

TABLE III
CAT-C EXAMPLES.

C.1 Letter-digit mixture	*C.2* Hexadecimal scan
89s7dgf78ger367gs6.com	02e4f47239ec4228bdf59872697367ce.com
9sd7fg87sgdfg7sfd.co.cc (s)	11de14271e4c4d66beaecdac7de4295a.com
s87fggsdfuyvsdvtftds.ar	fdf298c0b6894524ba373f230ef843ba.com
s89d7fgh37rsh7f8.eu	*C.3* Anchored letter w/ random digits
s89dfhshdf8hsdf.cw	a65255b65255.com
sd9f08hsdfybs76dft.cc	a686435b686435.com
sd98f7ghsdfysdg6f.co.gp (s)	a7098373b7098373.com

TABLE IV
CAT-M EXAMPLES.

M.1 Base-string mutation	*M.2* Evolving mutation
oogle.xx (s)	housemetaset.metase.xx (s)
googl.xx (s)	housemetaset.meta.xx
giogle.xx	housemetase.xx (s)
go9gle.xx	housemeta.xx (s)
gokgle.xx	house.xx (benign)
goohle.xx (s)	ho.xx

TABLE V
CAT-S EXAMPLES.

S.1 Fixed prefix + varying letters	*S.2* Fixed prefix + varying digits
searchodd.org	lonelyday01.in (s)
searchangle.org	lonelyday04.in
searchcommon.org	ginsburg03.in
searchhissing.org	ginsburg04.in (s)
findthousand.org	domain510005.com
findexpensive.org	domain490002.com (s)
clickbrake.org	agng78sagdfdkjdtwa195.com
clickafraid.org	agng78sagdfdkjdtwa655.com

names look partially random, except for a key difference: the letters or numbers come from a *limited* character set. As in Cat-R, some of the failure patterns occur in multiple bursts spreading over time. This seems to indicate a malware bot is running in the background sporadically or activated by certain user actions to generate bursts of DNS queries. Compared with Cat-R, the failure patterns generate fewer numbers of failures per burst, which makes them stealthier and harder to detect. We describe three sub-categories with representative examples.

(C.1) Random mixture of letters and digits: Examples are shown in Table III where both failed and successful query samples from several instances of the same failure pattern are presented (the successful queries are marked by "(s)"). The eSLDs in the queried domains consist of a mixture of letters and digits of various lengths, which at first glance looks random. A closer look reveals that the letters come from a limited set of characters, {`b`, `d`, `e`, `f`, `g`, `h`, `i`, `n`, `r`, `s`, `u`, `v`, `w`, `y`}, and the digits from `3` to `9`, with some letters (e.g., {`d`, `f`, `g`, `s`}) and digits (e.g., {`6`, `7`, `8`}) appearing more frequent than others. In addition, a diverse set of eTLDs (e.g., `com`, `net`, `ar`, `by`, `cc`, `ve`, `asia`, `co.cc`, `co.uk`, `web.gg`, `int.nf`, `pro.vg`) are involved.

(C.2) Hexadecimal scan: Table III shows examples of a failure pattern where any failed eSLD consists of a long string of characters from the hexadecimal representation of integers.

(C.3) Anchored letters with random digits: In this failure pattern, all eSLDs start with the letter "`a`" followed by a string of random digits of varying lengths, and then the letter "`b`" followed by the same string of random digits.

***Mutated* String Domain Name Failure Patterns (Cat-*M*):** This category groups various subtly different patterns, in which the failed eSLDs look similar to each other. They are *mutated* from a common string ($M.1$) or transformed from one set to another by changing (e.g., inserting, deleting, or substituting) one or two characters at a time ($M.2$). Note that for $M.2$, we have anonymized the country code and slightly changed the domain name itself, to avoid revealing the country of origin of our ISP. Table IV presents examples of such failure patterns.

For $M.1$, we observed a burst of more than 100 queries for domains mutated from the string `google`, with a couple of legitimate queries to `google.xx` and `gogle.xx` intermixed in between (`gogle.xx` is registered by Google, presumably to prevent typosquatting). Besides these legitimate queries, a significant portion is successful, each returning a single IP address. However, the returned IPs belong to a variety of ISPs. Many of these IPs have been blacklisted, and the remaining ones are deemed highly suspicious (e.g. they are in the same subnet as blacklisted IPs.). The queries were issued in a short span of 48 seconds, with the interval between consecutive queries varying from 0 to 2400 ms.

