Computer Networks 100 (2016) 28–44

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Mining agile DNS traffic using graph analysis for cybercrime detection

Andreas Berger[a,*], Alessandro D'Alconzo[e], Wilfried N. Gansterer[b], Antonio Pescapé[c,d]

[a] Telecommunications Research Center Vienna, Vienna, Austria
[b] University of Vienna, Faculty of Computer Science, Vienna, Austria
[c] Electrical Engineering and Information Technology Department of University of Napoli Federico II, Napoli, Italy
[d] NM2 srl, Napoli, Italy
[e] Austrian Institute of Technology, Vienna, Austria

ARTICLE INFO

Article history:
Received 30 April 2015
Revised 22 December 2015
Accepted 1 February 2016
Available online 15 February 2016

Keywords:
Cybercrime detection
Traffic analysis
DNS
Graph analysis
Network monitoring

ABSTRACT

We consider the analysis of network traffic data for identifying highly agile DNS patterns which are widely considered indicative for cybercrime. In contrast to related approaches, our methodology is capable of explicitly distinguishing between the individual, inherent agility of benign Internet services and criminal sites. Although some benign services use a large number of addresses, they are confined to a subset of IP addresses, due to operational requirements and contractual agreements with certain Content Distribution Networks. We discuss DNSMap, a system which analyzes observed DNS traffic, and continuously learns which FQDNs are hosted on which IP addresses. Any *significant* changes over time are mapped to bipartite graphs, which are then further pruned for cybercrime activity. Graph analysis enables the detection of transitive relations between FQDNs and IPs, and reveals clusters of malicious FQDNs and IP addresses hosting them. We developed a prototype system which is designed for realtime analysis, requires no costly classifier retraining, and no excessive whitelisting. We evaluate our system using large data sets from an ISP with several 100,000 customers, and demonstrate that even moderately agile criminal sites can be detected reliably and almost immediately.

1. Introduction

A recent report by Norton estimates the global annual cost of cybercrime as 113 Billion US$, with 378 Million victims per year [1]. Although a multitude of advanced detection (e.g., malware scanners) and mitigation (e.g., firewalls) techniques are available and the expenses for defense mechanisms are significant, the problem is clearly not under control. This is fundamentally related to the vulnerability of Internet end-users, who fall victim to automated attacks at vast numbers.

For example, users are lured into visiting malicious *exploit sites*, which in turn install malware on their machines (*drive-by-downloads*). This enables criminals not only to steal sensitive data directly from infected machines, and to extort money from the victims, e.g., by threatening them with blocking access to their data [2]. Even worse, remotely controlled, malware-infected machines (i.e., *bots*) serve as all-purpose platforms (i.e., *botnets*), and are, e.g., used for Distributed Denial of Service (DDoS) attacks and Spamming campaigns.

In this paper we aim at detecting malicious websites by monitoring DNS traffic in access networks. We exploit the fact that these sites are required to be stealthy *and* reliably available at the same time. For example, exploit sites need to be well-reachable by the targeted users, to

[*] Corresponding author. Tel.: +43 1505283030.
E-mail address: andreas.berger@alumni.tugraz.at (A. Berger).

http://dx.doi.org/10.1016/j.comnet.2016.02.009
1389-1286/© 2016 Elsevier B.V. All rights reserved.

ensure that each request can result in a possible infection, and therefore translate into yet another resource available to the criminal [3,4]. Similarly, Phishing and Spyware sites are profitable only if victims can reach them. Furthermore, the administration and maintenance of a botnet requires rendezvous points—so-called *Command-and-Control* (C&C) servers—from which the bots regularly download new commands and malware updates. High availability of C&C servers implies tight control of malicious operations, which translates into better botnet utilization, and thus into more revenue.

In order to achieve high availability and resilience against countermeasures, Internet criminals adopt networking strategies similar to Content Distribution Networks (CDNs). Malicious websites use Fully Qualified Domain Names (FQDNs) which often map to multiple IP addresses for redundancy, and these addresses change over time to render access restrictions ineffective (commonly called *Fast-Flux*) [5]. IP addresses are often re-used and host multiple malicious FQDNs. Furthermore, multiple FQDNs often represent the same criminal site, to impede DNS-based detection approaches and avoid FQDN-based blacklisting. Also, FQDNs change quickly over time, often based on specially crafted algorithms that take, e.g., the current time as input to produce a pseudo-random FQDN, known in advance only to the botmaster (Domain Generation Algorithms (DGAs)) [6,7].

Based on the previous definition of Antonakakis et al. [8] we refer to such highly dynamic FQDN-to-IP address mappings as *agile* DNS activity. The main goal of this paper is the reliable detection of such activity in Internet DNS traffic and thereby force criminals to abandon these CDN-like strategies at the cost of less reliable (i.e., less profitable) service operation.

1.1. Motivation and contribution

In contrast to existing work in this field, our system is exclusively based on the *mappings* between FQDNs and IP addresses. We track which FQDNs map to which IP addresses, and thereby establish an understanding of what represents “normal” DNS activity w.r.t. a *specific* FQDN and IP address. The resulting profiles are the basis of our detection approach. Any DNS mapping which involves an FQDN that does not “fit” to these profiles is considered suspicious, and enters the second analysis stage. There, we represent all suspicious mappings in a certain epoch as a bipartite graph, where nodes are FQDNs and IP addresses, respectively, and edges indicate the existence of a suspicious mapping between them. The structural properties of these graphs relate to the agility of the underlying DNS activity. E.g., high-degree IP address nodes host many different FQDNs, while high-degree FQDN nodes map to many different IP addresses. This allows us to apply standard graph analysis techniques to quickly identify malicious sites.

Our work is based on our detailed analysis of DNS traffic [9,10]. We make the following new contributions and propose a complete cybercrime detection system:

(1) We present an analysis system which processes large amounts of DNS traffic data in real time and continuously adapts over time without requiring a retraining phase. In contrast to most other approaches, the system does not require a priori whitelisting that often contains many generic entries as, e.g., $*$.`SUFFIX`. Therefore, our system allows significantly more control about which FQDNs are actually ignored. The set of analysis parameters is small, and can be intuitively tuned for new deployments.

(2) The basis of our approach is the new methodology *DNSMap* for assessing the level of agility of DNS FQDN-to-IP-address mappings. DNSMap tracks observed mappings between FQDNs and IP addresses and stores them efficiently. Note that in contrast to existing work [8,11], we always consider the *entire* FQDN instead of only a suffix. We extend our previous work [10], optimize several parameters, describe a system implementing this concept, and provide new details and experimental results.

(3) On top of DNSMap, we employ graph analysis for analyzing sets of suspicious DNS mappings, in order to address the distributed (CDN-like) nature of malicious networks. We propose a set of graph analysis features, provide a complete description of our detection system, and demonstrate that even conservative settings reveal cybercrime activity in large DNS traffic data sets reliably and efficiently.

(4) Our approach requires significantly less training data than other approaches (2 days in our experiments) and our system is more sensitive to cybercrime activity. Furthermore, it can be easily tuned to reveal different kinds of malware activity, and is therefore not restricted to Fast-Flux detection. Consequently, it is more versatile than previous work, such as [5,12].

Due to confidentiality of network traffic data and the lack of publicly available tools, an objective comparison of the available approaches is typically not possible. Therefore, we provide the entire source code[1] of our analysis tools to enable the verification of our results by others.

The remainder of the paper is structured as follows. We compare our work to the state-of-the-art in Section 2 and discuss the individual components of our system in Section 3. In Section 3.1, we first review the basic concept of DNSMap introduced in [10]. Then, we extend by discussing additional details and presenting several improvements. In Section 3.2, we discuss the construction of a graph from the suspicious mappings which DNSMap produces and propose a number of analysis techniques for exposing group activity patterns. We discuss the system’s parameters in Section 4 and evaluate a prototype implementation using DNS traffic data from a large operator network in Section 5. Finally, we discuss experimental results, the system’s limitations, and possible evasion strategies in Section 6.

2. Related work

Most related work is based on training a classifier using publicly available blacklists, which are compiled, e.g.,

[1] https://github.com/anderasberger/pydnsmap.

from domain names found in Spam emails [11–13]. However, as shown by Kührer et al. [14], the coverage and completeness of blacklists is often limited, as the contents of any given blacklist depend on the analysis method and the data set they are based on. Furthermore, Kührer et al. find that domain names used for cybercrime are often blacklisted only more than 30 days after they were seen first and typically do not contain pseudo-randomly generated domains which are often employed by cybercriminals. Most existing systems use also whitelists to train their classifiers with what is considered known-benign data, as, e.g., the Alexa list[2] of popular domains. This is, however, problematic as well. For example, sub-domains of well-known, whitelisted sites (e.g., dynamic DNS services) are known to be used for criminal activities.[3] The general problem of acquiring high-quality ground truth for network traffic analysis is discussed in detail by Sommer and Paxson in [15]. Therefore, for quick and accurate detection of new malicious domains, our system is designed such that *no classifier training based on white/blacklists is required*.

Antonakakis et al. proposed a dynamic reputation system for domain names, called "Notos" [13]. The system processes DNS query responses from a passive DNS database and extracts a set of 41 features from observed FQDNs and IPs. A labeled training set of both benign and malicious FQDNs is used to derive eight domain classes (five benign, three malicious) by clustering the FQDNs according to their feature vectors. After this initial training, subsequently seen FQDNs are then assigned to one of these classes, according to their features. Ultimately, Notos derives a reputation score for each observed domain name. In a similar spirit, Bilge et al. presented their "EXPOSURE" system [11], which requires fewer features (15) and less training data (1 week), but has no notion of the relations between domains mapping to the same set of IP addresses. Both Notos and EXPOSURE employ the Alexa list for whitelisting popular domains. In contrast, our system uses fewer features, less training data and it requires no retraining and no a priori whitelisting. Furthermore, it does not require any labeled training data at all, which constitutes an important advantage over other approaches. Fundamentally, our system design is closer to signature-based detection approaches where parameters directly encode the specific, yet complex, DNS activity patterns one wants to be alerted about. While this may still yield false positives/negatives, our system *guarantees* to report any such traffic. Detection results can be *intuitively interpreted*, thereby avoiding the typical limitations of techniques based on machine-learning (see [15]).

A number of approaches target specific aspects of malicious DNS activity. The system described in [8] aims at detecting malware at the upper DNS hierarchy, and is therefore restricted to a subset of domain names (e.g., $*$ .com). Yadav et al. discuss the detection of algorithmically generated domain names [6]. However, many malicious sites do not use such names or use improved name generation algorithms which complicate such an analysis (cf. Fig. 10). Furthermore, also benign sites use such domain names [9], and may cause many false positives.

Choi et al. proposed "BotGAD", a system which targets the DNS group activity patterns of the monitored (client) hosts [16]. The approach is based on the idea that malware infected hosts would periodically query the same (malicious) domains. The authors evaluate the approach experimentally and reveal 20 previously unknown botnets. This approach is complementary to the one we present in this paper. Considering the client hosts in the analyzed graph is a possible direction for extending our system.

