POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

FINDING, CHARACTERIZING AND TRACKING DOMAIN GENERATION ALGORITHMS FROM PASSIVE DNS MONITORING

Relatore:
Prof. Stefano ZANERO
Correlatori:
Dr. Federico MAGGI
Dr. Lorenzo CAVALLARO

Tesi di Laurea Magistrale di:
Stefano SCHIAVONI, matr. 765764

Anno Accademico 2011–2012

To my mother, father,
brother and sister.

Stefano

Abstract

A botnet is a network of compromised machines (bots) under the control a an entity (the botmaster), which uses them to perform illegal activities.

Modern botnets rely on domain generation algorithms, also known as DGAs, to build resilient command-and-control (C&C) infrastructures. Recently, researchers proposed approaches to recognize automatically-generated domains from DNS traffic to infiltrate into such C&C infrastructures and cause the masters to lose control of their bots.

Unfortunately, such approaches require access to DNS sensors whose deployment poses practical issues that render their adoption problematic. Instead, we propose a novel way to combine publicly-available and privacy-preserving databases of historical DNS traffic together with linguistic-based models of the suspicious domains. From this, we find automatically-generated domain names, characterize the generation algorithms, isolate logical groups of domains that represent the respective botnets, and produce novel knowledge about the evolving behavior of each tracked botnet.

We evaluated our approach on millions of real-world domains. Overall, it correctly flags 81.4 to 94.8 % of the domains as being automatically generated. More important, it isolates families of domains that belong to different DGAs. We were also able to verify the validity of our findings against live botnets (e.g., Conficker.B).

Sommario

I crimini informatici sono diventati particolarmente comuni, all'interno di un ecosistema sempre più consolidato e organizzato in cui le botnet (ovvero, reti di macchine compromesse sotto il controllo di una singola entità, il botmaster) continuano a ricoprire un ruolo fondamentale. Di fatto, le botnet si sono trasformate in una piattaforma di valore per attività di lucro online. A questo scopo, sono utilizzate per una moltitudine di attività malevole, dalla diffusione di spam e dal furto di credenziali, fino allo spionaggio.

L'ultimo rapporto ENISA sul panorama delle minacce legate al crimine informatico (Marinos and Sfakianakis, 2012) ha sottolineato come uno degli effetti del fenomeno del malware-as-a-service (MaaS) sia un crescente impiego di piccole botnet distinte, che sostituiscano quelle classiche di grandi dimensioni. Botnet di minori dimensioni possono più facilmente sfuggire ai controlli e non essere rilevate. Sfortunatamente, questo crea la necessità di monitorare un gran numero di minacce molto variegate, seppur con caratteristiche comuni.

L'identificazione di attività malevole legate al funzionamento delle botnet è un problema molto studiato per cui molte soluzioni sono state proposte. La direzione di ricerca più promettente consiste nell'identificare e mitigare il fenomeno delle botnet attraverso l'osservazione e l'ostruzione del traffico nelle loro infrastrutture di comunicazione. Tipicamente, chi si occupa di sicurezza ambisce a identificare gli indirizzi IP e i domini dei server di comando e con-

trollo (ingl., *command-and-control*, C&C) delle botnet. L'obiettivo è quello di isolare (ingl., *to sinkhole*) questi indirizzi IP affinché i bot non siano più in grado di mettersi in contatto con il proprio master.

La strategia più efficace e di semplice realizzazione per l'implementazione di meccanismi di comunicazione centralizzata tra bot e botmaster, prevede l'impiego di punti di incontro dipendenti dal tempo, generati attraverso algoritmi di generazioni di domini (ingl., *domain generation algorithms*, DGA). Il meccanismo dei DGA prevede che i bot e il botmaster concordino un algoritmo pseudo-randomizzato per generare una grande quantità di nomi di domini, che i bot usano per provare a contattare il proprio master. L'idea che rende questo meccanismo di comunicazione efficace e difficile da contrastare è che, in ogni momento, solamente un numero ristretto di domini è effettivamente attivo e rivela l'indirizzo IP del botmaster. Questo rende il *sinkholing* particolarmente costoso. Le autorità, infatti, dovrebbero tenere traccia di parecchie migliaia di domini prima di trovare quelli effettivamente impiegati come punto di incontro per le comunicazioni fra bot e botmaster.

Alcuni lavori di ricerca hanno proposto approcci differenti per riconoscere domini generati in modo automatico (ingl., *automatically-generated domains*, AGDs) con lo scopo di correlare questa informazioni con altri dati (e.g., blacklist, reputazione dei domini).

Gli approcci proposti nei lavori di ricerca hanno principalmente due limitazioni. Nonostante riescano a fornire dettagli approfonditi a proposito di domini malevoli considerati *individualmente* (e.g., Bilge et al. 2011, Antonakakis et al. 2011), non riescono a *correlare* gli abusi, distinti, ma legati uno con l'altro, perpetrati da questi domini. Invece, gli approcci che permettono di correlare in modo automatico i domini malevoli con le botnet per cui sono utilizzati (e.g., Antonakakis et al., 2012) richiedono accesso a traffico Internet la cui raccolta pone delle difficoltà. Più precisamente, la tecnica più avanzata a oggi disponibile (proposta da Antonakakis et al., 2012) richiede visibilità degli indirizzi IP associati alle macchine infette. A parte problemi di privacy, questo crea il problema scientifico delle ripetibilità degli esperimenti

e, più significativamente, richiede sensori di traffico installati fra le macchine infette e i server DNS che esse contattano per risolvere i nomi di dominio, con visibilità sulle richieste DNS originali. Una infrastruttura di raccolta dati del genere è molto difficile da installare (Perdisci et al., 2012).

Con questa tesi, presentiamo Phoenix, un sistema che utilizza l'analisi di traffico DNS pubblicamente disponibile per (1) identificare i domini generati in modo automatico, (2) caratterizzare gli algoritmi di generazione, (3) isolare gruppi logici di domini che rappresentano specifiche botnet e (4) produrre nuova conoscenza a proposito del comportamento di ognuna delle botnet osservate. Phoenix non necessita di conoscenza dei DGA attivi e, in particolare, non richiede sforzi di reverse engineering di malware. Infatti, tipicamente non è disponibili alcuna informazioni sui DGA attivi e sulla loro natura. Essendo basato sull'analisi passiva di traffico DNS, il nostro approccio garantisce ripetibilità scientifica degli esperimenti che involvono malware (Rossow et al., 2012) e preserva la privacy delle macchine infette. Inoltre, supera le difficolà intrinseche nella raccolta di traffico dall'infrastruttura DNS, che pone questioni e tradeoff sul dove collocare i sensori e come processare i dati raccolti.

Phoenix sfrutta l'osservazione che i domini generati automaticamente hanno nomi caratterizzati da alta entropia. Crea modelli linguistici per i domini *non* generati automaticamente (e.g., i domini benigni) e considera generati automaticamente quei domini che vìolano questi modelli. In altre parole, Phoenix automatizza il processo non banale di caratterizzare la "casualità" dei nomi dei domini. Questa operazione è particolarmente difficoltosa quando i nomi dei domini devono essere analizzati singolarmente e non in gruppi.

Inoltre, Phoenix raggruppa gli AGD secondo le loro relazioni dominio-IP, in modo da considerare insieme AGD generati dallo stesso DGA. L'idea dietro questa strategia è che raggruppare istanze di nomi di AGD generati dallo stesso DGA permette di estrarre delle "fingerprint" che caratterizzano l'algoritmo di generazione stessa, senza necessità di fare reverse engineering della sua implementazione. Questo conduce alla classificazione di nuovi do-

mini generati automaticamente. L'identificazione di gruppi di AGD correlati è fondamentale. Identificando queste gruppi, Phoenix fornisce agli analisti conoscenza preziosa, che permette loro di riconoscere, per esempio, gruppi di botnet che implementano algoritmi di generazione con caratteristiche simili—che corrispondono, magari, alla stessa botnet o a un'evoluzione di essa. In questo modo, gli analisti posso seguire l'evoluzione delle botnet e dei loro (magari variabili) server di controllo: dove questi sono collocati, e quante macchine sono utilizzate.

Abbiamo valutato Phoenix su milioni di domini ricavati da database reali. Globalmente, identifica correttamente dall'81.4 al 94.8% dei domini come generati automaticamente. Per di più, isola gruppi di domini generati da diversi DGA. Questo è stato utile, per esempio, quando il giorno 9 febbraio 2013 abbiamo ottenuto una lista riservata di AGD per cui non erano disponibili informazioni su quale fosse la botnet a essi collegata. Phoenix ha attribuito questi domini a Conficker: ulteriori indagini hanno confermato che in effetti si trattava di Conficker.B.

In conclusione, questa tesi dà i seguenti contributi innovativi:

- il nostro sistema identifica gruppi di AGD e modellizza le caratteristiche dell'algoritmo di generazione; rispetto a lavori di ricerca precedenti, il nostro sistema ha requisiti di installazione e funzionamento meno stringenti e più realistici;

- inoltre, il nostro sistema associa nuovi domini malevoli all'attività di botnet conosciute;

- infine, mostriamo come le informazioni sopracitate possano essere usate per generare nuova conoscenza che permetta agli analisti di seguire l'evoluzione nel tempo delle botnet.

Acknowledgements

My deepest gratitude goes to Stefano Zanero, Federico Maggi and Lorenzo Cavallaro, who supervised me during the realization of this research project. I feel extremely privileged to have had the opportunity of working on their side, and I thank them for everything they taught me along the way.

I want to thank my friends, who have always been supporting me, and all my colleagues at Politecnico di Milano, who shared with me countless hours of study during the years. Among these, I should mention the wonderful people of NECST Lab. This thesis is what I have been working on between coffee breaks and foosball games with them.

Last, and most important, thanks to my parents, who taught me everything that makes me the person I am. Everything I *did* or *will* accomplish, I owe it to them.

Stefano

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cybercrime has become very widespread, with an increasingly organized and consolidated ecosystem where botnets (i.e., networks of compromised machines under the control of an entity, the botmaster) continue to play a significant role. As a matter of fact, botnets have become a commodity platform for online lucrative activities. They are used for a variety of malicious purposes, ranging from spamming, information stealing and espionage.

The latest ENISA Threat Landscape (Marinos and Sfakianakis, 2012) highlighted that one of the adverse effects of this malware-as-a-service (MaaS) trend, is an increased number of small, distinct botnets, which are predicted to replace classic large botnets. Smaller botnets are likely to fly under the radar because of their size. Unfortunately, this creates the need of keeping track of such a diverse population of different yet related threats.

Identifying malicious activities related to botnets is a well-studied problem with many proposed solutions. One of the most promising research directions consist in detecting and mitigating botnets by observing and obstructing the traffic of their communication infrastructure. Typically, security researchers and vendors (we refer to them as *defenders*) strive to find the IPs or the domain names of the command-and-control (C&C) server of a botnet with the goal of creating sinkholing IPs: Eventually, the bots will starve not being able to contact their masters.