In the case of $M.2$, the suspicious queries follow the exact order (from top to bottom) of those shown in Table IV. The sequence starts with a two-part string separated by "." that is mutated from a legitimate website and gradually evolves to a shorter string by deleting one character at a time. The suspicious queries were issued in a short span of less than a minute, with the interval between consecutive queries ranging from 30 to 800 ms.

A key feature that distinguishes failed domains of this category from the previous categories is that they do not look "random." In fact, when examined in isolation, each appears as if it was generated by a typo made by a human user. Hence the failure patterns in this category are far stealthier. In addition, most failure patterns in this category were observed only once per client. Compared with the previous two categories, the failed queries are often associated with a significant number of successful queries that share the same distinct mutation/evolution patterns.

***Substring* Domain Name Failure Patterns (Cat-*S*):** This category contains a large number of distinct failure patterns, each containing only a small number of failed (as well as successful) query domains that share a common substring. Table V shows examples in the two sub-categories: *(S.1) fixed prefix with varying letters* and *(S.2) fixed prefix with varying digits*. These failure patterns are the least noisy and the most stealthy. All the examples in *S.1* and the first three examples in *S.2* have been generated by clients infected with Trojan malware such as `Troj/Agent-VUD` and `Troj/DwnLdr-JVY`.

Summary: The above categories are the major failure patterns we have uncovered. There are a few minor failures that are difficult to classify. For simplicity, we have placed them in one of the four categories that they are closest to.

Fig. 2 shows the cumulative distribution of the failure cluster sizes for all categories in `Aug2011` (`Apr2012` shows similar

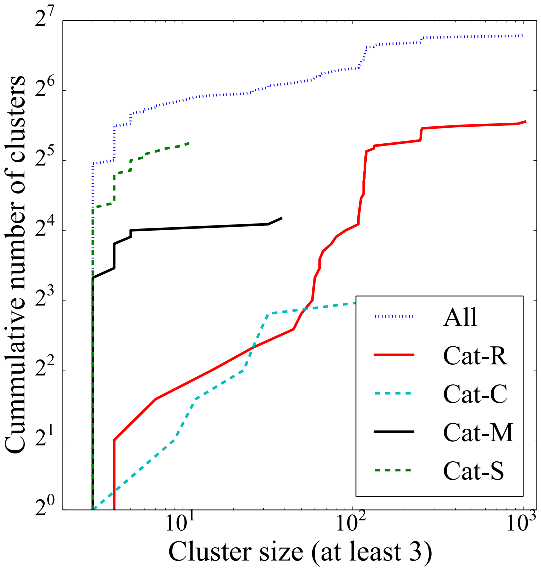

Fig. 2. Distribution of per-category labeled cluster sizes.

distributions). An (x,y) point shows that y clusters are less than or equal in size to x. Not surprisingly, we see that most failure clusters in Cat-R are large, with more than half of them containing nearly 100 or more failed domain names. Those in Cat-C have moderately large sizes, and those in Cat-M have generally smaller sizes. While the number is large, the sizes of Cat-S clusters are the smallest, with the largest containing around 10 failed domain names.

We provide in Table VII a quantitative analysis of each category. In the "Ground Truth" column, we summarize the results obtained through our systematic analysis and detailed manual inspection for both `Aug2011` and `Apr2012` datasets. We list the number of clients that exhibited any of the detected suspicious DNS query behaviors and a break-down of the number of clients that generated suspicious failure patterns belonging to each four major categories. We also list the total number of DNS failure patterns that we identified as well as a break-down of the clusters in each category.

V. Comprehensive Detection Framework

We present an evolutionary learning framework for automatically detecting and classifying diverse DNS failure patterns. The basic idea is to run multiple detection algorithms in parallel, with each designed to create a set of temporary clusters of a specific failure pattern, and let the clusters of different patterns evolve over time. The evolution is based on a fitness score that is a quality measure dynamically assigned to each cluster according to its cohesiveness, size, temporal closeness, etc. The score is updated as each cluster evolves. For example, clusters that expand over time by absorbing new failed query strings and clusters that re-occur repeatedly, would improve their fitness scores and increase their odds of survival. Once its fitness score passes a threshold, the cluster would be elevated as a *detected* failure cluster as part of the framework output. In contrast, those clusters that do not grow over time or subsumed by other clusters, would not survive.