Hu et al. analyzed the global IP usage patterns of Fast-Flux botnets by conducting measurements from 240 geographically distributed network locations [17]. They mainly aim at differentiating malicious fluxy activity from legitimate CDNs. Therefore, they continuously query a set of 5169 suspicious domains to see how the returned DNS mappings develop over time. Most notably, they find that whereas legitimate CDNs are trying to mostly return an IP address that is geographically close to the querying client, addresses involved in malicious activity are much more uniformly distributed over the world.

The work of Hu et al. is similar to other approaches which exclusively focus on detecting Fast-Flux activity [5,18], and which are therefore narrower in scope than our system. Holz et al. present the first empirical study of Fast-Flux networks and provide an early approach for automated identification [19]. The introduced traffic features are fundamental for many later detection approaches, including the one presented in this paper (see Section 3.2). Most recently, Perdisci et al. proposed "FluxBuster" [12]. Their system performs a sequence of four steps: (i) All information collected about a domain name d in an epoch $\varepsilon = 1$ day is aggregated. (ii) Domains that are unlikely to represent Fast-Flux are removed. More precisely, FluxBuster removes domains with a high average TTL, less than three IP addresses, and a low IP address diversity (i.e., IP addresses which belong to a small number of different /16 networks). (iii) The system finds clusters of domains which share a significant number of IP addresses. (iv) Finally, a supervised statistical classifier algorithm labels domains as flux or non-flux. At the end of an evaluation epoch, a set of 13 features is computed for each cluster. A C4.5 decision tree classifier, trained using a 4 months long training set and tuned with an extra 1 month of data, ultimately decides whether the domains and IPs in a cluster are related to malicious flux services. The authors state that the detection of flux domains can take *up to 30 h*. In contrast, our system can detect malicious activity *immediately*, and is not limited to Fast-Flux. Furthermore, our system is *more sensitive* than FluxBuster (which requires a minimum of 30 IP addresses per domain cluster *per day* to detect flux activity, while we required only 20 *per week* in our experiments), and uses much less training data. At the same time, our approach requires less parameter tuning, fewer features, and no periodic classifier retraining.

[2] www.alexa.com

[3] http://krebsonsecurity.com/2014/07/microsoft-darkens-4mm-sites-in-malware-fight/ .

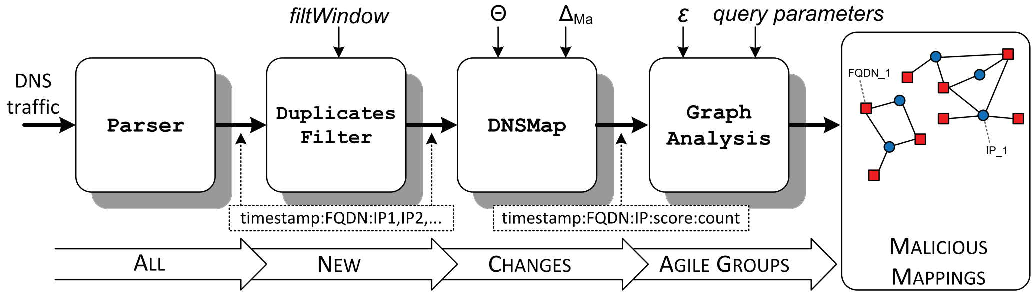


Fig. 1. System overview. We discuss the main components in Section 3 and the parameters in Section 4.

3. System design

The proposed analysis system comprises four main stages, shown in Fig. 1. The *Parser* reads DNS traffic data (from the wire/from a dump file), and further processes DNS NOERROR request responses. Any queries for an FQDN which were answered with one or more IPv4 address records are extracted, and are forwarded to the *Duplicates Filter*. Recall that our approach is based solely on DNS mappings, therefore there is no added value in multiple evaluations of the same mapping within short time. The filter consequently reports each mapping only once, and ignores any further occurrences within a time interval `filtWindow`. The remaining mappings are then processed by *DNSMap*, which in turn outputs a series of *change events*. All events within a certain evaluation epoch ε are then processed by the *Graph Analysis* module which produces the final detection results. DNSMap and the graph analysis component are discussed in the remainder of this section.

3.1. DNSMap—detecting agile DNS

The level of DNS mapping agility of *benign* Internet services varies widely. Websites of small companies are hosted on the same IP addresses for years, while large enterprises host their services on many different IP addresses. Both of them typically use sets of FQDNs which are somewhat similar to each other, following, e.g., the pattern **.example.com*. In contrast, CDNs and hosting providers reuse the same IP address for a large variety of different sites, while in access networks sometimes dynamic (DHCP) addresses host, e.g., private websites using dynamic DNS providers. The level and the type of DNS agility therefore depends on the specific site (i.e., FQDN) and the IP address we are looking at, and is usually limited. Even large companies only use IP addresses in a certain set of networks, though this set may be large. Private customers may use many different IP addresses due to DHCP, but these typically all belong to the same network. Malicious sites have different constraints: they need to change both FQDNs *and* IP addresses often, so to avoid mitigation actions (e.g., blacklisting) and react to new “business” requirements (e.g., a new Phishing campaign which requires a different FQDN).

DNSMap’s main objective consists of providing an adaptive characterization of this agility. Instead of just counting, e.g., the number of IP addresses for a given FQDN [11,13], we infer the real “unusualness” of these DNS mappings from evaluating how normal *the given* FQDN appears for a *specific* IP address. In this section, we derive a system that scales to large volumes of DNS data and enables almost instantaneous detection of unusual, suspicious mappings.

In the following we adopt the established DNS terminology: we consider *mappings* between FQDNs (*domains*) and IP addresses hosting them. We refer to the λth level of an FQDN as λ-LD, e.g., the 1-LD of www.example.org is `org`, the 2-LD is `example`, and the 3-LD is `www`. Note that we use a regularly updated list of public suffixes[4] to identify the 1-LD, i.e., the 1-LD of yahoo.co.uk is `co.uk` and not `uk`.

3.1.1. Measuring FQDN similarity

A fundamental requirement of our system is the ability to assess the similarity of domain names. Therefore, we review our formulation of the *Domain Divergence* (DD) between two FQDNs X and Y [10]. Let X_λ be the λ-LD of X and $|X_\lambda|$ its length (Y_λ, $|Y_\lambda|$ resp.). Let $|X|$ be the number of domain levels in X ($|Y|$ resp.). For each domain level $\lambda > 1$ we first compute a weight w_λ based on (i) the hierarchical “importance” of λ (i.e., more significant levels with lower λ receive more weight), and (ii) related to the length of the currently compared domain level. A dampening constant α controls the rate of decrease of w_λ with increasing λ. Based on our experience, setting $\alpha = 1$ is a good choice, which we use throughout this paper. For each domain level, we further compute a *partial domain divergence* dd_λ between X_λ and Y_λ, using the Levenshtein ratio (LR) which is based on counting the necessary edit operations (i.e., add, delete, replace[5]) for transforming one string into another [20]. More precisely,

$$dd_\lambda = \begin{cases} 1 & \text{if } \lambda > \textsc{Min}(|X|, |Y|) \\ (1 - LR(X_\lambda, Y_\lambda)) \cdot w_\lambda & \text{else} \end{cases} \quad (1)$$

[4] https://wiki.mozilla.org/Public_Suffix_List.

[5] For our experiments, we use a Python implementation of the Levenshtein ratio, which assigns a cost of 2 to the replacement operation, i.e., one “replace” is considered equivalent to one “delete” and one “add”. LR(s1,s2) is then defined as $1 - \frac{\#OP}{|s1|+|s2|}$, where $\#OP$ is the weighted sum of operations, and $|s1|$ and $|s2|$ are the lengths of the two compared strings.

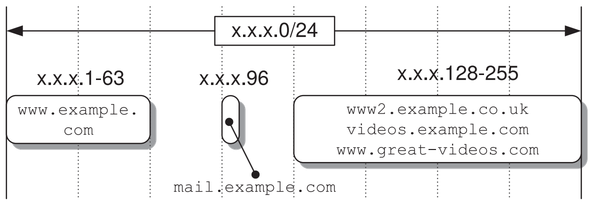


Fig. 2. Illustrative example for the information contained in DNSMap: three IPBlocks holding five FQDNs in total.

where $w_\lambda = l_\lambda \cdot 1/(\alpha + \lambda)$ and $l_\lambda = \textsc{Max}(|X_\lambda|, |Y_\lambda|)$. The *Domain Divergence* DD is then defined as

$$DD := \textsc{Min}\left(\frac{\sum_{\lambda=2}^{\textsc{Max}(|X|,|Y|)} dd_\lambda}{\Omega} + \delta \cdot \beta, 1.0\right) \quad [0, 1] \qquad (2)$$

where $\Omega = \sum_{\lambda=2}^{\textsc{Max}(|X|,|Y|)} w_\lambda$ and δ is 0 when the 1-LDs of X and Y are identical, else $\delta = 1$. Note that, in contrast to the earlier formulation in [10], we assign a fixed penalty β when X and Y have different 1-LDs, which is independent of the LR of the two 1-LDs. This is more intuitive than our previous formulation—all 1-LDs are in fact equally different from each other. We set $\beta = 0.05$ for all following experiments.

3.1.2. Detecting significant DNS changes

The detection of *changes* in DNS mappings requires an up-to-date understanding of the *historic* DNS mappings for a particular range of IP addresses. The basic components for holding this information are *IPBlocks*, which describe *continuous* ranges of IP addresses and the set of FQDNs mapping to them. Fig. 2 shows an (illustrative) example: in the /24-network from IP address x.x.x.0 to x.x.x.255, three IPBlocks are identified. Addresses 1–63 exclusively host www.example.com, address 96 hosts mail.example.com, and addresses 128–255 host Example's UK webserver (www2.example.co.uk), and a video site which is reachable via videos.example.com and www.great-videos.com.

However, keeping track of the FQDNs mapping to an IPBlock is only the first step. Ultimately, we aim at understanding which FQDN *patterns* are used per IPBlock, so to be able to quickly evaluate how much they differ from subsequently seen FQDNs. In general, a particular IP range may be used by several classes of very different FQDNs. Therefore, we cluster the FQDNs in an IPBlock according to their *Domain Divergence*. For each cluster c_k, we derive a *label* L_k, which is based on the level-wise string median [21] of all FQDNs f_i in c_k. By construction, $DD(L_k, f_i) \le \Theta$, where the *domain divergence threshold* $\Theta \in [0, 1]$ is a system parameter. The FQDN of any new DNS mapping is then compared to the cluster labels of the corresponding IPBlock, and is either accepted as "sufficiently similar" (i.e., $\le \Theta$), or requires that a new cluster is created to contain it. Note that this strategy *enforces* that for all FQDNs f_j which map to an IPBlock there exists a cluster for which $DD(L_k, f_j) \le \Theta$. Also note that this is far more efficient than comparing a new FQDN to all FQDNs seen previously at this IPBlock. We refer the reader to [10] for a detailed discussion of the clustering algorithm and the construction of the labels.