The most reliable yet easy-to-manage bot-to-master centralized communication mechanism relies on establishing time-dependent rendezvous points through domain generation algorithms (DGAs). In these mechanisms the bots and the master agree on pseudo-random algorithms that generate an overwhelming number of domain names where the bots try to contact their master. The key, which makes these communication mechanisms effective and difficult to deal with, is that, at any moment, only a limited number of domains is active thus revealing the true IP of the master. This characteristic makes sinkholing very expensive for the security defenders, because they would need to track several thousands of domains before finding the one used as the rendezvous point.

Researchers have proposed various approaches for recognizing automatically-generated domains (AGDs) with the goal of correlating this information with other datasets (e.g., blacklists or domain reputation scores).

The existing research approaches have two main shortcomings. Although they provide rich details on *individual* malicious domains (e.g., Bilge et al. 2011, Antonakakis et al. 2011), they fail in *correlating* distinct but related abuses of such domains. The approaches that are able to automatically correlate malicious domains to the underlying botnets (e.g., Antonakakis et al. 2012) require access to Internet traffic whose collection poses issues. More precisely, the state-of-the-art approach proposed by Antonakakis et al. [2012] requires visibility on the infected machines' IP address. Besides privacy issues, this creates the research problem of repeatability and, more importantly, requires a traffic sensor deployed between the infected machines and the DNS servers that they contact to resolve the domain names, with visibility over the original DNS queries. This collection infrastructure is very hard to deploy (Perdisci et al., 2012).

We propose Phoenix, which leverages the analysis of publicly-available passive DNS traffic to (1) find AGDs, (2) characterize the generation algorithms, (3) isolate logical groups of domains that represent the respective

botnets, and (4) produce novel knowledge about the evolving behavior of each tracked botnet. Phoenix requires no knowledge of the DGAs active in the wild and, in particular, no reverse engineering of the malware. In fact, no information is usually available about active DGAs and their nature. Being based on passive DNS traffic, our approach guarantees scientific repeatability of malware experiments (Rossow et al., 2012) and preserves the privacy of the infected computers. Moreover, it overcomes the difficulties of collecting data from the DNS infrastructure, which poses questions and possible tradeoffs on where to place the observation points and how to process the recorded data.

Phoenix leverages the insight that AGD names show high entropy. It creates linguistic models of *non*-AGDs (e.g., benign domains) and considers as automatically-generated those domains that violate such models. In other words, Phoenix automatizes a non-trivial process of characterizing the "randomness" of domain names. This task is particularly challenging when the domain names are considered one at a time, instead of in groups.

Then, Phoenix groups these automatically-generated domains according to their domain-to-IP relations, so to put together AGDs generated by the same DGA. The rationale behind this functioning is based on a the observation that associating instances of AGD names yielded from the same AGD allows to "fingerprint" the generation algorithm itself, without reversing its implementation. This boosts the detection and the classification of newly-seen AGDs. Finding "families," or groups, of related AGDs is fundamental. By identifying these families, Phoenix provides security analysts with valuable insights, which allow them to recognize, for instance, groups of botnets that implement a DGA with common characteristics—which refer, possibly, to the same botnet, or an evolution. Therefore, the analysts can follow the evolution of these botnets and their (changing) C&Cs over time, where these are hosted, and the number of machines involved.

We evaluated Phoenix on millions of domains drawn from real-world datasets. Overall, we were able to correctly recognize 81.4 to 94.8% of the domains as being AGDs. More importantly, Phoenix isolated families of do-

mains that belong to different DGAs. Also, on February 9th we obtained an undisclosed list of AGDs for which no knowledge of the respective botnet was available. Phoenix labeled these domains as belonging to Conficker: Further investigation eventually confirmed that it was indeed Conficker.B.

In summary, this thesis makes the following original contributions:

- our system can identify groups of AGDs and model the characteristics of the generation algorithms; with respect to previous work, our system has less restrictive and more realistic requirements for the deployment;
- also, our system can automatically associate new malicious domains to the activity of known botnets;
- last, we show that the above findings can be used to build new correlated knowledge that allow security analysts to track the evolution of botnets.

The reminder of this work is structured as follows.

- In **Chapter 2** we give an overview of the botnet phenomenon, together with the preliminary concepts needed to understand it. We then cover in details the command and control of botnets. In particular, we focus on the architecture of the C&C channel and we comment about the importance of its design in the context of building resilient C&C infrastructures. On this basis, we describe the evolution of the C&C architectures in response to increasingly tight resilience and security requirements. Finally, we introduce the idea of DGAs, which leads to the state-of-the-art C&C architectures.
- In **Chapter 3** we cover in details the literature, which aimed at answering research questions related to ours. In particular, we discuss the work that focuses on detecting DGA-based botnets active in the wild and point out the major shortcomings. We conclude the chapter by

acknowledging the need of overcoming such limitations by introducing Phoenix.

- In **Chapter 4** we introduce Phoenix with a top-down approach. First, we describe the high-level modular architecture and clarify the role of each module. Then, we give details about the internal functioning of each module, explaining the rationale behind every strategy adopted. At the end of the chapter, we describe the implementation of our prototype of Phoenix.

- In **Chapter 5** we describe the evaluation strategies we adopted with the (far from trivial) purpose of validating Phoenix: A quantitative validation of the single components and a qualitative validation of the whole approach. We then give the results of such experiments, and we comment on them.

- In **Chapter 6** we discuss the limitations of Phoenix and we comment on possible future work, which may overcome the limitations and continue our research. Finally, we draw some conclusions.

Chapter 2

Botnets Command and Control

In this chapter we cover the preliminary concepts needed to understand the botnet phenomenon and we give an overview of the phenomenon itself.

We then go in details in describing the *command and control* of botnets by introducing the concept of *command-and-control channel* and emphasizing its importance in the context of botnet architectures. We observe how the C&C channel represents a single point of failure for the botnet, and thus we underline the importance of its design. On this topic, we cover the evolution of the C&C architectures in response to increasingly tight resilience and security requirements.

At the end of the chapter we introduce the idea of domain generation algorithms, which allows the state-of-the-art command-and-control architectures employed by botnets active nowadays.

2.1 Preliminary Concepts: The DNS

The domain name system (DNS) (for reference, see Mockapetris 1987a,b) is a very basic and significant component of the Internet infrastructure. Its main purpose is to provide a naming system to identify computers, services or resources connected to the Internet or a private network. For instance, by employing the DNS, Internet users can reach the services of Google using

the humanly-readable alias (i.e., domain name) `google.com`, instead of the numeric IP address `173.194.35.37`.

With the term DNS we identify both the hierarchical distributed database storing the mappings between aliases and addresses, and the UDP-based protocol used by end-users to query such database. In the following of this section, we give some details about both, together with a definition of *domain name* and other related terms.

2.1.1 Domain Names

A domain name is a sequence of hierarchical labels separated by dots (e.g., `www.example.com`). Specifically, the rightmost label (e.g., `com`) is referred to as top-level domain (TLD), the second rightmost label (e.g., `example`) as second-level domain and so on. TLDs can be either generic top-level domains (gTLDs), as in the case of `com`, `org` or `info` or country code top-level domains (ccTLDs), as in the case of `it`, `fr` or `uk`. Different authorities regulate the registration of domain names featuring different TLDs.

Not all second-level domains can be registered by Internet users. For instance, although it is possible to register the domain `example.com`, it is not possible to register `example.uk`. This happens as `uk` ccTLD authority mandates user domains to be registered under `co.uk`, leaving `*.uk` domains reserved for different purposes. To handle cases like this with ease, we introduce the term effective top-level domain (eTLD), also known as *public suffix*[1]. An eTLD is a suffix under which Internet users can register a domain name. Examples of eTLD are `com` and `co.uk`.

2.1.2 DNS Distributed Hierarchy

The DNS is a hierarchically-distributed database, meaning that there is no single server storing all the domain name-IP mappings. This, in fact, would lead to problems such as the following (Kurose and Ross, 2006).

[1] `http://publicsuffix.org/`

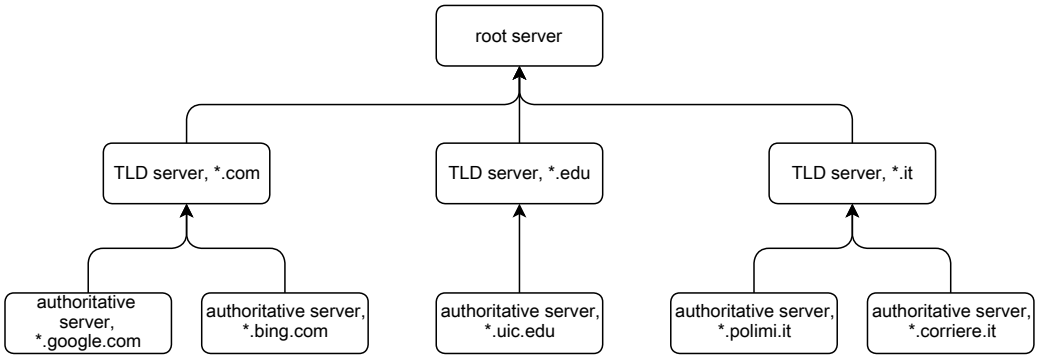

Figure 2.1: *DNS servers exemplified hierarchy.*

Single point of failure The failure of a centralized server would hit the availability of the whole Internet.

High traffic volumes The traffic volume to process on such centralized machines would be overwhelming.

Geographical distance Latency issues would arise for those users geographically distant from the DNS server.

Mantainance A single server would need to store all the records related to all Internet hosts (domain-IP mappings). This would require high update rates of the mappings.

To face these problems, the DNS is both replicated (to handle huge amounts of traffic), and hierarchically organized, so that each DNS server is responsible for the resolution of a limited and well-defined set of domains.

The DNS server hierarchy is organized as depicted in Figure 2.1 and features three levels.

Root server As of today, 13 root DNS servers (actually, they are clusters of failover servers) exist. These servers do not store any domain-IP mapping but are instead employed to point to TLD servers.

TLD servers Each TLD server is used to resolve domain names under its jurisdiction. For instance, the `com` TLD server will be queried during the resolutions of all `*.com` domains.

Authoritative server Each company or institution having hosts publicly accessible through the Internet runs an authoritative DNS server. For example, every domain `*.google.com` is resolved through an authoritative DNS server under the control of Google. This guarantees fast updates of the domain-IP mappings.

When a user wishes to resolve a domain name, he explores the above hierarchy top-down, as exemplified in the following section.

2.1.3 DNS Querying Procedure

Figure 2.2 depicts the interactions that are carried out when an host wishes to resolve a domain name (e.g., `www.google.com`).

First, the requesting host contacts a DNS resolver with a DNS query for the desired domain (step 1). The DNS resolver is the DNS local server (also known as *default name server*) responsible of handling DNS requests originated from the subnetwork where it resides. It is responsibility of the DNS resolver to obtain the IP address associated to `www.google.com` and return it to the querying host, guaranteeing the transparency of such procedure.

Then, the DNS resolver, parses the domain name requested, and identifies the TLD. In our case, this would be `com`. It then queries (step 2) a root DNS server to obtain the IP address of a TLD DNS server for such TLD (step 3). Now, the DNS resolver queries this server (step 4) to obtain the IP address of an authoritative DNS server for `*.google.com` domains. The TLD server replies (step 5) with the address of the authoritative DNS server.