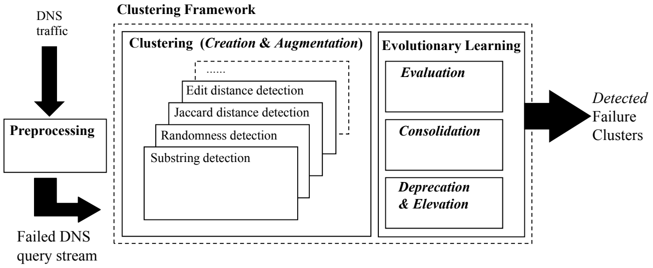

Fig. 3. System architecture.

We first describe the evolutionary learning framework that is the basis of our system. We then describe the various detection algorithms we use to form the clusters of suspicious DNS query failures.

A. System Architecture and Evolutionary Learning

Fig. 3 depicts the key system components of our framework. The system operates on a stream of DNS queries generated by each individual client. The stream of DNS queries is first passed through the *Preprocessing* module that filters failed queries without eTLDs, failures resulting from DNS overloading or misconfigurations, or other benign failures (as discussed in Section III). This module then extracts the *eSLD* strings contained in the failed DNS queries.

The *Cluster Creation & Augmentation* module runs multiple clustering algorithms in parallel, and periodically generates potential clusters of failed domain names. The "brain" of the system lies in the *Evolutionary Learning* module that consists of three submodules. The *Evaluation* module assigns each cluster with a fitness score updated over time. The *Cluster Consolidation* and the *Cluster Deprecation & Elevation* modules use the cluster fitness to merge closely related clusters, subsume smaller clusters into larger ones, deprecate temporary clusters that do not grow sufficiently or re-occur over time, and elevate those with fitness scores exceeding the threshold to the *detected* status.

Cluster Creation and Augmentation: This module operates in two modes. (i) *Cluster creation:* given a collection of failed *eSLD* strings, it extracts possible termporary clusters by running different clustering algorithms in parallel. (ii) *Cluster augmentation:* given a new failed domain, the module adds it to an existing cluster if applicable.

Clusters created in the first mode evolve over time. The operations of the system are illustrated in Fig. 4 from the perspective of the "lifecycle" of temporary clusters. Each failed eSLD is fed to all clustering algorithms in parallel to evaluate whether it can be included in one of existing temporary clusters $C_1, \ldots, C_m$. If affirmative, it is included in the corresponding cluster(s) – the clusters thus expand in size. If all existing clusters reject, it is put into the pool of unclustered queries.

For efficiency, instead of operating on one string at a time, we apply the cluster augmentation operation every Δt (e.g., 1 minute) to all the failed eSLDs generated within Δt. In

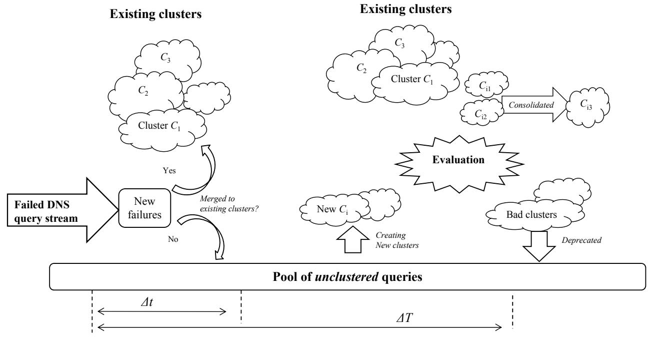

Fig. 4. Illustration of operations.

contrast, the cluster creation mode runs various clustering algorithms in parallel to identify and create new clusters from the pool of unclustered failed eSLDs. It is invoked periodically every ΔT (e.g., 1 hour), or when the pool size exceeds a threshold.

Evolutionary Learning - Cluster Fitness Score and Cluster Evaluation: We assign each temporary cluster, C_i, a fitness score based on four metrics of cluster quality: (i) cluster cohesiveness, v_i, (ii) cluster size, z_i, (iii) temporal closeness, a_i, and (iv) the number of occurrences, n_i. The definition of cohesiveness depends on the specific clustering algorithm and is discussed in Section V-B. We normalize v_i to a value within $[0, 1]$, with 1 being the most cohesive. The cluster size is self-explanatory. The temporal closeness, a_i, is the average inter-arrival time of consecutive failed queries in the cluster. The number of occurrences, $n_i \geq 1$, records the number of times the same/similar cluster occurs.