The set of IPBlocks as a whole is therefore a summary of "what maps where", and constitutes the foundation of the detection system. A key requirement for our system is the ability to efficiently find the IPBlock which contains a particular IP address. This is accomplished by organizing the IPBlocks in a set of RBTrees[6] [22]. Each leaf node in a tree represents one IPBlock, which is keyed on the starting IP address. The *containing* IPBlock for any IP address can then be found in $\mathcal{O}(\log N)$ time, where N is the number of IPBlocks per tree.

Algorithm 1 AddMapping(*fqdn*,*ip*).

```
1:  ipb ← DNSMap.GetContainingBlock(ip)
2:  if not ipb then                              ▷ IP not seen recently
3:      ipb ← IPBlock()
4:      ipb.AddCluster([fqdn])
5:      return (NEW, 1.0)
6:  end if
7:  if ipb.ContainsFQDN(fqdn) then               ▷ DNS mapping is known
8:      return (OK, 0.0)
9:  end if
10: closestCluster, DD ← ipb.ClosestCluster(fqdn)
11: if DD ≤ Θ then                   ▷ new FQDN, similar to prev. FQDNs
12:     closestCluster.Add(fqdn)
13:     return (OK, DD)
14: end if
15: if ipb.IsFull() then             ▷ this IPBlock reached max. capacity
16:     return (FULL, DD)
17: end if
18: ipb.AddCluster([fqdn])
19: return (NEW, DD)
```

Adding DNS mappings: A new DNS mapping is added by using the AddMapping procedure (shown in Algorithm 1), which returns a status code and a DD score that describes how well it "fits" to previously seen mappings. The individual steps are the following. Given *ip*, we first look up the IPBlock which contains *ip* (line 1). If exactly the same mapping is already stored in DNSMap, we return a zero score, indicating a perfect match (line 8). Else, we find the cluster to which *fqdn* fits best (line 10). If it is not sufficiently similar (w.r.t. Θ) (line 11), and if we can create yet another cluster for this IPBlock[7] (line 15), we consider this mapping to represent a *significant DNS change* (line 19). Every call to AddMapping which returns the status code *NEW* is called a *change event*, which triggers the output of the following information:

⟨ timestamp ⟩ ⟨FQDN⟩ ⟨IP⟩⟨score⟩ ⟨count⟩

The ⟨count⟩ field holds the number of IPBlocks to which ⟨FQDN⟩ maps at the time ⟨timestamp⟩ of the DNS change. This can be found by a simple table lookup, and is used later by the graph analysis component.

We emphasize that these change events by themselves are not exclusively indicative for malware, as most DNS changes represent completely normal Internet activity. Note that in addition to *real* changes (e.g., newly emerging services), we also register *perceived* changes, which are related to services that possibly existed for long time, but were never queried from the monitored network (see

[6] One tree per /8 network.

[7] The number of clusters per IPBlock is limited for performance reasons. Similarly, we limit the number of FQDNs per cluster (Table 1).

discussion on *limited visibility* in [10]). Typically, DNSMap outputs about 100,000 change events per day.

Maintenance: With time, the number of clusters per IPBlock would steadily increase, and therefore the system as a whole would become less and less sensitive to DNS changes. Therefore, we provide a MAINTENANCE routine which is executed every time interval Δ_{Ma} (see Section 4.2). The procedure takes care of removing outdated FQDNs (which were not recently observed in the traffic), and of reclustering the remaining ones. The following sequence of actions is performed. First, we remove all IPBlocks which do not contain any FQDN clusters. Second, for each IPBlock, we remove all empty clusters. And finally, we remove all FQDNs from all clusters from all IPBlocks. This implements the following removal strategy. All FQDNs are removed at most Δ_{Ma} after they were observed. Clusters which are not re-populated with any FQDNs during Δ_{Ma} are removed, and therefore "live" longer than Δ_{Ma}. IPBlocks are removed only if they do not contain any clusters, which can happen at earliest $2\cdot\,\Delta_{Ma}$ after their creation.

3.1.3. Merging and splitting IPBlocks

So far, we assumed that IPBlocks have a fixed size, which statically represent a certain number of IP addresses. We did, however, not discuss how IPBlocks are created and how they attain their size. This is explained in the following. We initialize each new IPBlock using a single IP address, and therefore require a procedure for "growing" IPBlocks such that they represent an IP *range*. This is accomplished by merging neighboring IPBlocks, given that we find that they host similar FQDNs. Conversely, we provide a splitting procedure for reverting previous merge operations, in order to be able to reflect DNS changes over time. A required key concept is the similarity measure σ between two IPBlocks, which we define in the following.

Consider two IPBlocks $\mathcal{A}$ and $\mathcal{B}$, which contain $m_{\mathcal{A}}$ and $m_{\mathcal{B}}$ domains, respectively. From all clusters in $\mathcal{A}$ we select the subset of clusters $c_k^{\mathcal{A}}$ with labels $L_k^{\mathcal{A}}$ which satisfy $\mathrm{DD}(L_k^{\mathcal{A}}, L_j^{\mathcal{B}}) \leq \Theta$ for at least one cluster $c_j^{\mathcal{B}}$ of $\mathcal{B}$. The relative share of domains of $\mathcal{A}$ that "fit" to the cluster configuration of $\mathcal{B}$ is then $\sigma_{\mathcal{A},\mathcal{B}} = \sum_k \frac{|c_k^{\mathcal{A}}|}{m_{\mathcal{A}}}$, which we call the *similarity* measure of two IPBlocks.

Definition 1 (**MERGECONDITION**)**.** Two IPBlocks $\mathcal{A}$ and $\mathcal{B}$ are merged *iff* (i) they are direct neighbors in the IP address space, (ii) they contain addresses from the same Autonomous System, and (iii) $\sigma_{\mathcal{A},\mathcal{B}} > \gamma$ and $\sigma_{\mathcal{B},\mathcal{A}} > \gamma$. In other words, we merge two IPBlocks when at least a percentage γ of FQDNs of each IPBlock are in clusters which are similar to the neighboring IPBlock's clusters.

In our experiments, we set $\gamma = 50\%$, i.e., merging requires at least a simple majority of similar domains. The resulting merged IPBlock then contains $m_{\mathcal{A}} \cup m_{\mathcal{B}}$ domains which we recluster.

Conversely, consider two IPBlocks $\mathcal{A}$ and $\mathcal{B}$ that were created from a third IPBlock $\mathcal{Z}$ such that $\mathcal{A}$ represents the first half of $\mathcal{Z}$'s IP addresses and $\mathcal{B}$ the second half. Let a cluster of FQDNs be "active" on an IP range if the cluster contains at least one FQDN which mapped to an IP address in this range. Let $\mathcal{A}$ contain all those domains of $\mathcal{Z}$ where the containing cluster was active in $\mathcal{A}$'s IP range (and for $\mathcal{B}$ respectively).

Definition 2 (**SPLITCONDITION**)**.** An IPBlock $\mathcal{Z}$ is split *iff* $\mathcal{A}$ and $\mathcal{B}$ do not satisfy MERGECONDITION.

We evaluate MERGECONDITION for all stored IPBlocks after each time interval Δ_{Mg}, and SPLITCONDITION every Δ_{Ma}. Typically, $\Delta_{Mg} < \Delta_{Ma}$ (see Section 4).

3.2. Graph analysis

DNSMap provides a stream of change events, most of which represent normal, non-malicious DNS activity. In the following, we apply graph analysis techniques for mining these events. Our approach is based on the conjecture that malicious, CDN-like DNS activity maps to *community structures* in a graph.

We consider the sets of FQDNs $\mathcal{F}$ and IP addresses $\mathcal{I}$ for which we received a change event within an evaluation epoch ε (e.g., 1 day). These events are represented as a bipartite graph $\mathcal{G}(V, E)$ with vertices V and edges E, where $V = \mathcal{F} \cup \mathcal{I}$ and each $e \in E$ indicates the existence of a mapping between an FQDN and an IP address. As our graph analysis component operates on prefiltered data, ε can be implemented as a sliding window, to create a new graph, using an updated list of change events, every few minutes. This ensures that newly collected evidence is quickly integrated in the graph, which allows for almost immediate detection.

As a first analysis step, we partition the graph and find the set of *connected components*, i.e., subgraphs which are not connected to each other [23]. Each constitutes a group of FQDNs and IP addresses which are (directly or indirectly) related to each other over time, *and* which were found to represent a significant change. Note that such a partitioning would not be possible without DNSMap. A direct representation of all DNS mappings as a graph would yield giant connected components, where the majority of services is (indirectly) connected to many others, due to the widespread use of CDNs. DNSMap accomplishes the omission of edges which represent activity that is similar to previously seen one, and therefore makes the graph separable.

As a second step, we remove all components which contain only one FQDN and one IP address (in the following called *singles*), as such mappings do not represent any kind of agile activity. In our experiments, this always had a dramatic effect and removed $\sim 99\%$ of all components. The following analysis focuses on the remaining components, which we refer to as *agile groups*.

Agile groups are subject to filtering rules, which are based on a set of features we describe in the following section. The remaining subgraph then represents the final output of our system, which we prune for a list of malicious FQDNs and IP addresses.

3.2.1. Agile group features

We classify the individual agile groups according to the following features, all of which are exclusively based on DNSMap's output and do not require any kind of active probing or elaborate processing. These features are later used to formulate queries, which return the FQDNs and IP addresses of those agile groups which match the activity patterns encoded in the queries.

$\phi 1$. The number of FQDNs per agile group. Cybercrime reuses IP addresses for multiple different FQDNs, in which case this number is high.

$\phi 2$. The number of IP addresses per agile group. Cybercrime hosts FQDNs on multiple IP addresses over time, in which case this number is high.

$\phi 3$. The number of different Autonomous Systems (ASes) per agile group. We find the AS for an IP address using MaxMind's freely available database.[8]

$\phi 4$. Although a benign FQDN might be known to map to N IP addresses, the graph analysis module may receive events indicating suspicious activity of the same FQDN on M other IP addresses, often with $N >> M$. We therefore remove all events that involve FQDNs for which we have significantly more reason to believe that they are benign than malicious. The maximum ⟨count⟩ value C_{max} for a reported FQDN indicates to how many IPBlocks this FQDN mapped in total. We compute the ratio $\phi_4 = M/C_{max}$, and consider FQDNs with $\phi_4 < 0.3$ as *likely benign*. In the remainder of this paper, we do not consider such FQDNs for further analysis. Note that this does not necessarily imply that the entire agile group an FQDN is belonging to is ignored.

$\phi 5$. Hosting providers typically allocate a set of IP addresses in a sub-network to a large, diverse set of FQDNs. These addresses often form dense groups, i.e., they differ only by a small (integer) value. In contrast, cybercrime hosting is often less organized. For example, Fast-Flux uses IP addresses that are scattered over multiple networks. This has been recognized also by other authors, and was often addressed by computing entropy-based measures from the set of used addresses [12].