Finally, the DNS resolver contacts `google.com` authoritative DNS server (step 6) to resolve `www.google.com`. The authoritative server *knows* the IP address associated to that domain, and returns it (step 7). At this point, the

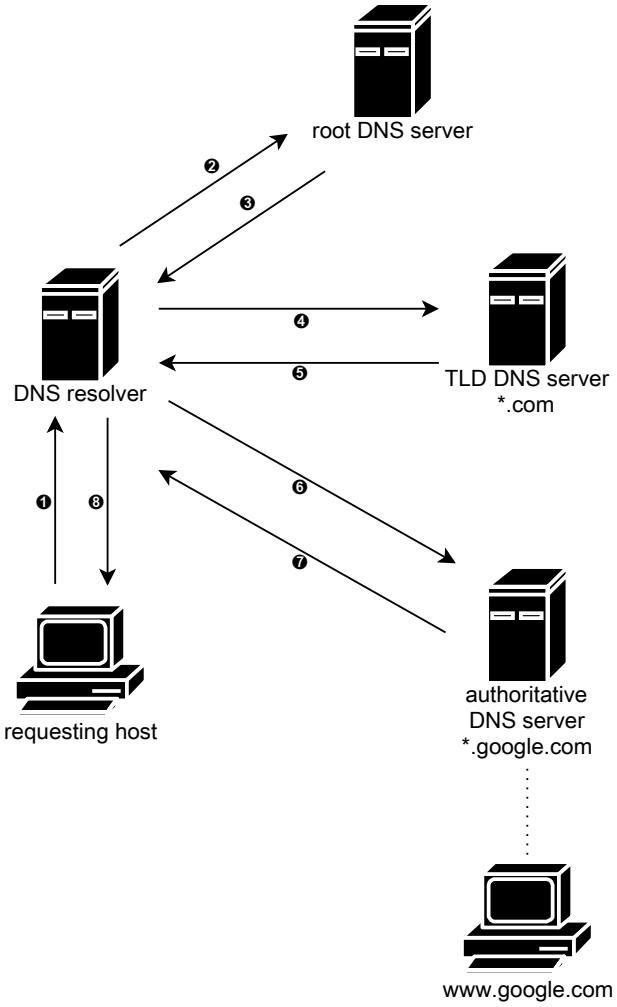

Figure 2.2: *Resolution of domain name* `www.google.com`.

DNS resolver completes the querying procedure and returns (step 8) the IP address associated to `www.google.com` to the requesting host.

Clearly, this is a basic scenario, which is rarely verified in practice. Most of the transactions, in fact, do not usually take place thanks to caching mechanisms, which allow to store the results of previous DNS queries so to not repeat the same queries continuously. For instance, the DNS resolver may not need to contact (every time) a root DNS server to obtain the address of the `com` TLD server.

Another important remark, useful for understanding the next chapters, is that no DNS server, beside the DNS resolver, has visibility over the IP address of the requesting host. This, in fact, is irrelevant to complete their tasks.

2.2 Introduction to Botnets

With the term *botnet* we refer to a network of malware-compromised machines, which receive and respond to the commands of a server (i.e., the *C&C server*). The role of the server is to dispatch the instructions of a human controller, the *botmaster*. With the purpose of evading detection, the botmaster can hide behind proxy machines (the so-called *stepping-stones*) while interacting with the C&C server, as shown in Figure 2.3. The bots are infected with a malicious program (the *bot*) and are used to carry out malicious activities or attacks on behalf of its controller (e.g., spamming, phishing, service disruption, credentials stealing).

In the following of this section we go in details in analyzing different aspects of the botnet phenomenon.

2.2.1 Purposes

The main purpose of recruiting thousands of machines under the control of a botmaster is to carry out malicious activities. In fact, miscreants can derive

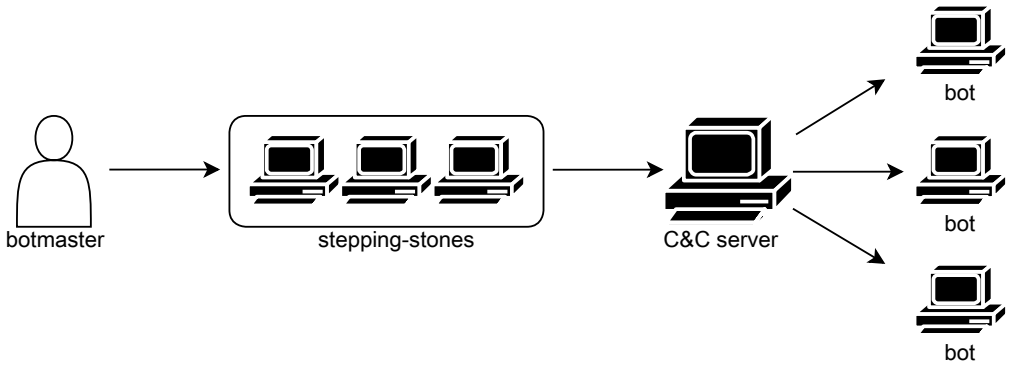

Figure 2.3: *Basic structure of a botnet.*

large financial gains from using the machines of innocent and unaware users for their malicious purposes, while remaining substantially hidden.

In the rest of this section, we give *some* examples of such malicious activities, although modern botnets are designed to fulfil a wide range of purposes at different times, according to the instructions received by the bots. This leads to the creation of the so-called MaaS business model, in which botmasters lease the service of their botnets for different purposes, on request.

Information Gathering The bot binaries can be employed to harvest private data from the compromised machines. In the simplest case, these data may include passwords, email addresses useful for spam, credit card, bank account or social security numbers. In most advanced scenarios, the bots can be used to steal intellectual properties or to spy on users, companies or even rival nations.

Distributed Computing Remote compromised machines can be use as distributed storage or as a grid to perform heavy computations. Examples range from hosting of copyright-protected files to distributed password or CAPTCHA cracking. These computational activities can be particularly stealthy, as are usually carried out on the compromised machines while they

are unattended by their owners (e.g., when the screen saver is active, or when the machine appears in sleep mode).

Network Service Disruption Botnets can be employed to carry out distributed denial of service (DDOS) attacks. In this scenario, the bots are instructed to overwhelm victim services by sending huge amounts of requests over a short period of time. This inevitably causes systems to crash and to be unavailable to legitimate users, with a consequent economical loss. Being able to launch DDOS ma be also used for cyber-extortion as large companies prefer to pay money, rather than lose the availability of their services and thus their credibility.

Spamming The activity of spamming (i.e., unsolicited marketing) requires to send large amount of emails to advertise malicious services or frauds. Performing such activities is not trivial, as security personnel blacklists the sources of such unsolicited emails. In this scenario, spammers can leverage the bots and use them as mail server proxies. This guarantees that spamming emails are originated from thousands of different sources, making the blacklisting not effective.

Malware Spreading The malware installed on the remote machines can be used to fetch and propagate other malware. This is the case, for instance, of *droppers*, whose only purpose is to piggyback the installation of other malware on the compromised machines.

2.2.2 Propagation

Botnet-related malware can propagate (as all malware does) in different ways. The bot binary can be instrumented to propagate without user intervention, with tactics similar to the ones employed by worms. When this is the case, the bot binary performs *scanning* operations, in which the machines on the same network of the bot are probed looking for open ports or known vulnerabilities,

which may allow the bot to install clones of itself on the newly-discovered machines.

In different scenarios, instead, botnet-malware can need (unwilling) user intervention to be installed on target machines. This is the case of malware distributed through drive-by download (i.e., downloaded and installed when an user visits a compromised website), or through infected media (e.g., infected USB flash drives).

2.2.3 Topologies

Botnets can exhibit different network topologies, depending on how the bots receive commands from the C&C server (and thus from the botmaster).

Centralized In a centralized topology, bots communicate directly with their botmaster. By far, this is the simplest implementation possible, as it relies on a well-establish client-server architecture. Moreover, it allows low latency communications and guaranteed packet delivery, which is essential to syncronize the infected machines. This may be needed, for instance, when the botmaster wishes to launch a coordinated attack.

Centralized architectures suffer from the presence of a single point of failure (i.e., the C&C server). Nevertheless, miscreants began using DNS to make centralized botnet infrastructures more reliable and, more important, harder to map and take down (Antonakakis et al. 2011, 2012, Holz et al. 2008a, Passerini et al. 2008, Perdisci et al. 2012, Yadav et al. 2012). Such architectures, which are the preferred choice for nowadays botnets, are covered in depth in Sections 2.4 and 2.5.

Examples of botnets featuring centralized topologies are: PushDo (Decker et al., 2009), Torpig (Stone-Gross et al., 2009a) and Rustock (Chiang and Lloyd, 2007).

Decentralized In distributed topologies, there is no obvious master-slave relation between bots and C&C servers. This can be achieved in many ways.

First, C&C servers can be replicated and spread over different geographical locations. This way, the failure of a single C&C node has limited impact on the operation of the whole botnet. Moreover, bots can be organized in a peer-to-peer (P2P) network, in which commands can enter the P2P network at any node and are broadcasted following unpredictable routes between the infected machines.

It is clear that implementing such decentralized topologies is far from trivial. In fact, it requires the implementation of ad-hoc protocols, which can guarantee connectivity between the botmaster and *all* the compromised machines, possibly through multi-hop routes. Moreover, adopting such decentralized architectures may cause C&C communications to experience severe and unpredictable delays.

Examples of botnets with decentralized topologies include Storm (Grizzard et al. 2007, Holz et al. 2008b), Nugache (Stover et al., 2007) and Waledac (Stock et al., 2009).

This work deals with centralized botnet topologies, which are the most widespread nowadays. This is the case because botmasters can achieve high resilience of the communication architectures while keeping their design reasonably simple.

2.3 Command-and-control Channel

The command-and-control channel is the logical communication channel established over the Internet between a bot and its botmaster (see Figure 2.4). Its role is to allow messages to flow between the malware running on the infected machine and the C&C server under the control of the miscreant architect of the botnet.

The communication is usually started by the bot, and follows a so-called *rallying* phase in which the bot discovers the *rendezvous* point (i.e., the coordinates where to contact its C&C server). The botmaster can not be

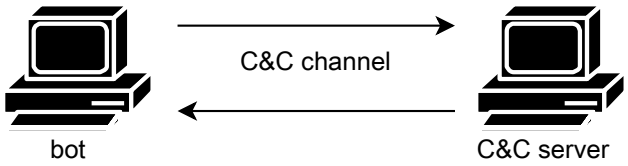

Figure 2.4: *C&C channel.*

the one initiating the communication because he is usually not aware of the addresses of the infected machines under his control.

2.3.1 Communication Protocols

Both transport-layer protocols TCP and UDP can be employed during the communication. Usually, TCP is chosen as it gives an abstraction from network details by providing a reliable packet delivery, easing the implementation of the malware and the C&C server.

At application level, the author of a malware can choose to employ a standard protocol (e.g., HTTP) or to implement its own. The disadvantage of the second choice with respect to the first is that custom-made protocols generate network traffic easily distinguishable from that of standard protocols, making the detection of ongoing C&C communications easier.