We define f_i, the fitness score of cluster C_i, as a function of these four metrics: $f_i = F(v_i, z_i, a_i, n_i)$. Intuitively, high scores are assigned to clusters with tight cohesiveness and short inter-arrival times, and those that grow consistently over time or re-occur. We use the following heuristic function F to map v_i, z_i, a_i, and n_i to a fitness score in $[0, 1]$:

$$f_i = \frac{1}{1 + e^{-\alpha_i z_i}} \cdot [1 - (1 - v_i)^{n_i}] \cdot e^{-\beta \max\{a_i - a_0, 0\}}. \quad (1)$$

The formula follows simple intuitions. All three multiplicative terms fall within $[0, 1]$. The first term is the logistic function, widely used in many applications (e.g., for modeling population growth), and it grows monotonically from 0.5 to 1 as $\alpha_i z_i > 0$ increases. α_i $(0 < \alpha_i \leq 1)$ differentiates types of clusters and controls the effect of the cluster size on the fitness score (see Section V-B). Its value, along with the minimum cluster size threshold, are algorithm specific. Based on our experiments, we selected the values reported in Table VI.

When $n_i = 1$, the second multiplicative term is simply v_i; when $n_i > 1$, the second term assigns a "new" cohesiveness metric to the cluster that is an increasing function of n_i. It states that even when a cluster is less cohesive, if such a cluster re-occurs over time, it is rewarded with a higher fitness score. The third multiplicative term is an exponential decaying function, where parameter a_0 is a constant (say, $a_0 = 1$ sec). If $a_i \leq a_0$, the third term is 1; otherwise, it decreases as $a_i - a_0$ increases. The scaling constant β controls the rate of the decay; in our implementation, we set $\beta = 0.1$ based on our experiments.

TABLE VI
CLUSTERING ALGORITHM SPECIFIC PARAMETERS.

Parameter	Rand	Jacc	Edit	Subs
α_i	$0.1v_i$	$0.2v_i$	$0.4v_i$	$0.8v_i$
min threshold z_i	10	8	5	4

Cluster Consolidation, Deprecation and Elevation: As the temporary clusters evolve over time, we compute and update the metrics, v_i, z_i, a_i and n_i, as well as the overall fitness score f_i, associated with each cluster. On every ΔT period, we evaluate the existing clusters to determine whether some clusters can be consolidated, deprecated or elevated.

Given two clusters C_i and C_j identified by the same clustering algorithm, we evaluate whether they are sufficiently similar based on the metric used by each clustering algorithm. We then decide whether to merge the two clusters as a result. Two similar clusters occurring in different times are considered as two occurrences of the same cluster. We consolidate two clusters only when the resulting new cluster has a higher fitness score than both existing clusters. Such consolidation starts with the two clusters with the smallest sizes, and proceeds recursively until no more consolidations are possible.

Temporary clusters that have not grown in size since the last update (say, at time t_0) are penalized, and the fitness score at time t is reduced to $f_i(t) = e^{-\gamma(t-t_0)} f_i(t_0)$ where γ is a decay factor. We choose $\gamma = 0.01$ (with t in units of minutes) that showed good results in our tests. When its fitness score falls below a threshold, the cluster is removed from further consideration. We also remove eSLD strings in the pool of unclustered queries every three ΔT periods. Furthermore, we compare the clusters identified by different clustering algorithms, to make a joint deprecation decision. Suppose we have a larger cluster C_1 identified as one type, and a smaller cluster C_2 of a different type. We deprecate C_2 if it passes both an overlap test $|C_1 \cap C_2|/|C_2| \geq 0.9$ and a fitness score test $f_1 > f_2$. Such deprecation decision is crucial for removing poor-quality or redundant clusters.

Finally, temporary clusters whose fitness scores exceed certain threshold (e.g., $f_i > 0.75$) for a long period of time (e.g., $5\Delta T$) are elevated to detected failure patterns. We record these clusters, together with all the metrics, as part of the output of the system.

B. Clustering/Detection Algorithms

We describe the clustering algorithms developed in conjunction with our framework. In each algorithm, we implement hierarchical agglomerative clustering with different metrics to identify similarities of string properties. We remark that the presented algorithms are meant as example algorithms to illustrate how diverse failure patterns can be detected. While the efficacy of our framework hinges on the accuracy of the algorithms employed, the framework in itself is general as it allows additional – and more sophisticated – algorithms to be installed as “plug-&-play” detection modules.