However, entropy provides an overall metric for the distribution of the IP addresses in the agile group, but cannot capture the difference between *dense* and *sparse* groups of addresses. In the following, we therefore derive a score describing this "scatteredness". For each /24-network per agile group, we find the lowest (x_1) and the highest (x_n) of the integer representation of n IP addresses, and compute the average distance $\mu = (x_n - x_1)/n$ between them. We weight μ logarithmically and derive $\rho \in [0, 1]$. Additionally, ν conveys on how many addresses μ is based, and therefore expresses the confidence level of ρ.

$$\rho = -\log_2\left(\frac{\mu}{255}\right)/8 \quad \nu = -\log_2\left(\frac{n}{255}\right)/8$$

For each /24-network i, we compute a score $\sigma_i \in [0, 1]$ and define the agile group's "scatteredness" ϕ_5 as the mean of these scores.

$$\sigma_i = \rho + (1-\rho)\cdot\nu \quad \phi_5 = \textsc{Mean}(\sigma_i) \quad [0, 1]$$

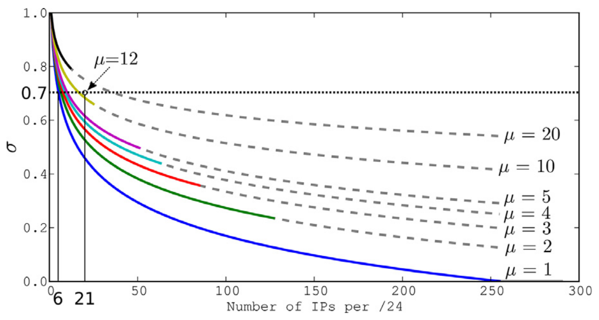


Fig. 3. IP distance scores σ for different numbers of IP addresses per /24-network seen, and different mean distances μ between them.

Table 1
DNSMap parameters.

Param.	Range	Description	Setting
Θ	[0,1]	DD threshold	[0.25,0.30,0.35,0.4,0.45]
Δ_{Ma}	(0,∞)	Maintenance interval	2 days
Δ_{Mg}	(0, ∞)	Merge interval	6 h
Mcl	[1,∞]	Limit number of clusters per IPBlock	50
Msz	[1,∞]	Max. number of FQDNs per cluster to store	30

Fig. 3 shows the values of σ_i for different μ and n. The dashed lines indicate theoretical values which cannot be reached (e.g., there cannot be 100 IP addresses with a mean distance of 20 in a single /24). E.g., in order to reach $\phi_5 \le 0.7$, a malicious component must *on average* either use $\ge$ 21 addresses per /24, or use at least six addresses (per /24) which are direct neighbors, or implement a trade-off between these two extremes. This is hard to achieve for cybercrime using agile DNS, which aims at always being "on the move" to evade countermeasures. In the remainder of this paper, we remove all agile groups with $\phi_5 \le 0.7$. Note that this is a conservative setting, as we typically observed $\phi_5 > 0.9$ for malicious components.

We emphasize that one could easily define more features (e.g., based on [5,8,11,12]), with the additional advantage of being able to exploit the graph's structural connection information. However, we did not consider this necessary for the purpose of this paper, as the detection results were excellent, and as a lower number of features can be tuned more intuitively.

4. Parameters and tuning

We turn now to discussing the configuration of our system consisting of DNSMap and graph analysis. We summarize DNSMap's parameters and the settings used in the experiments in Table 1. The settings of Δ_{Mg}, Mcl, and Msz mostly affect the degree of loss of the system's DNS information compression. More aggressive settings (i.e., lower Δ_{Mg} and higher Mcl and Msz) consume more system resources (i.e., CPU and memory), and create more accurate DNSMap representations. The limits imposed by Mcl and

[8] http://dev.maxmind.com/geoip/geolite.

Table 2
Examples for domain divergences in Fig. 2.

	d_1	d_2	d_3	d_4	d_5
d_1 = www.example.com	0.0	0.3	0.09	0.39	0.58
d_2 = mail.example.com	0.3	0.0	0.35	0.31	0.75
d_3 = www2.example.co.uk	0.09	0.35	0.0	0.44	0.63
d_4 = videos.example.com	0.39	0.31	0.44	0.0	0.77
d_5 = www.great-videos.com	0.58	0.75	0.63	0.77	0.0

Msz are reached for less than 0.1% of the IPBlocks and clusters we retrieved during our experiments. The most critical parameters are the divergence threshold Θ and the maintenance interval Δ_{Ma}, as well as the graph query parameters. We provide guidance for tuning them in the following.

4.1. Domain divergence threshold Θ

The configuration of Θ controls DNSMap's ability to absorb FQDN changes, and heavily influences the system's performance. High settings (i.e., closer to 1.0) make the system more "permissive", and lead to a lower number of change events, and therefore to reduced detection sensitivity. Furthermore, the overall number of clusters is reduced, which leads to a lower memory footprint, and to faster processing times. Low settings (i.e., closer to 0.0) have the opposite effects, and make the system more sensitive to differences between two given FQDNs. Therefore, we investigate how low we can set Θ while still allowing for small variations in DNS mappings.

The Domain Divergence provides a numerical measure for the similarity of two FQDNs which aims at reflecting the intuitive interpretation of a human analyst. Table 2 shows the domain divergences between the FQDNs from Fig. 2, and gives a first "hands-on" idea of the behavior of this metric. For our application, we definitely want to consider d_1 and d_2 similar, which requires $\Theta \geq 0.30$. Conversely, we clearly want to differentiate d_3 and d_5, therefore $\Theta < 0.63$.

For a more formal analysis, let us now consider the comparison of two FQDNs consisting only of two domain levels each. The maximum editing budget for two similar FQDNs X and Y, where $|X| = |Y| = 2$ and $X_1 = Y_1$, is given as $\Theta \cdot (|X_2| + |Y_2|)$ (based on the definition of LR - see Section 3.1.1). As one replacement operation consumes two editing "tokens", this allows, e.g., one different character if $|X_2| = |Y_2| = 6$ and $\Theta = 0.30$ $(0.30 \cdot (6 + 6) = 3.6 \rightarrow 3.6/2 \simeq 1)$. This leaves only limited degrees of freedom for malicious FQDNs trying to evade the detection threshold. Of course, the number of "credits" is proportional to the length of the FQDN. Longer FQDNs provide more freedom, at the cost of having to contain longer stable patterns.

By construction, the weight of the Levenshtein ratio diminishes with increasing domain level λ (see Eq. (1)). To illustrate this effect, we investigate the comparison of two FQDNs which differ w.r.t. 3-LD only. The influence of the unweighted string difference dd_3/ω_3 on DD depends on the fixed w_3, as well as on the relative length of 3-LD. We define this length as $\zeta = l_3/(l_2 + l_3)$. Fig. 4 illustrates the influence of a string difference on 3-LD on the total domain divergence DD between two FQDNs, assuming that both contain three domain levels and do not differ at all w.r.t. 1-LD and 2-LD. For example, in order to consider www.exampledomain.com and mail.exampledomain.com similar, we would need to set $\Theta \geq 0.19$ $(dd_3/\omega_3 = 1.0,\ \zeta = 4/17 = 0.24)$. On the other hand, xxxxxx.example.com and yyyyyy.example.com would require $\Theta \geq 0.39$ $(dd_3/\omega_3 = 1.0,\ \zeta = 6/13 = 0.46)$.

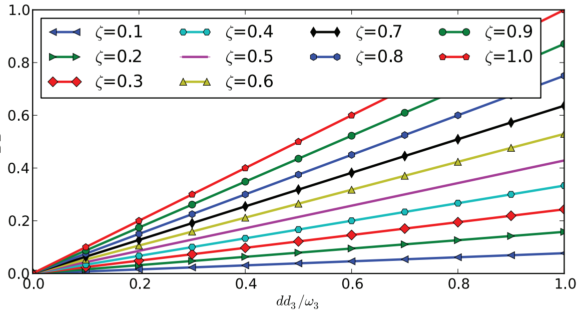


Fig. 4. Domain divergences DD for FQDNs with three domain levels which differ only w.r.t. 3-LD. Note that $\zeta = 1.0$ is an unreachable upper bound (as always $l_2 > 0$).

Based on this initial analysis, we evaluate the results for $\Theta = [0.25, 0.30, 0.35, 0.4, 0.45]$ in the following experiments (Section 5). Lower settings are to be preferred, as long as they do not severely impact the performance and number of false positives.

4.2. Maintenance interval Δ_{Ma}

The maintenance interval setting controls DNSMap's inertia to changes, as well as the memory consumption. The less often old DNS mappings are removed, the more memory is required and the longer we remember that (benign) FQDNs have been seen. At the same time, it becomes more likely that the storage limits of the IPBlock (Mcl and Msz) are exceeded, and that we are therefore forced to ignore a DNS mapping. Also, this causes the system to be less sensitive to changes, as more stored FQDNs per IPBlock increase the chance that a new FQDN is similar to one of them. Therefore, for high detection sensitivity, we want to set Δ_{Ma} to a low value. However, due to the daily cycle of Internet activity, a reasonable lower limit for Δ_{Ma} is 1 day—more aggressive cleanup would prematurely remove, e.g., peak-hour activity. During our experiments we found that some IP addresses of popular services are used even less often than once per day. We therefore set Δ_{Ma} =2 days for our experiments.

4.3. Graph analysis parameters

The final analysis stage requires as input a set of six parameters. The settings for ϕ_4 and ϕ_5 were already discussed in Section 3.2.1. Features ϕ_1, ϕ_2, and ϕ_3 intuitively quantify the FQDN/IP/AS-diversity per graph component over an evaluation epoch ε. In order to be able to detect even moderately agile cybercrime activity [24], we aim at setting ϵ as large as possible. Together, the setting of ε and

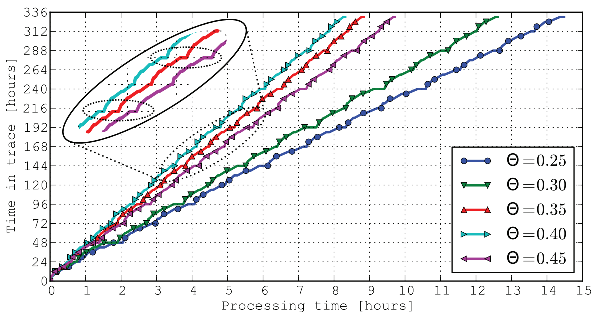


Fig. 5. Processing time for DS1, using different Θ.

the thresholds for ϕ_1, ϕ_2, and ϕ_3 define the lower bound for the degree of agility we target.

In our experiments (Section 5.3.3), we focus on two extreme scenarios. On the one hand, *Fast-Flux* networks use many (potentially hundreds) of IP addresses in many different ASes over time (large ϕ_2, ϕ_3), but only a limited number of FQDNs (small ϕ_1). On the other hand, graph components representing (agile) *Malware Hosting*, typically use several FQDNs (large ϕ_1), but often only a small set of addresses in a low number of ASes (small ϕ_2, ϕ_3).