In the past, botmasters used to strongly rely on the simple IRC protocol to exchange information with their bots, as it proved itself to be usable with ease for the purpose. Nevertheless, the improvements of security defenders in monitoring the IRC channels and detecting botnet-related communications, pushed miscreants to start using HTTP and exploiting its large use to better masquerade their activities. As of today, HTTP is widely used in C&C communication, possibly in its version over SSL (i.e., HTTPS). The latter option allows encrypted communication between bot and botmaster, making eavesdropping of network traffic ineffective for the detection of malicious activities.

2.3.2 Use Cases

The C&C channel is logically bidirectional, meaning that there is a flow of meaningful data between botmaster and bot in both directions.

The kind of information that is transferred over the C&C channel strongly depends on the purpose of the botnet. For example, when the malware on the infected machine is used to harvest sensitive data, communications take place to transfer stolen data, which may include passwords, credit cart numbers or other credentials, from the bot to the botmaster. When instead the bots are used as proxies to remotely launch network service disruptions attacks (e.g., DDOS), or as a computational grid to perform computational-expensive operations (e.g., CAPTCHA breaking), data flow from the botmaster to the bot, carrying the specifications of the tasks to be performed on the compromised machines.

Obviously, by no means these use cases are disjoint: It is common that botnets are multifunctional and can be used for all the tasks mentioned—and more—on request.

Generally speaking, it should be understood that the existence of the communication channel is fundamental for the botmaster to use the infected machines for his malicious purposes, thus being able to monetize the development of the malware.

2.3.3 Single Point of Failure

The importance of the C&C channel in the overall botnet infrastructure and operational activities translates in it being the single point of failure of the whole system. In particular, an obstruction of the communication channel could lead to a temporary or even permanent loss of connectivity between bot and botmaster. When the latter event occurs, the malware on the infected machine stops receiving instructions by its master and turns dormant. The machine remains infected, but the malware becomes innocuous. Basically, a botmaster *loses* the bot with which he can no longer establish a commu-

nication: Only a new infection with a new malware—not through the C&C channel—could bring the machine back under his control.

It is no surprise that defenders concentrate their efforts in blocking C&C communications as a medium to disable botnets. This strategy, in fact, is the only viable option to tackle the threat without the need of sanitizing each individual infected machine.

It is thus clear that, given the efforts and the investments needed to have a malware propagate over a large number of machines, what a botmaster fears the most is the possibility of having its C&C channel obstructed and losing the control on the infected machines. For this purpose, a number of increasingly effective rallying mechanisms have been employed in the years by miscreants to build bullet-proof C&C channels.

2.3.4 Threat Modeling

From the botmaster perspective, defenders represent a threat. It is thus fundamental for him to understand precisely the threat model, so to build a C&C infrastructure resilient to possible interventions.

Before analyzing these possible interventions (i.e., what defenders can *do*), it is worth commenting about the assumptions of what defenders *know*. The first assumption considers the question of defenders owning a sample of the malware family used to infect the bots. It is known that modern malware propagates in a way to avoid infecting machines under the control of security vendors or research centers, so to remain undiscovered as long as possible. Nevertheless, it safer and conservative for a miscreant to assume that defenders have the availability of the malware binary.

The fact that defenders may posses a malware sample is obviously not a threat *per se*, but it raises the question of whether the malware can be successfully analyzed. Studying a malware, in fact, can disclose useful information (e.g., the communication protocol or the rallying mechanism employed by the bot to communicate with the botmaster). This information can be leveraged by defenders to design proper countermeasures.

Malware authors can use a multitude of techniques to make the analysis of the binary complex or even not feasible. Some of these anti-forensic techniques prevent static analysis of the binary code (i.e., analysis of the malware sample without executing it). These techniques range from code obfuscation to code encryption, which both harden the reverse engineering process. Other anti-forensic strategies target binary dynamic analysis (i.e., observation of the binary running in a sandboxed and instrumented environment). Such strategies include, for instance, instructing the malware not to execute under suspect conditions, like a virtual machine.

Despite what has been said, a malware author has no guarantee that the malware he engineered will not be successfully reversed and understood by the defenders: It is thus safer to suppose it eventually will. To summarize, while designing the C&C infrastructure of a botnet, the miscreant should consider his opponents, the defenders, as in possess of a sample of the binary of the malware and perfectly able to understand the behavior coded in it.

Given these preliminary assumptions about the level of knowledge of the defenders, it is interesting to point out the kind of operations they can put in place to disrupt the C&C channel of a botnet they are targeting: takeover and sinkholing.

Takeover With the term *takeover* of a botnet, we refer to the act with which a defender takes the control over an existing botnet, by excluding the botmaster from the command. This operation may be performed in a variety of ways, which will be clear later on, and is concluded with the defender redirecting the C&C channel to point to some machines under his own control, instead of to the C&C server under the control of the botmaster. In this scenario, the defenders deploy machines, which mimic the behavior of the C&C server under the control of the botmaster, and performs an hijacking on the C&C channel (see Figure 2.5). When the bots start contacting the server under the control of the defenders, they are instructed to act innocuous, so that the overall botnet is disabled.

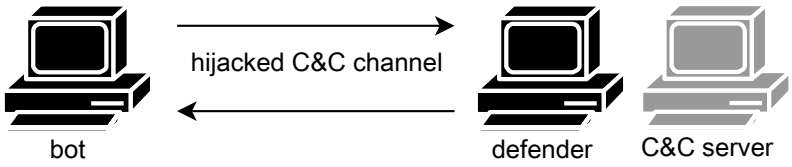

Figure 2.5: *C&C channel hijacking.*

In the literature, there is a well known example of successful takeover: Stone-Gross et al. [2009a]. In 2009, researchers from the University of California at Santa Barbara managed to take control for ten days of the Torpig botnet, a network of about 180,000 compromised machines used to harvest private credentials from target users. The researchers hijacked the C&C channel and started communicating directly to the malware-infected machines belonging to botnet, mimicking the legitimate botmaster. For ten days, they instructed the remote machines not to perform any malicious activity and they collected information to estimate the diffusion of the Torpig malware.

Taking over a botnet is for sure the most ambitious result a defender can hope to accomplish. Although, this could be provably not possible, if the miscreant secures his malware properly.

Botnet takeover is predicated upon the possibility that defenders instrument a machine under their control to mimic the behavior of the C&C server. This way, bots can be fooled in believing they are in contact with their botmaster, even if they are not. To summarise, botnet takeover is possible under the condition that the malware infecting the remote machines does not authenticate the botmaster before exchanging any message with him. This, in fact, was the condition that led to the Torpig takeover: A broken authentication in the C&C mechanism of the botnet.

Unfortunately for the defenders, a sound implementation of a proper authentication mechanisms (i.e., asymmetric key authentication, with the botmaster owing a private key and the bots the corresponding public key) is enough for a malware author to ensure that no C&C hijacking will take place, thus that nobody—beside the legitimate botmaster—will ever control

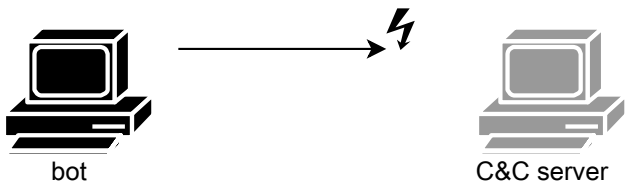

Figure 2.6: *C&C channel sinkholing.*

the infected machines belonging to the botnet. This remains true under our hypothesis that defenders have a complete knowledge of the malware binary, thus of the public key.

For the above reasons, in this work we do not consider the option of defenders doing a takeover of a botnet, as it is not always feasible. Instead, we focus on other operations that defenders can deploy to leave malware authors with no prompt response.

Sinkholing With *sinkholing* we refer to the operation with which defenders fight the creation of functioning C&C channels, preventing the botmaster from communicating with the compromised machines under his control (see Figure 2.6). This is usually done by means of blacklisting the IP addresses hosting the C&C servers or the malicious domains, which point at them, thus preventing network packets originating from the bots to reach the botmaster.

Sinkholing does not require defenders from mimicking the botmaster, but can be equally effective in disrupting the operations of a botnet. The botmaster must take care of designing C&C architectures resilient to sinkholing and, differently from the case of takeovers, there is no immediate way for them to address the issue.

2.4 Basic Rallying Mechanisms and Security

The resilience of the C&C channel to sinkholing depends from the rallying mechanism employed (i.e., the strategy bots use to identify the rendezvous point with their C&C server). Different approaches can be taken for this

purpose. Each of them guarantees the fulfilment of certain requirements while showing some weaknesses. These weaknesses are those exploited by defenders in their effort of mitigating the botnet-related threats.

In the reminder of this section we go through different rallying mechanisms that botmasters can employ to be immune to sinkholing. Our presentation lists rallying mechanisms in order of their increasing sophistication and leads to the introduction of the state-of-the-art approach employed by botnets active nowadays.

2.4.1 Hardcoded IP Address

A bot can simply *know* the IP address where to contact its C&C server (see Figure 2.7). This information can be contained in the data segment of the binary, can be read from an external file or can be the result of a deterministic and internal computation over some initialized data. In any case, the IP address is shipped with the malware.

Small variations of this paradigm are possible. A malware can leverage *a list* of IP addresses instead of just one. Moreover, the IP address (or list of addresses) does not need to remain constant over time. While communicating with its botmaster, in fact, a malware can receive updates of IP addresses to use in the future with a migration-by-delegation mechanism. This guarantees a degree of mobility to the botmaster who can change the address of its C&C server without losing connectivity with its bots.

In all the cases described, at any moment in the lifetime of a bot, it knows the "coordinates" of the rendezvous point (i.e., the IP address) to use in the next communication with the botmaster. This information is contained in the binary of the malware (or in its dependent files) and can be extracted—no matter how difficult this operation can be. In other words, the malware *leaks* the sensitive coordinates of the rendezvous point.

The latter observation leads to the conclusion that in any moment there exist a rendezvous point (an IP address, or a short list of addresses), which is single point of failure for the botnet. Sinkholing the IP addresses associated

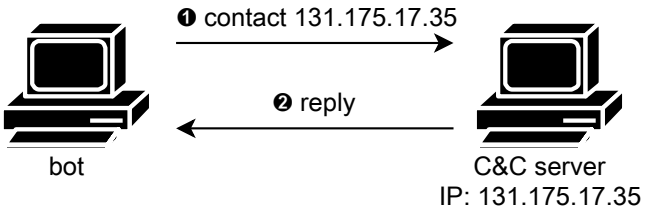

Figure 2.7: *Rallying mechanism with harcoded IP address.*

with that specific rendezvous point would lead to the malware not being able to contact its C&C server, and not being able to recover in any way its connectivity with it. In other words, a timely and punctual intervention of defenders in blocking the IP addresses under the control of the botmaster would lead to a breakdown of the whole botnet.

The rallying mechanism described clearly shows some weaknesses as it provides little flexibility for the botmaster and poor resilience to defenders interventions.

2.4.2 Hardcoded Domain

As mentioned briefly in Section 2.2.3, the botmaster can introduce a degree of freedom in the management of the C&C channel by leveraging the DNS protocol. In this case, the bots are instrumented to have a domain (or a list of domains, possibly updated over time with a migration-by-delegation) to query to obtain the IP address of the C&C server (see Figure 2.8).