Randomness Based Detection (Rand): The key syntactic feature of DNS failures in Cat-R is that all failed eSLDs of various lengths are random-looking. To determine if an eSLD is random, we leverage the fact that in any language, characters do not appear randomly or independently after each other. For example, the letter ‘u’ occurs very frequently after ‘q’ in English. We create a dictionary of domains crawled from dmoz.org, a multilingual directory of web links, from which we compute the conditional probabilities of one character occurring after another. To test the randomness of an eSLD, s, we compute $\lambda(s)$, the log-likelihood of the character sequence in s comparing with that of generating the same string from a uniform distribution. We then convert $\lambda(s)$ to a *randomness score* within $[0, 1]$, with a value closer to 1 indicating a string closer to random. The cohesiveness v_i is the average randomness score of all strings in cluster C_i.

Jaccard Similarity Based Detection (Jacc): The key common feature of the failed queries in Cat-C is that they are permutations of a limited character set. Jaccard similarity based detection algorithm is very effective in clustering such failed queries. Given two strings s_1 and s_2 on two character sets A_1 and A_2, the Jaccard similarity is defined as $J(s_1, s_2) = |A_1 \cap A_2|/|A_1 \cup A_2|$. We use a modified Jaccard similarity measure that takes the length of the strings into account by multiplying $J(s_1, s_2)$ by a weight $w(s_1, s_2)$ that is a logistic function of $\min\{|s_1|, |s_2|\}$. In other words, longer strings with the same (standard) Jaccard similarity measure are considered more similar than shorter strings. We use this modified Jaccard similarity metric to perform clustering, employing a method similar to [11]. The cohesiveness metric v_i for a Jaccard similarity cluster C_i is computed as the average of the pair-wise (modified) Jaccard similarity of all strings in the cluster.

Edit Distance Based Detection (Edit): In order to detect the mutated string failure patterns in Cat-M, we apply the Levenshtein edit distance [20] that measures the dissimilarity of the two strings. The edit distance of two strings, $Edit(s_1, s_2)$, is the minimum number of single-character edit operations (i.e., insertion, deletion, substitution) required to transform s_1 to s_2. We use a normalized version [21] that takes string lengths into account and produces a value within [0,1]. We apply this distance metric to group strings with short edit distances occurring together within a relatively short time span, and declare them an edit-distance cluster when the number of such strings exceeds the threshold. The cohesiveness metric v_i for an edit distance cluster C_i is computed as one minus the average pair-wise normalized edit distance over all pairs.

Substring Based Detection (Subs): Given s_1 and s_2, we apply the standard substring extraction algorithm to find the (longest) common substring — for instance, a common substring can be extracted as a by-product of computing the edit distance of the two strings. When applying this method to an ensemble of strings, we extract the common substring for all pairs of strings in the ensemble. We sort the extracted substrings based on the frequency they occur in the set of extracted pair-wise substrings and discard those with a length of less than four. We select the most frequently occurring substrings using a cut-off threshold based on the empirical frequency distribution. For each selected substring, we group the strings containing the substring and declare it a substring cluster if the cluster size $z_i \geq 4$. Let h denote the length of the substring for a substring cluster C_i. The cohesiveness metric v_i is defined using the logistic function, $v_i = 1/(1+e^{-(h-3)})$.

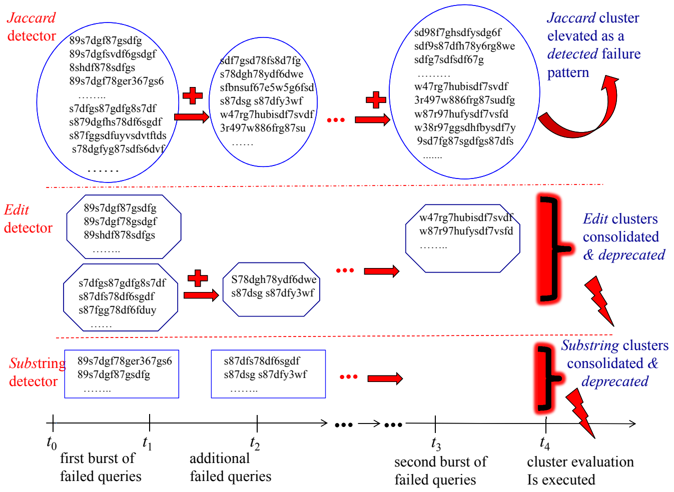

Fig. 5. Framework illustration using an example.