5. Experimental evaluation

The following experiments are driven by three goals. First, we evaluate the performance of DNSMap for different settings of Θ, and demonstrate that it scales even to large deployment scenarios over extended periods of time. Then, we investigate whether there exists indeed a certain level of low agility in benign networks which needs to be absorbed by DNSMap, so that the corresponding FQDNs and IPs do not spoil the detection results. Finally, we discuss the configuration of the graph analysis stage and the detected cybercrime activity. All experiments were done on a standard PC (Intel i5@3.1 GHz) with 16 GB of main memory, of which we used at most 9 GB during our tests. We experimented with two data sets (see Table 3) where duplicates were filtered out with a time window `filtWindow` = 1800 s (Fig. 1) before processing them, in order not to have disk I/O and the DNS parser dominate the processing time. All our tools are implemented in Python and are available at https://github.com/anderasberger/pydnsmap.

5.1. Performance

In the following, we provide an empirical evaluation of our system's performance. We refer the reader to the appendix for a formal performance evaluation. Fig. 5 shows the performance of DNSMap for processing DS1, using different settings for Θ. Processing 48 h of DNS data requires only ~1–2 h, and the processing time grows about linearly with the length of the observation period. Interestingly, and in contrast to our initial expectations, the system does not necessarily run slower with lower values of Θ. We found that often the main difference in processing times originates from the time required for the evaluation of SplitCondition. This can be seen from the marked regions in Fig. 5, which show a relative drift happening every 2 days of trace time, which corresponds to Δ_{Ma}. Higher values of Θ cause more IPBlocks to be merged over time, and therefore reduce the overall required number of evaluations of SplitCondition. However, larger IPBlocks may also have more clusters, and therefore the time required by call increases. Overall, the setting of Θ has only limited effect on the processing time, and easily allows for realtime processing.

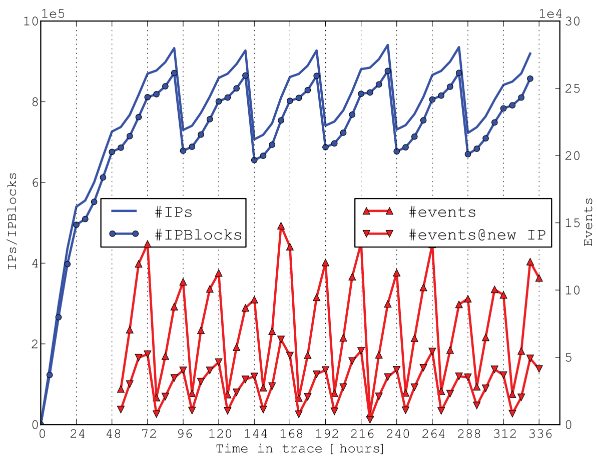


Fig. 6. Number of IP addresses/IPBlocks/events created by DNSMap over time, for DS1, with Θ = 0.35. By configuration, the first change events are produced after 48 h. Note the two different scales on the *y*-axis.

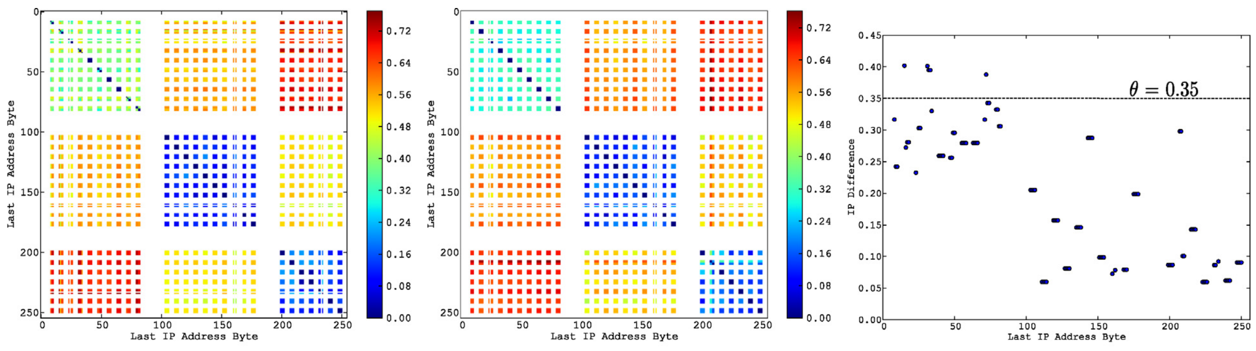
Next, we investigated the volume of DNSMap's output in terms of number of IPBlocks and change events over time, which define the memory requirements and the size of the graph to be analyzed, respectively. Fig. 6 shows the results for Θ = 0.35. After an initialization period of 48 h, the system reaches a stable working point which, from then on, only adapts to changes in traffic volume (i.e., the number of IP addresses seen) depending on the time of day. Note that the number of IPBlocks is not significantly lower, which indicates that the vast majority of IP addresses do not host similar services as their neighbors. The number of change events also depends on the time of day, and is relatively stable throughout the entire 2 weeks of data. Around a third of all change events relate to new IP addresses (i.e., which were at this time not contained in DNSMap), which underlines the high dynamicity of Internet DNS traffic.

5.2. Benign service agility

The main objective in the design of DNSMap is the ability to describe the agility of IP address ranges, and to find a set of labels for the FQDNs which map there. Ideally, most benign IP ranges would host similar FQDNs over time, and can therefore be distinguished from highly agile cybercrime activity. In the following, we evaluate to which extent this assumption holds using data set DS1 (see Table 3). We set Θ = 0.35 and analyze DNSMap's output from processing the entire 2 weeks of DNS data.

Table 3
Data sets from large access provider.

Name	Time frame	Total mappings	Unique FQDNs	Unique IP addresses	Unique clients
DS1	11/22/2010–12/05/2010	3.2B	13M	1.6M	~ 500k
DS2	11/15/2011–11/21/2011	2.4B	5.4M	1.2M	~ 1M



(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Fig. 7. Subfigures (a) and (b) represent the status of the DNSMap representation at 6 days (resp. 14 days) into the data set. Shown are the relations between the observed IP addresses of the /24-network of Akamai for which we saw the most IP addresses (102 out of 255) being used. We show the differences of all pairs of IP addresses in this range, w.r.t. to the FQDNs they host. Figure (c) shows the difference over time.

An indicator of the overall DNS agility is given by the final number of clusters per IPBlock. Many clusters would indicate many dissimilar FQDNs per IP address, which would contradict the assumption that we can extract stable FQDN patterns per IPBlock. After processing the entire 2 weeks of data, DNSMap stored ~ 850k IPBlocks and ~3.4M FQDNs. The mean number of clusters per IPBlock was 2.2, with a 95th's percentile of 6.0 clusters. Therefore, only 5% of the IPBlocks required more than six clusters for containing all FQDNS mapping to them in (at least) the last 4 days ($2 \cdot \Delta_{Ma}$). 75% of all IPBlocks just had a single cluster. Overall, there was significant stability in the extracted FQDN patterns.

But how does DNSMap represent the DNS activity in one particularly busy (i.e., agile) network, and how does this develop over time? For this purpose, we define the *difference* between two IP addresses as the average of the smallest *Domain Divergences* between any two pairs of cluster labels of the two containing IPBlocks. We compute this difference for each pair of IP addresses in a certain range. The difference between two IP addresses from the same IPBlock is therefore zero. Fig. 7a shows the differences between the IP addresses in a /24-network of Akamai, w.r.t. the hosted FQDNs, after 6 days.[9] The diagonal represents the difference of each address to itself, white space indicates that the corresponding address has not been seen by DNSMap. Three different sub-ranges are clearly visible, which are all rather homogeneous by themselves, but quite dissimilar from each other: (A:10-80) addresses ending on 10–80 host a large variety of different FQDNs of smaller services; (B:105-160) host various Facebook services, plus a small set of other (large) sites (e.g., Symantec); (C:200-250) almost exclusively host Facebook sites. This apparent dissimilarity indicates that we are able to identify sub-ranges with specific hosting patterns. Fig. 7b shows the same information after processing all 14 days with DNSMap. Interestingly, the relative differences between the addresses are almost unchanged. Note, however, that some slightly larger blocks appeared along the diagonal: the availability of more DNS data over time apparently caused DNSMap to merge neighboring IP addresses. Finally, Fig. 7c shows the difference between both snapshots. We find the containing IPBlocks for each address at both times, and compute their differences. Block A shows some limited variation in the hosted FQDNs, which is expected for a range of smaller sites that are observed rarely. Still, most IP address differences are below $\Theta = 0.35$, therefore most of these services would not trigger a change event. Blocks B and C are remarkably stable. Even after 8 days, the cluster labels are highly similar.

5.3. Cybercrime detection

We now turn to the evaluation of DNSMap's change events for detecting cybercrime activity. We consider an agile group as malicious if we were able to confirm that at least one FQDN in this group is malicious. Indeed, in our trials we found such evidence for a significant share of FQDNs in each group. Note that we exclusively investigate the FQDNs contained in the change events created by DNSMap. One may want to retrieve the list of *all* FQDNs mapping to an IP address, as soon as it was found to host malicious services, given the availability of such data *ex post*. It is, however, not immediately obvious where to stop with such an analysis, as one might then also consider retrieving the IP addresses which were used by FQDNs which

[9] We chose 6 days to allow for some initial training for more representative results.

mapped to a malicious address, and so on. Hence we leave such an analysis for future work.

The evaluation of our results is a challenging task, as there exists no reliable ground truth. A central metric for evaluating classification systems is the *false positive rate* (FPR), which is defined as $\text{FPR} = \text{FP}/(\text{FP} + \text{TN})$ where FP and TN are the total number of false positives and true negatives, respectively. As we do not have ground truth available, we cannot calculate FPR. However, in our experiments we analyze more than a million FQDNs and report at most only a few thousand as malicious. Intuitively, most analyzed FQDNs are not malicious and are therefore true negatives. Therefore, the false positive rate is expected to be extremely low—in the order of *one promille*. Therefore, we do not report it separately for each experiment.

Due to the lack of ground truth, we cannot assess the number of false negatives. However, we can adjust the parameter settings of our system such that the corresponding agile activity is guaranteed to be reported. This comes at the cost of reduced classification *precision*, which is defined as TP/(TP + FP) where TP is the number of true positives. Sometimes we observe significantly more FP than TP, which results in low precision. However, the absolute number of results (i.e., TP + FP) is generally low in our experiments[10] and can be sorted out manually (see Section 5.3.2). Therefore, we deliberately trade precision for high sensitivity, to assure that even moderately agile activity is detected, by setting the system parameters accordingly.

Similar to [8,11,12], we use publicly available blacklists as one means for assessing the quality of our results. Note, however, that we never found *all* detected malicious FQDNs in any single blacklist we used. This is not only because of limitations of the underlying analysis systems, but also due to the fact that blacklists are based on different data sets (e.g., Spam emails) from other networks. Typically we do not know what specific techniques were used to flag a domain name as malicious, nor can we be absolutely sure that the blacklisted domains are actually malicious [14].