In this architecture, the botmaster achieves more flexibility of management of the C&C server, being able to move it across different networks while having the rendezvous domain always pointing to the correct and updated IP address. This strategy guarantees the ineffectiveness of any operation of IP sinkholing that the security defenders may put in place. In fact, when the botmaster realizes that he lost connectivity with the bots under control (because of an IP sinkholing), he can simply move the C&C server to another IP

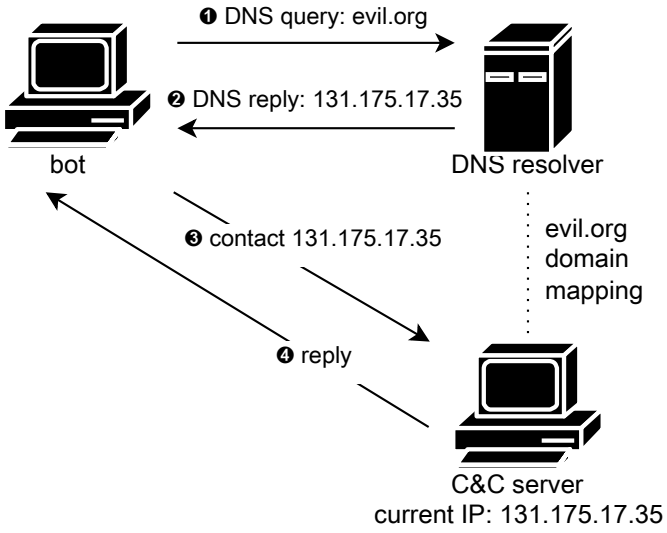

Figure 2.8: *Rallying mechanism with harcoded domain.*

address, update the mapping of the domain on the authoritative DNS server and restore connectivity with the bots.

Moreover, using a domain instead of an IP address allows to create a so called fast-flux service network (FFSN) (Holz et al. 2008a, Passerini et al. 2008). In this scenario, the DNS mapping of the rendezvous domains is changed with high frequency to point to throw-away C&C servers, which can be located on the infected machines themselves. This way, the botmaster hides himself, using the bots as relays for his commands. Moreover, network traffic analysis aimed at detecting C&C channels is hardened, as the C&C servers appear quickly moving.

Introducing domain name resolution through DNS is effective for the variety of reasons described. Nevertheless, the problem of having at any time a single and identifiable point of failure stands. Malware analysis, in fact, can lead to the disclosure of the rendezvous domain currently used for rallying. Sinkholing is now possible at DNS level targeting the rendezvous domain.

It is worth pointing out that, despite the fact that the use of rendezvous domains shows similar theoretic weaknesses to the use of rendezvous IPs, the

operation of domain sinkholing is much harder on the defender side, as it requires interaction with Internet authorities less prone to security interventions. Moreover, recall that the DNS protocol does not allow fast domain revocation (due to caching issues). Even worse, Jiang et al. [2012] proved in their work that the large majority of DNS implementations are affected by a vulnerability with allows domain names to remain resolvable long after they have been revoked.

2.4.3 Design Flaws and Requirements Statement

The rallying mechanisms described so far appear different in their implementations. Nevertheless, they show some common design flaws, which make them weak in responding to sinkholing operations performed by security defenders. We can summarize these flaws as follows.

Information leaks The analysis of the malware leads to precise information about the rendezvous point between bot and botmaster. We stated previously that the malware-reversing process can be extremely difficult and time-consuming, so the analysis is by no means straightforward. Nevertheless, if successfully performed, it leads security researchers to precise defence operations, which may disable a botnet by simply sinkholing a set of IP addresses or domains.

Applying the terminology of information security, we can say that the rallying mechanisms described in the previous sections are *secure by obscurity* instead of *secure by design*. Malware authors, in fact, would need to rely on defenders not being able to analyze the binary sample for their C&C channels to be resilient.

Weak migration strategy We mentioned that rendezvous points (both IP addresses and domains) can migrate over time and we made clear that migrating can be particularly helpful in response to sinkholings. The rallying strategies described so far rely on a migration-by-delegation

mechanisms: Once a C&C channel is establish between bot and botmaster, the botmaster can provide new rendezvous information to the bot for further communications. In this scenario, if a security defender is able to obstruct the C&C channel before the agreement takes place, the bot ignores the coordinates of futures rendezvous points. It is then not able to contact his C&C server and remains dormant.

The reason of the aforementioned weakness is because migrations require an explicit agreement between the two parties involved in the communication.

The statement of the design flaws leads to a definition of design requirements for more resilient and effective rallying mechanisms.

Agnostic malware binary The malware should be designed in a way to leak no information about future rendezvous point between bot and botmaster. This translates in the malware being agnostic of such information.

Theoretically speaking, this requirement is difficult to fulfil. In particular, we are requiring to be able to establish a communication between bot and botmaster, thus have two entities *meet*, with one of them having no knowledge about *where* this will happen.

Agreement-less migrations Bot and botmaster should be able to agree on a future rendezvous point without explicitly acknowledging each other by means of a communication. This ensures that the bot is never left in an incoherent internal state where he has no knowledge of how to communicate with his botmaster.

From the requirements above, the state-of-the-art rallying mechanisms based on the so-called *domain generation algorithms* is brought to life (Antonakakis et al. 2012, Stone-Gross et al. 2009a).

2.5 Domain Generation Algorithms

Domain generation algorithms (DGAs) have become the most common technique[2] for implementing effective rallying mechanisms.

In this mechanism, the bots and the C&C server implement a common algorithm to generate every day[3] a large and time-dependent list of domain names. These domains—we will refer to them as automatically-generated domains (AGDs)—are pseudo-randomly generated based on unpredictable seeds (e.g., Twitter trending topic, results of specific Google searches). Only one (or a few) of these AGDs is actually registered by the botmaster and pointing to the true IP address of the C&C server (see Antonakakis et al. 2012, Stone-Gross et al. 2009a). The bots query all the AGDs until the DNS resolver answers with a *non*-non-existent domain (NXD) DNS reply, containing the IP address of the C&C server. The process is depicted in Figure 2.9: The bot tries to resolve the AGDs `flufnm.info` and `ymhej.biz`, without success; it then queries the AGD `yxzje.info` and obtains the IP address `131.175.17.35` hosting the C&C server.

The process described is repeated periodically. Every day different AGDs are generated, and the rendezvous domain (i.e., the *registered* AGD) varies.

2.5.1 Security Analysis

The DGA-based rallying mechanism, despite being easy to explain and to implement, shows a certain degree of sophistication. In particular, it allows the fulfilment of the requirements listed in Section 2.4.3, making DGA-based C&C channels resilient to possible interventions of security defenders. This remains true under the conservative hypotheses we stated in Section 2.3.4.

[2] `https://blog.damballa.com/archives/1906`

[3] We assume the time unit of the *day* for the sake of clearness.

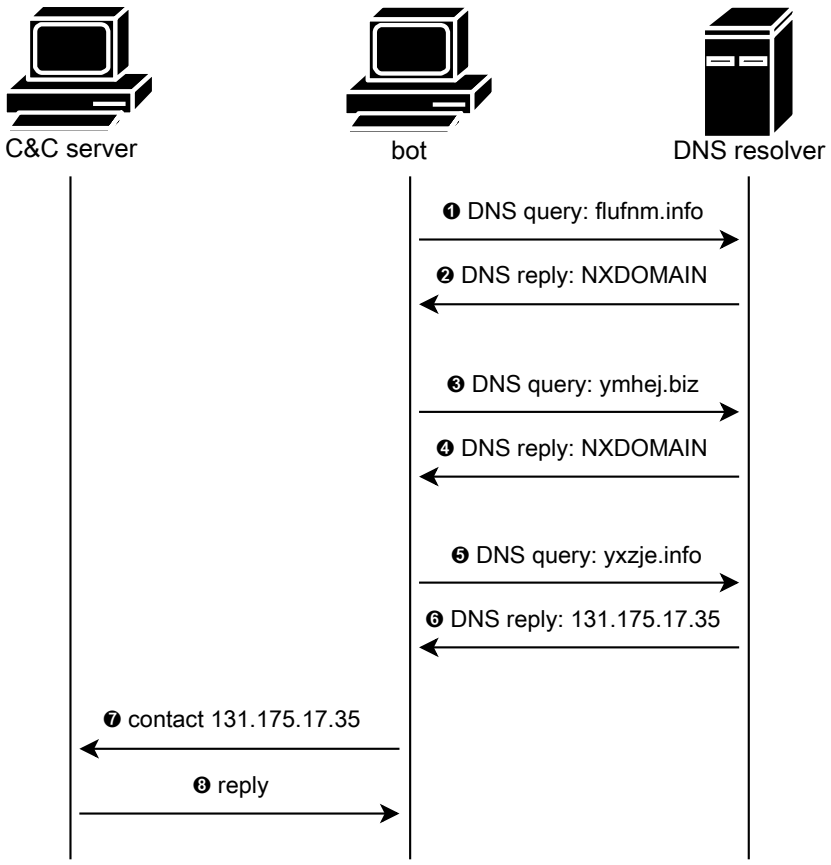

Figure 2.9: *Example of execution of DGA-based rallying mechanism.*

2.5.1.1 Agnostic Malware Binary

By design, the malware binary code implementing the DGA functionalities leaks poor information about future rendezvous points between bot and botmaster. This *agnosticism* is the result of two (possibly combined) design decision and is not affected even if we consider the case of defenders having access to the (deobfuscated) source code of the malware.

External Randomization Seeds DGA-based malware employ external and time-dependent data (e.g., Twitter trending topic) to seed its randomization algorithm. A defender wishing to predict the behavior of the bot (e.g., to disclose future rendezvous domains) would need to access such data. This is obviously not possible, as it would require the prediction of future phenomena. Generally speaking, malware authors succeed in making the reconstruction of realistic malware execution conditions impossible. As a consequence, answering the question *"What will it be the rendezvous domain tomorrow?"* is never possible. Not even the malware author can answer such question.

Vague Rendezvous Point Disclosure Even in the case of a malware implementation featuring no external and unpredictable sources, it is still impossible for security defenders to predict with precision the coordinates of future rendezvous points. This makes the design of proper countermeasures a difficult task.

When correctly reverse engineered, a binary sample exposes an overwhelming amount of candidate rendezvous domains, which are equally likely to be chosen by the botmaster to establish C&C connections and thus registered. For instance, Conficker.C's DGA generates 50,000 AGDs daily. This leads to an **asymmetry of cost and efforts** between botmaster and defender. In fact, it is sufficient for a botmaster to register *one* domain to contact its bots, while defenders would need to sinkhole (or register, with the relative fees) *all* the domain pool, to avoid it.

To summarize, the information leaked by the analysis of the malware sample is insufficient to design any proper and effective countermeasure.

2.5.1.2 Resilient Migration Strategy

A DGA-based rallying mechanism allows resilient migrations of C&C channels, as it does not requires migration-by-delegation approaches.