C. An Illustration of Framework in Action

To illustrate our evolutionary framework, we use a semi-random looking failure pattern $C.1$, shown in Table III as an example. We run four detection algorithms, *Rand*, *Jacc*, *Edit* and *Subs* in parallel. Since the failed eSLD strings have a limited character set (e.g., d, f, g, s, 7, 8) and bi-grams (e.g., dg, gf, gs) occur more frequently than others, many of the failed domain names in $C.1$ fail our randomness test. Therefore, we focus primarily on the operations of Jacc, Edit and Subs, as illustrated in Fig. 5.

When applying the three detectors to the first burst of 20+ failed eSLD strings that occur in a span of 5 minutes, the *Jacc* detector groups all failed queries into a single temporary cluster. Both the *Edit* and *Subs* detectors group subsets of these strings into multiple clusters at time instant t_1 in Fig. 5 (only one or two are shown). In the next few minutes, more failed queries of the same pattern continue to occur. The *Jacc* detector includes them in the same cluster and the cluster quickly expands. On the other hand, the two *Edit* clusters grow slowly, each adding one or two strings, while the *Subs* clusters do not grow; instead, new *Subs* clusters are created. Hence, only the *Jacc* cluster increases its fitness score significantly.

When the next burst of the same failure patterns occur in 25 minutes, the *Jacc* detector recognizes these as a new instance of the previously detected cluster, includes them in the same *Jacc* cluster, and increments the occurrence count n_i. In contrast, both the *Edit* and *Subs* detectors include some of them into the existing clusters and create new clusters out of some remaining failed eSLDs. When the *Cluster Evaluation* module is executed, the fitness score of the *Jacc* cluster exceeds the elevation threshold. The *Cluster Consolidation, Deprecation & Elevation* modules are then invoked. The *Jacc* cluster is elevated to the "detected" status and recorded in the output of the system. Various temporary *Edit* and *Subs* clusters that have survived so far are subsumed by the *Jacc* cluster, and thereby removed from the system.

VI. Evaluation

To evaluate the accuracy of our framework in detecting malicious activities, we run our framework on the DNS traffic of each individual client from the same set of clients that we manually labeled in Section III-B. These clients contain at least five unique eSLDs in the DNS failures. There are a total of 802 such clients from `Aug2011` and 1,277 from `Apr2012`. For each client, our framework generates a (possibly empty) set of suspicious clusters, which we term *reported clusters*. The quality of our framework is measured by how these reported clusters match our labeled clusters (or the ground truth), on both the cluster level and the client level.

A. Methodology

Baseline: To quantify the benefits of our framework, we design two baseline systems that we compare against. In (i) the *Standalone mode,* we implement a simple approach where each of the four detection algorithms runs independently (i.e., in isolation) on the whole 24-hour trace, and only consolidate among the clusters reported by the same detector. In (ii) the *Batch mode,* clusters across different detectors are consolidated. The consolidation is based on the same overlapping threshold used in our framework (Section V-A), except that here, we do not conduct fitness test or utilize temporal features.

The baseline comparisons serve two purposes. First, we quantify the benefits of integrating clusters of multiple categories, as opposed to running them in isolation (i.e., the Standalone mode). Second, we evaluate the benefits of our evolutionary framework that manages the lifecycles of all clusters in an online fashion (vs the Batch mode). The results from our framework are in the "Framework" columns of Table VII, whereas the baseline comparison results are presented in the "Standalone" and "Batch" columns.

Next, we illustrate our use of True Positive (TP), False Negative (FN) and False Positive (FP) notations. We focus on Table VII, using the Cat-S row and the "Framework" columns for the `Aug2011` dataset as examples.

Cluster-Level Evaluation: Suppose a client has a set of clusters $LC = \{L_i\}$ manually labeled as Cat-S clusters, and a set of clusters $RC = \{R_j\}$ reported by the Cat-S detector. We want to measure how LC "matches" RC. For simplicity, we use binary classification as follows. For each labeled cluster L_i, we find from RC the R_j^* that has the largest *Jaccard index* w.r.t. L_i, i.e., $J_i^* = \max_j |L_i \cap R_j|/|L_i \cup R_j|$. If $J_i^* \geq 0.5$ (i.e. a reasonable degree of similarity), we match L_i and R_j^* and consider them a *TP cluster* of Cat-S, and remove R_j^* from RC. Otherwise, L_i is considered an *FN cluster* of Cat-S. When all L_i have been processed, the remaining clusters R_j in RC are *FP cluster*s of Cat-S. Summing over all categories for each TP/FN/FP of the same client, we have the overall cluster count. We aggregate the numbers on all clients and produce the statistics.