As an example, we consider the blacklist from exposure.iseclab.org. The underlying analysis is based on [11], which pursues similar goals as our approach. We downloaded the list on November 22, 2011, i.e., 1 day after the end of DS2.[11] Out of ~ 60,000 domains in the blacklist, only 478 were also present in DS2. This indicates that either the blacklist contained many outdated domains or that the traffic seen by EXPOSURE was significantly different from DS2. Only 16 of these domains were reported as changes by DNSMap, and entered the second analysis stage, which resulted in zero final alerts. We manually checked all 16 domains, and could not find any evidence that they were related to cybercrime activity. In fact, most of the sites were still online (2.5 years after the data set was recorded), which indicates that they are actually benign. Out of the remaining 462 domains, 426 were first observed *during* the initial 2-days training phase of our system, and were therefore correctly classified as representing no DNS change later on. The remaining 36 domains were seen the first time *afterward*. Nevertheless, for 304 out of the 426 we found the *same* group of eight IP addresses hosting 103 *other* domains during our experiments, which were not found by [11]. Viewing this activity as an agile group enabled us later to understand that this was a single cluster of botnet sinkhole IP addresses (see Section 5.3.1). We checked both the 122 other domains and the remaining 36 domains manually and found no evidence for malicious activity. The automated comparison with existing blacklists can therefore be grossly misleading, in particular for evaluating the number of false negatives. In our example, we detected all malicious-hosting IPs reported by [11] without any false positives, while a simple comparison of the results would have yielded 478 false negative FQDN reports.

Consequently, we employ a *semi-manual* approach for evaluating our results. We rely on a set of publicly available blacklists[12] and on online services[13] for getting a first idea of the true nature of a given domain. If, e.g., we found a domain on a Spam blacklist and a corresponding report on phishtank.com, we considered this a true positive. Additionally, we did *manual* Google searches for most reported domains, which typically led us to a variety of malware analysis sites. Still, for some domains we were not able to find any information online, despite the fact that their activity would definitely deserve a closer look.

Due to lack of ground truth, we cannot reliably quantify the number of false negatives. However, we do know that our system *guarantees* to detect all DNS activity exceeding the limits imposed by the parameter configuration. Thus, we choose highly sensitive configurations which leave very limited room for malicious, agile DNS activity, and show that even these settings yield extremely few false positives.

5.3.1. Results with limited training

Our system is able to provide valuable results even with *minimal training*. To demonstrate that, we analyze the change events of the third day ($\varepsilon = 1$ day) of our data set DS1, i.e., after 2 days of initial training. When setting $\Theta = 0.35$ we received 332,606 change events, out of which 181,822 *singles* and 17 *likely benign* FQDNs could immediately be discarded.

First, we targeted groups of agile FQDNs sharing a set of IP addresses (Malicious Hosting) and discarded all components with $\phi_1 < 10$, $\phi_2 < 2$, and $\phi_3 < 2$. The remaining 199 FQDNs were split in 11 agile groups: the largest one represented four sinkhole IP addresses for botnet mitigation, which hosted 51 random-looking FQDNs. 28 FQDNs hosted on five addresses in three ASes clearly represented cybercrime, which we call the *jailcoach* cluster (see Section 5.3.3). Another group of 10 FQDNs related to online pharmacy Spam (e.g., bargaincapsules.ru). The remaining eight groups with 110 FQDNs in total represented benign sites using a small set of different hosting providers, which were previously requested not often enough to be known to the system. More aggressive

[10] A few thousand FQDNs in a few tens of groups per week.

[11] The blacklist service was not available for the time frame of DS1.

[12] malwaredomainlist.com, joewein.net, [11], abuse.ch.

[13] phishtank.com, projecthoneypot.org, projecthoneypot.org, virustotal.com

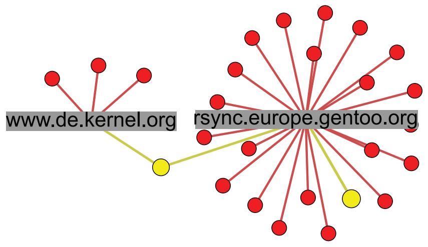


Fig. 8. Whitelisting example: the relative node betweenness of rsync.europe.gentoo.org is 92%, and it is therefore by far the most important node in this component. Removing it after manual investigation causes this component to fall apart and lets also www.de.kernel.org disappear from the final results. Note that we collapse several IP addresses from the same AS for better visualization (yellow nodes). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

thresholds for ϕ_1, ϕ_2, ϕ_3 would have removed all false positives, without missing any true positive.

Second, we targeted sites which quickly jump from one IP address to the next (Fast-Flux) and required that $\phi_1 \geq 1$, $\phi_2 \geq 20$, and $\phi_3 \geq 5$. After filtering the original graph, only eight FQDNs in three groups were left. The first one contained the malicious Fast-Flux domain ttcuhdwk.biz (mapping to 38 IP addresses in 28 ASes). Another group of six malicious FQDNs mapped to 39 IP addresses in 32 ASes. The remaining FQDN irc.efnet.org on 21 IP addresses in 20 ASes was a false positive. A manual investigation revealed that this FQDN was not requested *at all* during the first 2 days. Although this represented no malicious activity, it therefore still represents the kind of agile DNS which we aim to detect.

5.3.2. Targeted whitelisting

The case of http://irc.efnet.org irc.efnet.org showed that certain benign DNS activity is indistinguishable from cybercrime. This is an intrinsic limitation of DNS analysis, which also affects other approaches. We explicitly acknowledge this, and integrate a fine-grained procedure for targeted whitelisting in our system. That is, in contrast to other approaches which remove large numbers of domains a priori [5,8,11,12], we identify those which are truly indistinguishable from cybercrime, and remove only them.

We take advantage of the representation of DNS mappings as a graph: some agile groups may be large and contain, e.g., thousands of FQDNs and IP addresses, and therefore it is not immediately clear which of these FQDNs should be checked for potentially being whitelisted. For every agile group, we find the FQDN node with the maximum normalized *node betweenness* [23]. This indicates that it lies on many shortest paths between two other nodes, and is therefore important for the component's connectivity. After checking manually that the FQDN is actually benign, we create a new whitelist entry, which often causes the group to fall apart. Subsequently, we rebuild the graph without considering this FQDN. The remaining parts of the former component are then typically removed when applying the parameter thresholds. Fig. 8 shows a simple example.

In other cases, there is no single FQDN responsible for a group of sites being reported. One such service is NTP, which involves a large number of different IP addresses around the globe, which are used for many FQDN aliases (see also [12]). Another example is CoralCDN, which uses FQDNs like `PREFIX.nyud.net`, where `PREFIX` is an FQDN of an existing website. A cached copy of `PREFIX` is located in a large peer-to-peer network, and the address of the allocated server is returned to the requesting host. In other words, a large number of addresses serves an ever changing set of FQDNs, which resembles cybercrime activity. In this case, there exists an FQDN *pattern* which we should whitelist. Therefore, we find the popularity of FQDN suffixes for each group, and can quickly identify the dominant one, which we then whitelist eventually. Table 4 shows the *entire* set of highly specific whitelist patterns we derived during our experiments, and which we use throughout the remainder of this paper.

Table 4
Whitelist used for experiments.

	Whitelist
NTP	**.ntp.org*; de.pool.ntp.arcor-ip.net time.mk.cc; ntp.pch.at
CoralCDN	**.nyud.net*; **.nyucd.net*
Other	cf.www.directadserver.com; grey-area.mailhostingserver.com cf.www.forwardizm.com; pool.sks-keyservers.net **.files.wordpress.com*; liveupdate.symantecliveupdate.com safebrowsing-cache.google.com; www.apple.com rsync.europe.gentoo.org

5.3.3. Cybercrime detection scenarios

As discussed in Section 4.3, the agile group parameters allow us to encode queries to target specific cybercrime hosting strategies. For brevity, we limit ourselves here to two main scenarios. First, we address Fast-Flux (FF) activity by requiring that $\phi_1 \geq 1$, $\phi_2 \geq 20$, and $\phi_3 \geq 5$. This configuration resembles the goals of [12], but is more sensitive and allows for much quicker detection (see Section 2). Second, we find Malicious Hosting (MH) patterns by requiring that $\phi_1 \geq 50$, $\phi_2 \geq 4$, and $\phi_3 \geq 2$. For our experiments, we consider the change events of the last week of DS1 (i.e., $\varepsilon = 1$ week), and of the last 4 days of DS2 (i.e., $\varepsilon = 4$ days). For each data set and each detection scenario, we investigate the agile groups which are reported for $\Theta = [0.25, 0.30, 0.35, 0.4, 0.45]$. Table 5 shows the overall results, which demonstrate that even for low settings of Θ and large ε the absolute number of false positives is still small. Certainly, we could have whitelisted a limited number of further FQDNs, up to the point where we would have received no false positives at all, for this particular data set. However, as benign DNS activity is dynamically changing over time, some false positives are always to be expected. Therefore, we rather show these realistic results.

In the following, we discuss the results for $\Theta = 0.35$ in detail. We checked manually that this setting consistently revealed all malicious agile groups that were found using lower Θ, and returned fewer false positives (cf. Table 5).

Fast-Flux. For DS1, we retrieve 291 FQDNs in 13 agile groups, which we sort according to the number of IP

Table 5
Results for different values of Θ and two detection scenarios (FF, MH). Shown is the precision of the analysis, i.e., the ratio of confirmed malicious agile groups and the total number of reported agile groups. We show absolute numbers as well as the calculated ratio (in brackets).

		Θ = 0.25	Θ = 0.30	Θ = 0.35	Θ = 0.40	Θ = 0.45
DS1	FF	4/14 (0.29)	4/13 (0.31)	4/13 (0.31)	3/10 (0.3)	2/9 (0.22)
	MH	9/42 (0.21)	9/33 (0.27)	9/29 (0.31)	7/27 (0.26)	6/26 (0.23)
DS2	FF	9/21 (0.43)	9/21 (0.43)	9/20 (0.45)	9/19 (0.47)	9/18 (0.50)
	MH	4/19 (0.21)	4/16 (0.25)	4/14 (0.29)	4/10 (0.4)	3/8 (0.375)

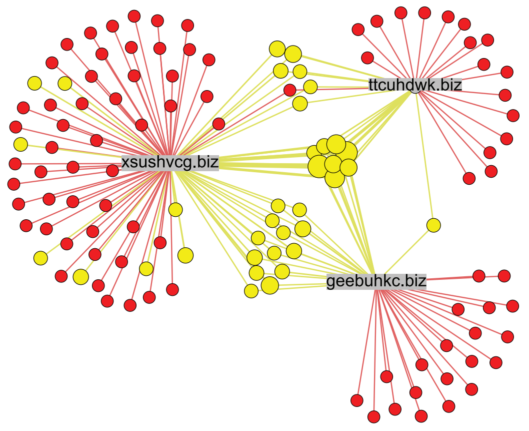


Fig. 9. Group of Fast-Flux sites.

addresses per group. The top four groups include three confirmed malicious ones, and none of them contained less than 50 IPs. The largest group contains the malicious FQDNs xsushvcg.biz, geebuhkc.biz, and ttcuhdwk.biz (which we found already in Section 5.3.1), and 365 IP addresses in 155 ASes (see Fig. 9). The three other malicious groups contained 16/86/152 FQDNs and 136/50/23 IP addresses, respectively. The remaining nine groups with 34 FQDNs in total were misclassified benign, yet highly agile, sites (e.g., connect.facebook.net).