Bot and botmaster evolve with identical but decoupled dynamics: They both execute the same time-dependent code and can syncronize without exchanging messages. Every day, in fact, both bot and botmaster know the updated candidate rendezvous domains. The botmaster can proceed registering a few and pointing them to the C&C server, while the bot tries to resolve them by means of DNS queries. This procedure can be carried out day after day and it is not predicated on the C&C communications being actually established every day. In fact, we can say that the bot resets daily its operational state and never finds itself in the condition of not knowing how to contact his botmaster in the future. Since the bot behaves as described, it can not happen that it permanently loses connectivity with its C&C server, no matter how many days the communication between the two fails.

From a botmaster standpoint, DGAs allow extreme flexibility of C&C communications. Indeed, the botmaster can move its C&C server across networks by simply changing the mapping in the DNS resolutions of the AGDs, without worrying of the consequences of possible downtimes. This way, the miscreant can migrate its servers with the purpose of evading the blacklisting and sinkholing of the IP addresses he employed. A situation like the one described is depicted in Figure 2.10, where the bot is able to contact its C&C server even if security defenders keep on sinkholing the IP address where it resides.

In a DGA-based network, there is no operation security defenders can put in place—no matter how hard—to permanently disconnect a botmaster from its bots. Indeed, every operation of C&C obstruction, which we already ex-

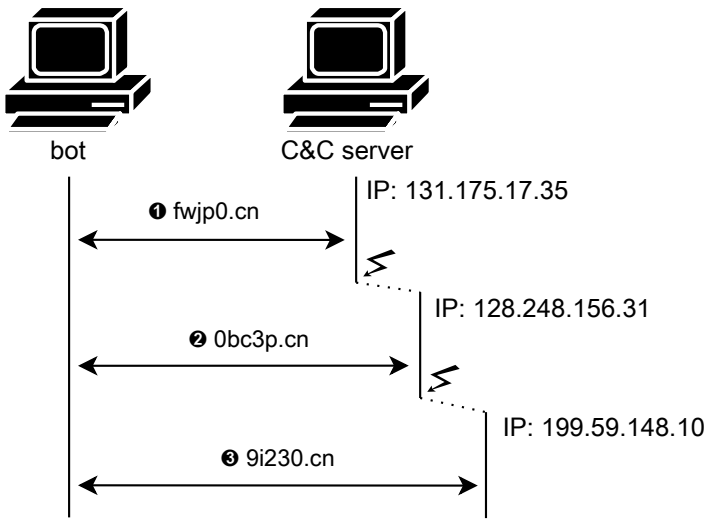

Figure 2.10: *DGA-based rallying mechanims response to sinkholing.*

plained being extremely complex and expensive, would have a limited impact on the overall communication architecture of the botnet.

2.5.2 Side Effects

Interestingly, the DGA-based rallying mechanism shows two major side effects. We go in details in describing them, as they provide the *hooks* that the security researchers employ to mitigate the phenomenon (see Section 3.3). Our research work is no exception.

Random Domain Names As we mentioned, AGDs (both registered and not registered) are the result of "randomized"[4] generations. This leads to AGD names that appear as high-entropy strings, like `gxyqo.tk`, `yodyxxu.info` or `0d1a81ab5bdfac9c8c6f6dd4278d99fb.co.cc`. Instead, legitimate domain names are meant to be human-friendly, easy to remember

[4]Here, we use the term "random" without a strict statistical meaning (e.g., random variable with a given distribution).

and pronounce (e.g., `bankofamerica.com`): We call these domains humanly-generated domains (HGDs).

The fact that AGDs appear distinguishable from HGDs is a risk for the DGA-based rallying mechanism. Nevertheless, the reasons why AGDs feature high-entropy names are two: (1) they are samples of a pseudo-randomized generation processes, (2) they should not be already registered by legitimate users. For these reasons, we claim that designing DGAs able to produce non-noisy DGAs is far from trivial and could affect the effectiveness of the overall rallying mechanism.

NXD traffic Bots perform a disproportionately large amount of DNS queries, which result in NXD replies. This happens because the vast majority of such queries hit non-existing domains. On the other hand, legitimate hosts have no reasons to generate high volumes of queries yielding NXD replies.

The fact that DGAs generate *noisy* DNS traffic may raise alarms and make malware-infected machines identifiable. Nevertheless, we claim that designing a DGA, which does not produce overwhelming amounts of DNS NXD replies is *impossibile*. In fact, generating noisy DNS traffic is a direct consequence of the malware agnosticism (see Section 2.5.1.1): In order not to generate such NXD traffic (corresponding to failed C&C connection attempts), the malware would have to *know* the correct rendezvous domain. But, if it did, it would not be agnostic and could leak such information.

2.5.3 Case Study: Torpig

Torpig is one of the first botnets that implemented a DGA-based rallying mechanism. Its DGA implementation, completely reverse engineered by Stone-Gross et al. 2009a, is an instructive case study. In fact, it features some design flaws that help us underline the importance of each requirement we stated to guarantee the security of the rallying mechanism.

The Torpig DGA is seeded with the current date and a numerical parameter. First, it computes a *weekly domain* (`weeklydomain`), which depends from

the current month and year, but remains constant over the seven days of the week. Then, it tries to resolve the following domains: `weeklydomain.com`, `weeklydomain.net` and `weeklydomain.biz` (i.e., the original domain name followed by the TLDs `com`, `net` and `biz`) looking for an active C&C server. In case all the DNS queries fail, Torpig computes a *daily domain* (`dailydomain`) executing the algorithm depicted in Figure 2.11. It then tries to resolve the domains `dailydomain.com`, `dailydomain.net` and `dailydomain.biz`.

The algorithm described is a basic example of DGA. The malware generates time-dependent domains and tries to contact them with the purpose of finding active rendezvous points with the C&C server. Nevertheless, it shows two major weaknesses.

Lack of non-determinism The DGA is not seeded with unpredictable data. This means that defenders who successfully reverse the algorithm, can generate the candidate rendezvous domains for each given day, and prepare appropriate countermeasures.

Small set of candidate rendezvous points The DGA tries to resolve only six domains every day. This make it feasible for defenders to daily register *all* these domains, thus excluding the botmaster from the command and control, with an acceptable economical effort.

These weaknesses—together with the botmaster authentication issues described in Section 2.3.4—allowed Stone-Gross et al. to take over the Torpig botnet. The researchers reversed the DGA mechanisms, registered the rendezvous domains in advance and pointed them to their own (benign) C&C servers. By doing this, they excluded the botmaster from the command and control. Interestingly, the DGA mechanism showed its strength even in the weak implementation featured by Torpig. In fact, the botmaster was able to regain control over the bots after the 10-day period, by simply registering new rendezvous domains.

```
suffix = ["anj", "ebf", "arm", "pra", "aym", "unj", "ulj",
                        "uag", "esp", "kot", "onv", "edc"]

def generate_daily_domain():
    t = GetLocalTime()
    p = 8
    return generate_domain(t, p)

def scramble_date(t, p):
    return (((t.month ^ t.day) + t.day) * p) + t.day + t.year

def generate_domain(t, p):
    if t.year < 2007:
        t.year = 2007

    s = scramble_date(t, p)
    c1 = (((t.year >> 2) & 0x3fc0) + s) % 25 + 'a'
    c2 = (t.month + s) % 10 + 'a'
    c3 = ((t.year & 0xff) + s) % 25 + 'a'

    if t.day * 2 < '0' || t.day * 2 > '9':
        c4 = (t.day * 2) % 25 + 'a'
    else:
        c4 = t.day % 10 + '1'

    return c1 + 'h' + c2 + c3 + 'x' + c4 + suffix[t.month - 1]
```

Figure 2.11: *Torpig daily domain generation algorithm (Stone-Gross et al., 2009a).*

2.5.4 DGA-based Botnets

Beside Torpig, numerous other botnets have featured DGA-like rallying mechanisms in the past. Interestingly using DGAs as primary mean to establish C&C communications is a recent approach. In fact, botnets used to employ DGAs only as fallback mechanisms, for instance, to respond to unexpected and rare interventions of security defenders.

Proceeding chronologically, Bobax (Stewart, 2004) and Kraken (Royal, 2008) were two spamming botnets[5] employing DGAs. According to Royal, a Kraken bot binary wishing to communicate with his C&C server used to automatically generate domain names of dynamic domain name system (DDNS) providers (e.g., DynDNS[6]). Specifically, it generated a randomized string of six to eleven characters and appended the second-level domain of the chosen provider, generating an AGD (i.e., a candidate rendezvous domain) such as `smpyfxs.dyndns.org`.

Srizbi was one of world's largest botnets ever discovered, considered responsible of a significant percentage of all spam emails delivered over the Internet. Its bot featured a fallback DGA mechanism (see Shevchenko, 2008) to be used in case the C&C became unreachable. Interestingly, the fallback mechanism proved itself effective around November 2008, after the shutdown[7] of the McColo ISP by hand of local authorities and security defenders. At that time, the ISP was considered responsible of hosting C&C servers of botnets such as Srizbi, Rustock, MegaD and PushDo. The Srizbi botmaster, which was disconnected from the botnet as a consequence of such operation, regained control of his infected machines thanks to the resilience of the DGA-based rallying mechanism, as reported in Krebs [2010].

[5] Actually, some sources claim that Bobax and Kraken are actually two aliases for the same botnet. See, e.g., `http://www.secureworks.com/cyber-threat-intelligence/threats/topbotnets/`.

[6] `dyndns.org`

[7] `http://www.fireeye.com/blog/technical/botnet-activities-research/2008/11/mccolo-shutdown-nov-11-2008-1323-est.html`

By querying Google for security reports about Srizbi, we were able to reconstruct the timeline of the botnet as reported by security defenders.

Oct 28, 2008 Srizbi active C&C server is found at IP 208.72.169.212. It is reported as hosted on McColo ISP[8].

Nov 11, 2008 McColo ISP, where Srizbi servers are hosted, is shut down.

Nov 12 to 24, 2008 FireEye researchers register the DGAs generated by Srizbi bots (used as fail-safe mechanism, if the IP address hard-coded in the binary is unreachable)[9]. The intent is to exclude the botmaster from the command.

Nov 25, 2008 Srizbi bots start communicating with Estonian IP addresses 92.62.100.(9|12|13). The first command from the bot master is to update the hardcoded IP of the C&C. The new server is located in Estonia (specific IP is not available, but probably 92.62.100.13, hosted by Starline Web Services, AS39823). A Russian registrar service is used to register the AGDs queried by Srizbi bots and redirect them to the Estonian IPs[10].

Nov 27, 2008 Srizbi C&C server is cutoff by Starline Web Services[11].

Feb 11, 2009 New Srizbi C&C server is found at IP 92.62.100.97 hosted by Starline Web Services[12].

Feb 17, 2009 Srizbi C&C server goes offline. No further activity related to the botnet is registered.

[8] http://blog.fireeye.com/research/2008/10/mccolo-hosting-srizbi-cc.html

[9] http://voices.washingtonpost.com/securityfix/2008/11/srizbi_botnet_re-emerges_despi.html

[10] http://blog.fireeye.com/research/2008/11/its-srizbi-trun-now.html

[11] http://www.pcworld.com/businesscenter/article/154622/estonian_isp_cuts_off_control_servers_for_srizbi_botnet.html

[12] http://blog.fireeye.com/research/2009/02/bad-actors-part-1-compic.html

The timeline of Srizbi gives a good understanding of the flexibility that the botmasters can achieve by implementing DGA-based rallying mechanisms. In particular, it is clear how DGA-based botnets provide a good resilience to the defensive interventions of security operators.