Client-Level Evaluation: To measure the accuracy of detecting clients with suspicious DNS query behaviors, we cumulate the cluster-level results of each client and produce client-level statistics. If a client has at least one labeled clusters of Cat-S, we label this client as a *suspicious client of Cat-S* (note that one client may have labels of different categories); for example, 31 clients contribute to a total of 38 Cat-S labeled clusters in `Aug2011`. We consider a client as a *TP client* of Cat-S if it contains Cat-S labeled clusters and the Cat-S detector reports *at least one* Cat-S labeled cluster.

An *FN client* of Cat-S contains Cat-S labeled clusters, but no TP clusters of Cat-S. In other words, when a client contains labeled clusters of Cat-S, it is classified as either TP or FN client for this category. On the contrary, when it contains no Cat-S labeled clusters but has at least one falsely reported clusters by the Cat-S detector, it becomes an *FP client* of Cat-S; otherwise it is a *TN client* of this category.

We report TP/FN/FP per category. When calculating the overall client-level statistics for all categories (i.e., the first row), the number is not a simple sum over all categories, as one client may be reported by multiple detectors. In particular, we define an *overall FP client* as an FP client of at least one category, and at the same time, no TP client of any other category. For example, our framework results in zero overall FP clients on the `Aug2011` dataset; although we have three Cat-S FP clients, they are excluded from the overall FP client as they are also TP clients of other categories.

B. Results

The results in Table VII convey the following. First, for both datasets, our framework narrows down a large number of clients to few suspicious ones with very high accuracy. The false negatives are extremely low and false positives are well controlled, on both the client and cluster levels. At the client level, our framework achieves 100% recall, 100% precision on `Aug2011`, and 97% recall, 81% precision on `Apr2012`.

Second, compared with the two baseline schemes, our framework significantly reduces the cluster-level false positives, thanks to evolutionary learning. For example, in `Aug2011`, our framework yields only 11 FP clusters, as opposed to 113 for the Batch and 243 for the Standalone. This result shows the high detection accuracy of our framework.

Our framework has extremely low false negative rates. There are two FN clients/clusters of Cat-R in `Apr2012`. One is a client infected with `Win32/Cutwail.BQ` that generated no syntactically clusterable DNS failures. The other is a client infected by `Simda-E` whose DGA exhibits certain randomness with subtle structure (e.g., `hapydub`, `hanydow`, `halydob`, `hapidyz`, etc.), for which we would need a

TABLE VII

EVALUATION RESULTS OF OUR FRAMEWORK AGAINST TWO BASELINE SCHEMES: BATCH MODE AND STANDALONE MODE.

	Number of Clients											Number of Clusters									
	ALL	Ground Truth	Framework			Batch			Standalone			Ground Truth	Framework			Batch			Standalone		
			TP	FN	FP	TP	FN	FP	TP	FN	FP		TP	FN	FP	TP	FN	FP	TP	FN	FP
Aug2011	802	101	101	0	0	94	7	19	97	4	19	111	111	0	11	103	8	113	107	4	243
Cat-R	n/a	47	47	0	2	47	0	11	47	0	12	47	47	0	2	47	0	11	47	0	12
Cat-C	n/a	8	8	0	0	3	5	12	5	3	25	8	8	0	1	3	5	48	5	3	147
Cat-M	n/a	17	17	0	2	15	2	8	16	1	13	18	18	0	2	15	3	11	17	1	24
Cat-S	n/a	31	31	0	3	31	0	32	31	0	36	38	38	0	6	38	0	43	38	0	60
Apr2012	1227	68	66	2	15	63	5	50	63	5	50	74	72	2	33	66	8	278	68	6	662
Cat-R	n/a	39	37	2	7	37	2	14	37	2	14	39	37	2	8	37	2	15	37	2	15
Cat-C	n/a	5	5	0	6	2	3	4	2	3	19	8	8	0	7	2	6	16	4	4	90
Cat-M	n/a	0	0	0	5	0	0	13	0	0	27	0	0	0	6	0	0	153	0	0	182
Cat-S	n/a	26	26	0	11	26	0	59	26	0	59	27	27	0	12	27	0	94	27	0	375

specific detector different from the ones we deployed. We acknowledge the possibility of under-estimating false negatives, as our labeling was conducted manually (although very carefully) and might have missed some ground truth.