For DS2, we retrieve 321 FQDNs in 20 agile groups. A group with two FQDNs on 288 IP addresses in 125 ASes represented malicious Fast-Flux. Less obviously, a group of 12 FQDNs (34 addresses/12 ASes) was used for hosting malware (e.g., the Cycbot Trojan on fdg45e.nl.ai). Interestingly, seven other groups contained only one FQDN and exactly 20 IP addresses in 15–20 different ASes each. All these FQDNs were subdomains of syringemexican.com, and used domain prefixes which appeared to be randomly generated. We were unable to find any indication that these highly suspicious domains are actually benign. The remaining ten agile groups represented benign activity. Note that the initial training period for DS2 was only 2 days and $\varepsilon =$ 4 days, which was the reason for misclassifying some popular services. However, even in this case, such low numbers of false positives—10 groups for 4 days of data—can be quickly sorted out manually.

Malicious hosting. For DS1, we find 29 agile groups with 3409 FQDNs, of which nine groups with 892 domains in total represented malicious hosting activity. One particularly interesting group is shown in Fig. 10. It contains 147 FQDNs (eight addresses, five ASes), out of which we found 76 FQDNs in blacklists which were derived from Email Spam. Note that neither the number of IP addresses per FQDN nor the number of ASes is excessively high. Rather, the overall agility of the FQDNs and IP addresses as a group stands out, which made them detectable. The fact that a subset of these FQDNs were found also by the analysis of Spam emails underlines the complementarity of these approaches. Spam-based blacklists are typically highly accurate, but are limited in scope. Conversely, DNS-based analysis is less precise, but is able to provide the big picture for cybercrime activity, which goes beyond the detection of network resources hosting Spam content.

Another particularly interesting example is a group of 86 FQDNs (50 addresses, 28 ASes). Most of the FQDNs have the format `<PREFIX>.youngand<WORD>1.com` where ⟨`PREFIX`⟩ was a random string and ⟨`WORD`⟩ an English word (e.g., `girls`). Many of these FQDNs were found in Spam emails, and therefore appear in several blacklists. Other sites hosted on the same IP addresses do not appear in any blacklists, though (e.g., ulqsibydur.com). We suspect that these domains were used for a different purpose, and were therefore not seen by the Spam analyzers.

For DS2 we find 14 groups with 1399 FQDNs, of which 408 domains in four groups were indeed malicious. Note that the vast majority of non-malicious groups we found related to adult content offerings, which use a highly agile set of domain name aliases to attract more customers. False positives can hardly be avoided in this case, this should be addressed by custom whitelisting.

6. Discussion

In our experiments we found various kinds of malicious activity, ranging from Fast-Flux (probably used for Botnet Command-and-Control), over Phishing and exploit sites, to domains used in Email spam. Even moderate agility patterns are detected, and many of them relate to malware activity. Our parameter settings force cybercrime to significantly reduce its agility patterns for going undetected. However, this highly sensitive configuration comes at the cost of misclassifying certain benign activity as malicious. We argue that such activity *should* be reported though, as DNS analysis alone is not able to differentiate it from malicious services, when both show similar levels of agility. A targeted, in-depth analysis is *enabled* by the low number of misclassified domains, and *supported* by the structural representation of our approach (see Section 5.3.2). As the

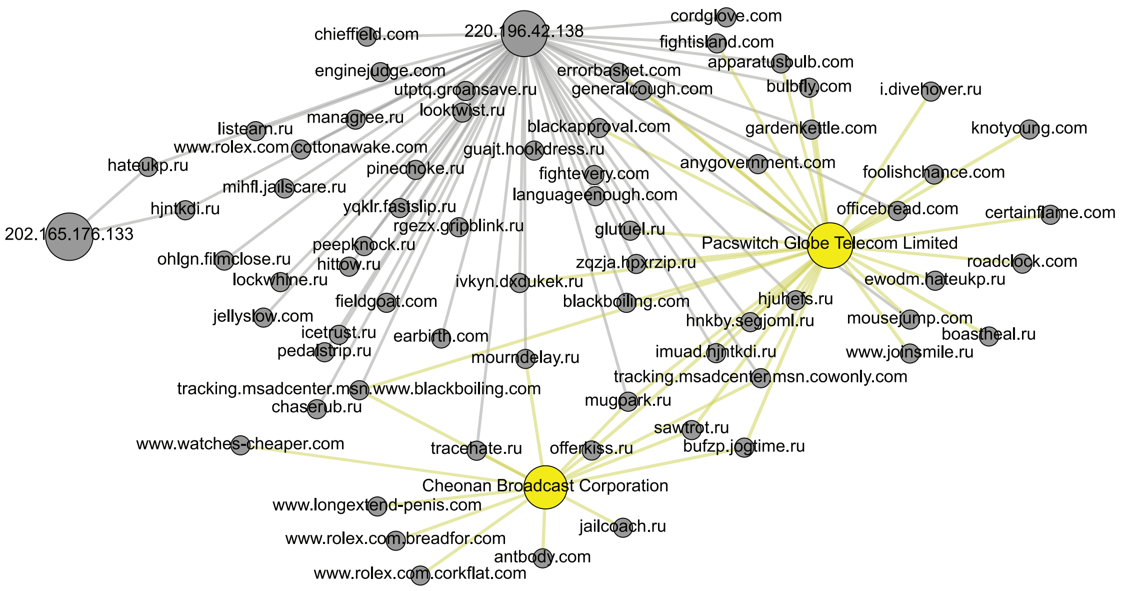


Fig. 10. The *jailcoach* cluster: a group of cybercrime sites, most of which seem to be generated by concatenating two English words (e.g., jailcoach.ru). Few sites relate to selling of fake watches (e.g., www.rolex.com.SUFFIX); others follow a different naming pattern (e.g., hjuhefs.ru). We removed about two thirds of the original graph for better visualization.

graph analysis step in our methodology is computationally inexpensive, one can run many queries using different parameter settings (i.e., ϕ_1, ϕ_2, ϕ_3) in short time and thereby interactively explore the DNS data at hand.

We consider the graph representation of the system's output as highly valuable. Many times we were able to understand immediately whether an agile group is indeed involved in malicious activity. Often, well-known domains (e.g., reddit.com) were connecting several benign sites, and whitelisting them caused an entire misclassified group to disappear from the results. Conversely, for some groups, only the existence of a small number of obviously dubious sites raised our suspicion about a group of domains which by themselves mostly appeared inconspicuous. We believe that it is essential to keep the human analyst in the loop, as many patterns are difficult to reveal in an automatic manner.

An important feature of our system is its ability to identify malware domains independently of the *number* of requests. This allows us to detect new outbreaks early, and enables timely reactions (e.g., blocking). In a real-world deployment, the number of requests *may* be used as an additional feature, to quantify the popularity of a particular detected site. However, it is not strictly required by the system. Likewise, one would construct more complex graph queries than we used in our experimental evaluation, by logically combining settings for different scenarios (e.g., Fast-Flux *OR* malware hosting).

Our approach is based on the high stability in DNS mappings of benign services (see Fig. 7). Popular sites require more hosting resources (i.e., IP addresses), and appear therefore more agile, thus requiring more training data for being properly modeled. An interesting property of this essential modeling step results from the fact that sites which are requested often are better modeled than less popular ones. We emphasize that DNSMap by design reports *all* DNS mappings which involve a new IP address, and such activity is therefore *guaranteed* to be detected if sufficiently many addresses (in our trials: 4–20 per week) are being used. We can detect such moderate fast flux activity due to the thorough understanding of the activity of benign sites. Large services may use many more IP addresses, but they use the same ones over time, and they do not constantly change the FQDN patterns.

6.1. Limitations

The evaluation of our system's output is heavily based on the concept of guilt by association, i.e., if *one* FQDN or IP address in a graph component is found to be malicious, then we consider *all* of them malicious. The advantage of being able to reveal malware activity which does not stand out by itself, comes with the drawback that also legitimate sites may wrongly be classified as malicious. In fact, in our experiments we occasionally found sites which appeared benign, but happened to map to IP addresses used by malware. We consider this tradeoff to be inherent to DNS-based analysis, though, which is unable to observe the actual data transfer between IP addresses. A focused in-depth analysis (e.g., using deep packet inspection) is recommended in these cases.

Note that the DNSMap representation is valid for a single vantage point only, and cannot be trivially transferred to other networks. This is because the mappings between FQDNs and IP addresses often depend on the geographical location of the requesting host, so to optimize the data transfer performance between server and client. Data sources like SIE[14] which provide aggregated DNS data from

[14] archive.farsightsecurity.com

multiple vantage points and therefore omit this additional information, have not yet been evaluated for our system.

6.2. Evasion strategies

In general, an adversary has two main options for evading our system: either escaping DNSMap's change detection, or going unnoticed w.r.t. the corresponding graph features, i.e., using only few FQDNs, IP addresses, and ASes per agile group. This comes at a cost, as fewer IP addresses being used per FQDN impacts the reliability of the malware service. Knysz et al. discuss fast-flux evasion strategies in [24], and derive models describing the relation between the number of online malware IP addresses and the availability of the corresponding malware sites. They consider a minimum number of 100 unique IP addresses per week and FQDN, which results (according to their model) in an average of 2.89 online IP addresses and a connection loss probability of 71.1%. Although this would already result in poor malware connectivity, this kind of activity is still comfortably within the sensitivity limits of our system. In fact, we would have revealed any Fast-Flux activity which involves $\geq$ 20 IP addresses per week and *agile group* (as opposed to a single FQDN). Therefore, we can easily detect malware activity even when all the proposed evasion techniques are implemented, and the overall malware utility is already considered poor.

Therefore, the adversary might try the second option, i.e., avoid that malware activity results in DNS change events in the first place. All new IP addresses being used are reported always, therefore the challenge lies in using only IP addresses which are known to DNSMap, and use FQDNs which are similar (i.e., $DD < \Theta$) to the corresponding IPBlocks' cluster labels. Less than ϕ_1 new FQDN “families” (i.e., clusters) on $< \phi_2$ IP addresses in $< \phi_3$ ASes can be introduced per epoch ε for going undetected. As shown in Section 4.1, our selection for Θ leaves only limited degrees of freedom, and forces the adversary to use stable patterns for constructing FQDNs which appear similar (e.g., by using a common suffix). This strategy cannot be changed for at least ε, i.e., 1 week in our case. In other words, the adversary is forced to use a less agile mapping procedure, which is the opposite of what was originally intended. This leaves significant time for other detection approaches (e.g., malware binary analysis) for identifying this pattern, and deriving a corresponding blacklist entry (e.g., $*\langle\texttt{SUFFIX}\rangle$). Furthermore, these restrictions have a severe impact on malware activity which requires flexibility in the choice of FQDNs for various reasons, e.g., Phishing FQDNs which often mimic the FQDN of the currently targeted site (e.g., www.example-bankk.com), or sites which use certain FQDN patterns for appearing benign (see, e.g., Fig. 10).