In more recent times, different versions of the Conficker malware, designed to harvest data from the infected machine and send spam emails, made extensive use of DGAs. Reports exist of the algorithm implementations featured in versions A and B (see Porras, 2009), and in version C (see Porras et al., 2009). Earliest versions of the DGA generated a daily amount of 250 domains to query, whereas version C generates 50,000 potential rendezvous domains. Of these 50,000, and unlike previous versions, only 500 are selected and queried once (see also Leder and Werner, 2009).

Similarly to Conficker, the Murofet malware, which is designed to harvest private data from compromised machines, implements a time-dependent DGA. The algorithm, reversed by Shevchenko [2010], generates 800 candidate rendezvous domains daily.

In the year 2013, Microsoft reported the existence of Bamital, a DGA-based botnet[13]. Moreover, Zeus—a toolkit to generate custom-made botnets to steal data from victim users—started featuring a DGA module[14].

[13] http://noticeofpleadings.com/

[14] http://blog.webroot.com/2013/03/14/new-zeus-source-code-based-rootkit-available-for-purchase-on-the-underground-market/

Chapter 3

State of the Art and Challenges

The problem of detection and mitigation of botnet is a widely covered topic in the literature (see Bailey et al., 2009). In this chapter we cover the most relevant research work, which specifically deals with topics related to the detection and mitigation of botnets that rely on the DGA rallying mechanism. First, we present techniques that leverage the analysis of DNS traffic to detect domain names related to all kinds of malicious activities. Then, we focus on the description of strategies that aim at detecting active C&C channels and rendezvous point. Finally, we go in detail of the literature work that specifically covers the detection of DGA-based botnets active in the wild.

In this context, we analyze the major shortcomings of the solutions proposed and we underline the necessity of overcoming such limitations. On this basis, we state the importance of Phoenix, the system that we propose to fulfil this necessity.

3.1 Detecting Malicious Domains

The DNS plays a fundamental role over the Internet. It is not surprising that malicious services depend on it as much as benign services do. Miscreants, in fact, employ the DNS to maintain large, distributed and reliable infrastruc-

tures, solving the same engineering challenges legitimate providers face. For instance, the malware that runs on bots leverages DNS resolutions to locate the C&C servers, obtaining extreme flexibility and resilience of C&C communications. Spam emails, instead, contain URLs that link to domains that resolve to scam servers. Identifying domains linked to malicious activities would help mitigate many Internet threats, ranging from botnets, phishinig sites and malware-hosting services.

In this section, we cover some relevant research work, which has been done to detect malicious domains by leveraging the analysis of DNS traffic collected at different observation points. The reason why we focus on the detection of malicious domains through the analysis of the DNS infrastructure is that, by its nature, DGA-related activities manifest themselves *only* when analyzed from this perspective. Thus, the purpose of this section is to understand how the techniques that mitigate the problem of malicious domains can be used to tackle DGA-related threats, and consequently which lessons can be learnt from it.

3.1.1 Exposure: Leveraging DNS

In their paper, Bilge et al. [2011] propose Exposure, a system that employs large-scale, passive DNS analysis techniques to detect domains that are involved in malicious activities.

The authors of Exposure show that the DNS traffic (i.e., DNS requests and responses) related to malicious domains exhibits different characteristics from the one of benign domains. On this basis, Bilge et al. propose fifteen features that allow to characterize different properties of domain names and the ways in which they are used. This leads to the implementation of a supervised classifier, able to tell benign and malicious domains apart. The classifier is trained on labeled datasets of both benign and malicious samples and is fed with DNS traffic collected at local DNS servers, without the need of visibility over the querying hosts (see Figure 3.1).

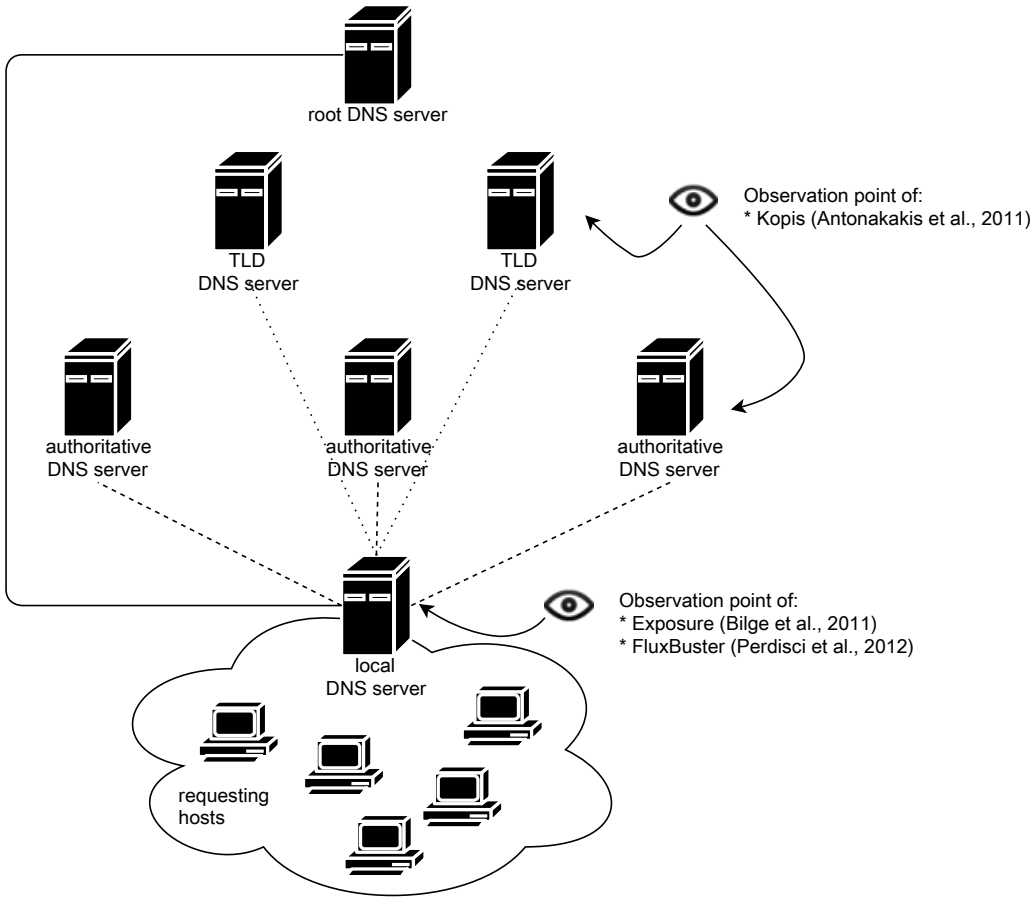

Figure 3.1: *DNS observation point of Bilge et al. [2011], Antonakakis et al. [2011] and Perdisci et al. [2012].*

Being Exposure a building block of our system, which we will introduce in Chapter 4, it is worth outlining the four classes of features Exposure uses to detect malicious domains.

Time-based Features Malicious domains tend to have peculiar time-related behaviors when compared to benign domains. For instance, malicious AGDs will often show a sudden increase followed by a sudden decrease in the number of requests. The same behavior would be uncommon for a legitimate domain.

Exposure monitors the amount of DNS requests for a given domain over time, and computes four discriminatory features. The first feature accounts for the number of days the domain has been queried. The other three features are computed with *local scope*, meaning that the analysis considers only the interval of time from the first to the last appearance of the domain. In this context, a measure is given to the daily similarity of request traffic involving the domain. Moreover, it is considered whether the domain is usually in an idle state or is queried continuously. Last, daily-repeating patterns are investigated.

Referring to these four features, benign domains usually have a longer lifespan and tend to be queried more regularly than malicious ones. Moreover, their DNS traffic is usually similar day after day and shows daily-repeating patterns (e.g., an Italian domain will be queried more often during the Italian light-hours of each day).

DNS Answer-based Features At any time, a domain name can map to multiple IP addresses. When this is the case, DNS servers cycle through the set of IPs and return different mappings to different DNS requests with the purpose of achieving load balancing. Also, a given IP address can correspond to multiple domains.

Malicious domains are often host on compromised machines. In general, these may be distributed across different autonomous systems (ASs) and countries, as miscreants do not usually target predefined regions. Also,

compromised machines are usually pointed by *many* malicious domains. For instance, this happens in the case of AGDs employed by DGA-based botnets.

In this context, Exposure employes four features to distinguish between malicious and benign domains. For each domain, the number of IP addresses associated to a given domain are taken into account. Then, the countries to which these IP addresses belong to are counted. Moreover, Exposure performs a reverse DNS query on the IP addresses to understand whether they seem associated to a residential Internet access or to a datacenter. Finally, the number of *different* domains resolving to the same IP addresses are counted.

TTL Value-based Features Each record of a DNS response is associated with a time-to-live (TTL) value, which expresses how long a given domain-IP mapping should be cached, and thus considered valid. Systems that aim at high availability often set the TTL to lower values and use round-robin DNS, in which domain-IP mappings change continuously. This way, when one of the IP addresses is not reachable at a given point in time, since the TTL values expires quickly, another IP address can be provided.

Using low TTL values and round-robin DNS is an useful technique for miscreants as well, as it allow to build malicious system with high availability, even in the presence of blacklisting and sinkholing operations. This leads to the creation of the fast-flux service networks (FFSNs), in which some compromised machines are selected to be proxies behind which other services (e.g., C&C servers, phishing websites) can be hidden. The managers of such malicious networks assign different levels of priorities to the proxy bots, which are reflected in the corresponding different TTL values.

The authors of Exposure observe that malicious domains exhibit more scattered usage of TTL values with respect to benign domains. On this basis, five features are computed for each domain, and the respective DNS responses, to capture this distinguishable behavior: average TTL, standard deviation of TTLs, number of distinct TTL observed, number of TTL changes and percentage of usage of specific TTL ranges.

Domain Name-based Features Benign services usually choose domain names that can be easily remembered by users (e.g., `bankofamerica.com`). On the other side, miscreants have no interest in employing domains fulfilling this requirement. This is particularly true when malicious domain names are generated by means of AGDs. When this is the case, it is not uncommon to have domains like `fea2f233d65f0bd3a62059c63b324cde.com`.

Exposure leverages two features to capture the *randomness* of a domain name: The amount of numerical characters in the domain name and the length of the longest meaningful substring (i.e., a word in the dictionary) contained. Both these features are normalized to the length of the domain name. Phoenix, which we will introduce in Chapter 4, employs linguistic-based features to capture the randomness of domain names. These features are documented in Section 4.2.1.1 and advance the related state-of-the-art.

3.1.2 Kopis: High-level DNS Observation Point

As depicted in Figure 3.1, Antonakakis et al. [2011] explore the usage of DNS traffic collected at the upper level of the DNS hierarchy (i.e., authoritative and TLD DNS servers) to accurately detect malware-related domains. Their system, Kopis, introduces novel features to leverage the previously-unexplored observation point and allows network operators to detect malicious domains within their authority. These features, which are used to feed a supervised classifier, can be logically divided in three groups, which we briefly overview.