Our framework produces noticeable false positives, in particular on `Apr2012` at the cluster level. It is caused by "over-splitting," where a (reported) cluster splits into two and the smaller part (or even both parts) contributes to the false positives, although it partially overlaps with the labeled cluster. We are working on ways to improve this. Another cause is our stringent definition of false positives: our cluster-level evaluation does not go beyond the detector boundary. For example, if a labeled cluster of Cat-S is falsely reported by a Cat-R detector rather than a Cat-S detector, the labeled cluster becomes a Cat-S FN and the Cat-R detector is considered an FP. Because of these two reasons, the calculated FP rates "under-estimate" the efficacy of our framework.

VII. CONCLUSION

We presented a systematic study of DNS failures using a large ISP datasets. Our findings demonstrate that attackers are employing a variety of disparate domain name patterns for their malicious activities. In addition to random-looking failures generated by domain-flux botnets, we have uncovered many diverse and stealthy DNS failure patterns. We build on these observations to design a comprehensive evolutionary learning framework to detect diverse clusters of suspicious DNS failures. Our framework is developed as a "plug-&-play" system, enabling new detectors to be easily incorporated. We evaluated our framework against a large set of manually labeled malicious clients and clusters, and showed that our framework identifies over 97% manually labeled suspicious clients with at least 81% precision.

We recognize that our framework has a few limitations. First, DNS failure monitoring only provides a "first-order" detector to generate alerts, and in itself is insufficient to confirm malicious activities. We plan to expand our analysis to suspicious successful DNS queries and the corresponding network traffic triggered by them (e.g. HTTP flows). We also note that stealthy malware that generates few DNS failures with little temporal correlation or syntactic patterns, will be difficult for our framework to detect. Addressing these challenges is part of future research.

VIII. ACKNOWLEDGEMENTS

Pengkui Luo and Zhi-Li Zhang were supported in part by NSF grants CNS-1117536, CRI-1305237, CNS-1411636 and DTRA grant HDTRA1-14-1-0040 and DoD ARO MURI Award W911NF-12-1-0385. Part of the research was conducted while Pengkui Luo was a summer intern at Narus Inc.

REFERENCES

[1] B. Stone-Gross *et al.*, "Your Botnet is My Botnet: Analysis of a Botnet Takeover," in *ACM CCS*, 2009.
[2] P. Porras, H. Saidi, and V. Yegneswaran, "A Foray into Conficker's Logic and Rendezvous Points," in *USENIX LEET*, 2009.
[3] S. Yadav *et al.*, "Detecting Algorithmically Generated Malicious Domain Names," in *ACM IMC*, 2010.
[4] Y. Gao *et al.*, "An Empirical Reexamination of Global DNS Behavior," in *ACM SIGCOMM*, 2013.
[5] N. Jiang *et al.*, "Identifying Suspicious Activities through DNS Failure Graph Analysis," in *IEEE ICNP*, 2010, pp. 144–153.
[6] M. Antonakakis *et al.*, "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware," in *USENIX Security*, 2012.
[7] ——, "Building a Dynamic Reputation System for DNS," in *USENIX Security*, 2010.
[8] L. Bilge *et al.*, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *NDSS*, 2011.
[9] M. Antonakakis *et al.*, "Detecting Malware Domains at the Upper DNS Hierarchy," in *USENIX Security*, 2011.
[10] S. Hao, N. Feamster, and R. Pandrangi, "Monitoring the Initial DNS Behavior of Malicious Domains," in *ACM IMC*, 2011.
[11] R. Perdisci *et al.*, "Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces," in *ACSAC*, 2009.
[12] Z. Zhu, V. Yegneswaran, and Y. Chen, "Using Failure Information Analysis to Detect Enterprise Zombies," in *SecureComm*, 2009.
[13] K. Sato *et al.*, "Extending Black Domain Name List by Using Co-occurrence Relation between DNS Queries," in *USENIX LEET*, 2010.
[14] Malware Domains, http://malwaredomains.com.
[15] Malware Domain List, http://malwaredomainlist.com.
[16] PhishTank, http://phishtank.com.
[17] Web of Trust, http://www.mywot.com.
[18] Public Suffix List, http://publicsuffix.org/.
[19] D. Plonka and P. Barford, "Context-aware Clustering of DNS Query Traffic," in *ACM IMC*, 2008.
[20] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics Doklady*, vol. 10, 1966.
[21] Y. Li and B. Liu, "A Normalized Levenshtein Distance Metric," *IEEE Trans. PAMI*, vol. 29, no. 6, June 2007.