7. Conclusion

We proposed and discussed a cybercrime detection system which is based on DNS FQDN-to-IP-address mappings. We extract these mappings from traffic data, and find profiles describing typical FQDN patterns for arbitrary-length IP ranges. Cybercrime uses DNS for combining high service availability with resilience to countermeasures. This *agile* DNS activity results in changes to the DNS profiles, which we further investigate using graph analysis. In a number of experiments we showed how to target different malware activity and discussed the difficulties of evading our detection system.

Further improvements are possible, which we consider for future work. We proposed a very small set of graph query parameters, which of course can be extended. For example, we conducted early experiments using a database for retrieving the total number of DNS queries for a certain suspicious FQDN. Typically, one would especially be interested in groups of malware sites which are looked up by many different hosts, and we expect a further reduction in the number of false alarms by introducing a corresponding feature. Furthermore, additional graph analysis measures (e.g., degree distribution) may yield interesting results. Another promising direction for future work is the integration of additional data. In particular, we will consider the inclusion of information describing the authoritative name servers for the domains represented in our graph. This is related to the ideas presented in [8,13] and is expected to link suspicious domains from different agile groups, and thus provide even better detection performance.

Acknowledgments

The work of Antonio Pescapé is partially funded by the Italian Ministry of Education, Universities and Research in the context of Art. 11 DM 593/2000 for NM2 SRL. We would like to thank Eduard Natale and Mirko Schiavone for supporting us with the software implementation. This work has been supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

Appendix A. Complexity analysis

Space. IPBlocks are stored in a set of RBTrees. Each IPBlock holds a set of clusters of domains. We limit the number of clusters per IPBlock to Mcl and the number of domains per cluster to Msz (see Table 1). In the worst case, each IPBlock represents a single IP address only, thus maximizing the number of required IPBlocks.. Therefore, in the worst case, the space complexity for storing DNS mappings for n observed IP addresses is $\mathcal{O}(\text{Mcl} \cdot \text{Msz} \cdot n)$.

The analyzed graphs require $\mathcal{O}(V + E)$ space, where the number of nodes $V = n^* + d^*$ for $n^* \leq n$ suspicious IP addresses and d^* suspicious domains. The number of edges E depends on the actual DNS mappings and is bounded above by $E_{max} = n^* \cdot d^*$. In practice, the number of edges is significantly lower (see Section 3.2).

Time. DNSMap consists of three central operations, namely adding domains (see Algorithm 1) as well as merging and splitting IPBlocks (see Section 3.1.3). While adding domains is highly efficient, merging and splitting are more costly operations (see also Section 5.1).

Adding domains to an existing cluster and adding a new cluster to an IPBlock correspond to simple set operations and have a time complexity of $\mathcal{O}(1)$. Visiting all stored IPBlocks corresponds to traversing the RBTree and requires

$\mathcal{O}(n)$ time in the worst case [22]. We focus on the more significant operations in the following.

Given a DNS mapping of domain d to IP address a, the lookup of the IPBlock containing a in the RBTree has a worst-case time complexity of $\mathcal{O}(nlogn)$ [22]. Assuming that no IPBlock is found, a new IPBlock is created in $\mathcal{O}(logn)$ time. If an IPBlock b is found, the *DD* between d and all cluster labels of b needs to be computed. The computation of *DD* is based on computing the Levenshtein distance, which has a complexity of $\mathcal{O}(j \cdot k)$ for processing two strings with length j and k, respectively [20]. Let m be the maximum length of a cluster label and $|d|$ the length of d. As there are at most Mcl clusters per IPBlock, the computation of all *DDs* has a time complexity of $\mathcal{O}(\text{Mcl} \cdot m|d|)$.

Merging and splitting IPBlocks is based on the similarity score σ defined in Section 3.1.3. It requires the computation of *DD* for all pairs of clusters of two IPBlocks. As *DD* is symmetric, the computation of σ has a worst-case time complexity of $\mathcal{O}(\text{Mcl}^2 \cdot m^2)$. In the (highly unlikely) worst case where $\sigma_{A,B} \leq \gamma$ for all A, B (see Section 3.1.3), each IPBlock needs to be merged (split) after each interval Δ_{Mg} (Δ_{Ma}). The cluster labels represent the string medians of the set of domains in the corresponding cluster and are recomputed when IPBlocks are merged or split. Computing the string median is an NP-hard problem [25]. However, as we limit the number of domains per IPBlock (by limiting the number of clusters as well as the number of domains per cluster), this poses no significant problem for real-world applications (see the empirical evaluation in Section 5.1).

Finally, graph analysis is based on finding the graph's components in $\mathcal{O}(V + E)$ time [23].

References

[1] Norton Cybercrime Report, Technical Report, Norton, 2013.
[2] G. O'Gorman, G. McDonald, Ransomware: A Growing Menace, Technical Report, 2012.
[3] C. Grier, L. Ballard, J. Caballero, N. Chachra, C.J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M.Z. Rafique, M.A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, G.M. Voelker, Manufacturing compromise: the emergence of exploit-as-a-service., in: Proceedings of ACM Conference on Computer and Communications Security, 2012.
[4] J. Caballero, C. Grier, C. Kreibich, V. Paxson, Measuring pay-per-install: the commoditization of malware distribution, in: Proceedings of USENIX Security, 2011.
[5] E. Passerini, R. Paleari, L. Martignoni, D. Bruschi, FluXOR: detecting and monitoring fast-flux service networks, in: Proceedings of the Fifth International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Paris, France, 2008.
[6] S. Yadav, A.K.K. Reddy, A.N. Reddy, S. Ranjan, Detecting algorithmically generated malicious domain names, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC), New York, NY, 2010.
[7] J. Aycock, What's in a name... generator? J. Comput. Virol. 8 (1–2) (2012) 53–60.
[8] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, D. Dagon, Detecting malware domains at the upper DNS hierarchy, in: Proceedings of USENIX Security, Berkeley, CA, 2011.
[9] A. Berger, E. Natale, Assessing the real-world dynamics of DNS, in: Proceedings of the Fourth International Workshop on Traffic Monitoring and Analysis (TMA), Vienna, Austria, 2012.
[10] A. Berger, W.N. Gansterer, Modeling DNS agility with DNSMap, in: Proceedings of IEEE INFOCOM Workshop on Traffic Monitoring and Analysis, Turin, Italy, 2013.
[11] L. Bilge, E. Kirda, C. Kruegel, M. Balduzzi, EXPOSURE: finding malicious domains using passive DNS analysis, in: Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, 2011.
[12] R. Perdisci, I. Corona, G. Giacinto, Early detection of malicious flux networks via large-scale passive DNS traffic analysis, IEEE Trans. Dependable Secure Comput. 9 (5) (2012) 714–726.
[13] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, N. Feamster, Building a dynamic reputation system for DNS, in: Proceedings of USENIX Security, Berkeley, CA, 2010.
[14] M. Kührer, C. Rossow, T. Holz, Paint it black: evaluating the effectiveness of malware blacklists, in: Proceedings of RAID, 2014.
[15] R. Sommer, V. Paxson, Outside the closed world: on using machine learning for network intrusion detection, in: Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, 2010, pp. 305–316.
[16] H. Choi, H. Lee, H. Kim, BotGAD: detecting botnets by capturing group activities in network traffic, in: Proceedings of the International ICST Conference on Communication System Software and Middleware (COMSWARE), New York, NY, 2009.
[17] X. Hu, M. Knysz, K.G. Shin, Measurement and analysis of global IP-usage patterns of fast-flux botnets, in: Proceedings of the Annual IEEE International Conference on Computer Communications (INFOCOM), Shanghai, China, 2011.
[18] R. Villamarin-Salomon, J. Brustoloni, Identifying botnets using anomaly detection techniques applied to DNS traffic, in: Proceedings of IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2008.
[19] T. Holz, C. Gorecki, K. Rieck, F.C. Freiling, Measuring and detecting fast-flux service networks, in: Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, 2008.
[20] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Sov. Phys. Dokl. 10 (8) (1966) 707–710.
[21] F. Casacuberta, M. de Antoni, A greedy algorithm for computing approximate median strings, in: Proceedings of National Symposium on Pattern Recognition and Image Analysis, Barcelona, Spain, 1997.
[22] R. Bayer, Symmetric binary b-trees: data structure and maintenance algorithms, Acta Inf. 1 (1972) 290–306.
[23] L.d. F. Costa, F. Rodrigues, G. Travieso, P.R.V. Boas, Characterization of complex networks: a survey of measurements, Adv. Phys. 56 (2007) 167–242.
[24] M. Knysz, X. Hu, K. Shin, Good guys vs. bot guise: mimicry attacks against fast-flux detection systems, in: Proceedings of IEEE INFOCOM, Shanghai, China, 2011.
[25] C. de la Higuera, F. Casacuberta, Topology of strings: median string is np-complete, Theor. Comput. Sci. 230 (1–2) (1999) 39–48.

Andreas Berger received B.Sc. and M.Sc. degrees in Information and Communication Technology from the Technical University Graz, Austria and a Ph.D. degree in Computer Science from the University of Vienna, Austria. His research interests include traffic monitoring and data analysis, with a focus on malware detection.

Alessandro D'Alconzo received the M.Sc. degree in Electronic Engineering with honors in 2003, and the Ph.D. in Information and Telecommunication Engineering in 2007, from Polytechnic of Bari, Italy. Since 2007 he is a senior researcher at FTW. His research interests range from design and implementation of statistical-based anomaly detection algorithms, to quality of experience evaluation, and application of secure multiparty computation techniques to cross-domain network monitoring.

Wilfried N. Gansterer is an associate professor at the Faculty of Computer Science of the University of Vienna. He received M.Sc. degrees from Vienna University of Technology in Mathematics and from Stanford University in Scientific Computing/Computational Mathematics, respectively, and a Ph.D. degree in Scientific Computing from Vienna University of Technology. His research interests include, among other topics, distributed computing and efficient algorithms as well as Internet security with a focus on botnet detection and prevention and e-mail spam.

Antonio Pescapé is an assistant professor at the Electrical Engineering and Information Technology Department at the University of Napoli Federico II (Italy) and Honorary Visiting Senior Research Fellow at the School of Computing, Informatics and Media of the University of Bradford (UK). He received the M.S. Laurea Degree in Computer Engineering and the Ph.D. in Computer Engineering and Systems, both at University of Napoli Federico II. His research interests are in the networking field with focus on Internet Monitoring, Measurements and Management and Network Security. He is a senior member of the IEEE.