Requester Diversity This group of features characterizes the geographical diversity of the machines (in this case, local DNS resolvers), which query each given domain. Here, by geographical diversity we mean the BGP prefixes, the ASs and the country codes the querying IP addresses belong to. The insight behind this group of features is that malicious domains are usually queried by machines distributed differently from those querying legitimate domains. For instance, malicious domains often expose a strong diversity of

the querying IPs. Moreover, differently from the case of (legitimate) popular domains, this diversity tends to increase while the related malware spreads.

Requester Profile Antonakakis et al. observe that the machines in large ISP networks are more likely to query malicious domains with respect to machines belonging to smaller, business-scale networks. The reason behind this is that ISP networks feature a multitude of hosts, where the administrators have little control on the malware propagation. On the other side, enterprise networks are usually better protected from security threats. In order to leverage these insights, Kopis estimates the size of the networks served by the different local DNS resolvers it observes. The goal is to associate a profile to each DNS requester (i.e., local resolver) so to weight its relevance.

Resolved-IPs Reputation Kopis obtains, via different means, the IP addresses resolving each of the candidate malicious domains[1]. It then evaluates a maliciousness score for these IP addresses (and relative BGP zones and ASs) by leveraging historical data (i.e., blacklists) of malware evidence. The insight is that malicious domains tend to resolve on IPs belonging to networks, which have hosted malicious activities in the past.

The work of Antonakakis et al. experiments the analysis of DNS traffic from a previously unexplored observation point. These poses great challenges, which are successfully tackled by Kopis. Nevertheless, the authors themselves acknowledge that Kopis cannot detect AGDs and consequently DGAs, which is the problem we are studying. These happens as AGDs usually have a short life-span, which makes them invisible to the sampling and feature extraction implemented by Kopis.

[1]Note that, given the observation point of Kopis, this information is not straightforward to gather. Specifically, it requires access to external sources.

3.1.3 FluxBuster: Detecting Malicious FFSNs

Perdisci et al. [2012] introduce FluxBuster, a system that identifies domains and IP addresses involved in the activity of FFSNs by analyzing DNS traffic collected at local DNS resolvers, without visibility over the requesting hosts (see Figure 3.1). Perdisci et al. observe that fast-flux domains are characterized by the following characteristics: (1) short TTLs, (2) high frequency of change of the set of resolving IPs, (3) high cardinality of the set of resolving IPs, (4) high dispersion of the resolved IPs across different networks. Leveraging these insights, FluxBuster uses a supervised classifier that computes features over sets of similar domains and labels them as belonging or not to a FFSN. Similar domains are those that resolve to similar sets of IP addresses. To perform the aforementioned task, the classifier is trained with labeled data of FFSN and *non*-FFSN domains.

Given a set $\mathbb{C}$ of domains and the corresponding set of IP addresses, which resolved the domains in $\mathbb{C}$, FluxBuster computes several classes of features. The features capture (1) the cardinality of $\mathbb{C}$ and of the set of IP addresses associated to $\mathbb{C}$, (2) the average TTL employed in the DNS records, (3) the diversity of the IP addresses (i.e., the number of different `/16` networks they belong to), (4) the variation of the IP sets over time. By combining these features, Perdisci et al. claim to be able to distinguish the activity of malicious FFSN domains from the one of legitimate domains. The false positive rate remains low even if among the legitimate domains some exist that belong to content distribution networks (CDNs). This is particularly interesting, as CDNs show some similarities with FFSNs, but are benign instead.

FluxBuster employes features and working procedures, which may be helpful when dealing with AGDs. Nevertheless, Perdisci et al. focus on detecting domains that are malicious from the viewpoint of the victims of the attacks perpetrated through botnets (e.g., phishing, spam, drive-by download). For instance, FluxBuster may detect phishing domains hosted on compromised machines. When this happens, the system detects the IP addresses of the

infected machines, and not those of the C&C servers under the control of miscreants.

3.2 Command-and-control Endpoints Detection

In this section we focus on the research problem of identifying C&C endpoints (i.e., C&C servers). For this purpose, we present two state-of-the-art systems, which employ two different approaches.

3.2.1 Disclosure: Leveraging NetFlow Data

As there is no ideal data source for large-scale botnet detection, Bilge et al. [2012] introduce Disclosure: A system, which leverages NetFlow data to detect C&C communications and to unveil the IP addresses of the C&C servers.

NetFlow is a network protocol proposed and implemented by Cisco Systems (see Claise, 2004) for summarizing network traffic. ISPs network appliances collect information about network flows (i.e., packet sequences with the same source and destination IP addresses and layer-4 protocol ports) and compute aggregated attributes about them. In particular, information about the parties involved in the flows (e.g., their IP addresses), the flow durations and the amount of bytes transferred are delivered to NetFlow collectors to be used for network monitoring and performances evaluation.

The use of NetFlow data for C&C server detection is justified by the need to overcome a substantial lack of other raw network data. In fact, the authors underline that both technical and administrative restrictions (e.g., privacy issues) result in difficulties in accessing other sources of traffic. Moreover, in the rare cases in which other sources of traffic are available, issues arise on how to process large amounts of raw data. On the other hand, NetFlow data is often available and concise, making it adequate for large-scale analysis.

The authors of Disclosure identify C&C communications by computing features from network flows and by leveraging a supervised classifier. The rationale behind this approach is that C&C network traffic tends to exhibit different NetFlow behaviors from that of benign traffic. The first insight is that C&C traffic usually features small network packets, of sizes with low variability. Benign traffic, instead, exhibits a wider range of packet size values. Moreover, the access patterns to legitimate servers performed by Internet users is generally chaotic, and does not show any distinguishable patterns. On the other side, bots trying to contact their C&C server generally show a recognizable and repetitive access pattern, due to the fact that they run instances of the same malware. Last, the benign traffic generated by legitimate users tends to have higher volume in the day-light hours, whereas malware samples usually continue their malicious activity and contact the C&C server without such discrimination.

The above observations lead to the definitions of the features, which Disclosure computes on the NetFlow data and uses to feed a classifier. Output of the classifier is a label, which determines whether a given network flow is part of a C&C communication. If this is the case, the identification of the C&C server involved in the communication can follow straightforward.

Even if the detection results of Disclosure appear positive, making the system worth mentioning, our feeling is that mining NetFlow data is "too simple". In particular, the detection system relies on assumptions which can be easily falsified by a miscreant wishing to evade detection. For instance, a botmaster could easily pad his C&C messages with randomized content, so to increase the size variability of his network flows. Similarly, repetitive access patterns can be obfuscated with a proper design of the malware. Adapting Disclosure to face these ever-changing threat seems hard to achieve, given that the dataset employed (i.e., NetFlow data) is poor of information.

3.2.2 Squeeze: Identifying Fallback Mechanisms

As we discussed in Chapter 2, modern malware implements sophisticated application logic, which guarantees robust C&C communications even when disruptions occur by hand of security defenders. In other words, malware features failover C&C strategies that alter the behavior of the bots in response to unexpected events such as the sinkholing of C&C servers or rendezvous domains. Such failover behaviors are unlikely to be detected through standard dynamic analyses of the malware samples, as they need special conditions to be triggered. Indeed, a manual reverse engineering of the malware may be needed to reveal the portions of binary code responsible of handling the failover functionalities.

In this context, Neugschwandtner et al. [2011] propose Squeeze, a system designed to automatically reveal C&C failover strategies. Squeeze executes malware samples in a virtualized environment and observes the network traffic generated through a lenient firewall. It then simulates network failures by dropping network packets and observes the network traffic generated by the malware as a consequence of such simulated failures. The rationale, is that malware samples will try to contact *backup* C&C servers or to trigger DGAs when faced with the loss of the first C&C contact point, thus unveiling such backup servers or DGAs. This leads to the generation of new blacklists.

Squeeze implementation features a complex multi-path exploration technique. Every time the bot tries to establish a communication, the system captures the execution state of the malware and separately triggers both the behaviors on two identical clones: When the communication is allowed and when it is obstructed. Eventually, failover strategies are triggered and unveiled without a manual analysis of the binary samples.

The approach employed by the authors is really promising. Nevertheless, it may produce misleading results when the failover behavior of a malware strongly depends on time-dependent information (e.g., Twitter trending topics). For instance, triggering the DGA of a malware may not lead to the disclosure of *realistic* AGDs, as their generation may be time-dependent.

3.3 Detecting AGDs and DGAs

The problem of detecting AGDs and identifing DGAs responsible of their generation is fairly recent. Therefore, only a handful of works have been proposed to address it. In this section, we present them in order of publication, as one extends and improves the other.

From the discussion of these works and their limitations, we state the need for a system that overcomes them: This gives the basis for our proposal of Phoenix.

3.3.1 Leveraging Linguistic Properties

Yadav et al. [2010] were the first to address the problem of detecting AGDs with a paper later republished in 2012. Their work is based on the observation that AGDs names exhibit patterns that differ substantially from those of HGDs. In fact, DGAs are designed to generate domain names that are not well-formed nor pronounceable. This way, the probability of an AGD of being already in use, thus not registrable by the botmaster, is lowered.

Yadav et al. propose a methodology that analyzes DNS traffic at the quest for domain names that appear automatically (i.e., randomly) generated. In particular, they present a technique for anomaly detection, which may be applied to *groups* of domains, able to distinguish between groups of automatically generated and groups of humanly generated domains. This is done by means of a supervised classifier, which computes linguistic features over previously-unseen domain groups (i.e., distribution of alphanumeric characters as well as bigrams) and computes distances of these groups from labeled groups of domains. The approach can be seen as a 1-nearest neighbor algorithm, in which each domain group is a data sample.

The authors explore different strategies to group together domains before feeding them to the classifier and performing the analysis.

Per-domain analysis Domains are grouped according to their sec-

ond level domain labels. So, for instance, `abc.example.com` and `def.example.com` would be considered together.

Per-IP analysis For each IP address, a group is created containing all the domains that resolved to that IP address.

Component analysis A bipartite graph is built, featuring a node for each domain, and a node for each IP address. An edge exists between two nodes when a given domain resolved to a given IP. Groups of domains are extracted by identifying connected components in the bipartite graph built as described.

The approach of Yadav et al. has two major shortcomings. First, it is completely supervised, meaning that datasets of AGDs should be provided as input during the learning phase. This is far from trivial, as AGD datasets are a scarce resource, which is available—if ever—months after the AGDs have been employed for their purpose. Moreover, the supervised approach prevents from detecting AGDs produced by previously unseen DGAs.

Second, the system is not able to classify domains, it classifies *groups of domains* instead. The reason behind this is because every domain name, no matter how benign, has a distribution of alphanumeric characters and bigrams, which is peculiar, due to the fact that domain names are usually really short. Grouping domains, however, is error prone and the strategies proposed seem to fail in real-world scenarios.

3.3.2 Leveraging DNS Failures

Yadav and Reddy [2012] extend their previous work and speed-up their detection process by leveraging DNS failures. Specifically, they exploit the knowledge that DGA-based malware generates significant amounts of NXD DNS replies to identify *registered* AGDs and the corresponding C&C servers. For this purpose, they leverage DNS network traffic collected at DNS resolvers, with visibility over the IP addresses of the hosts performing the DNS queries, as depicted in Figure 3.2.

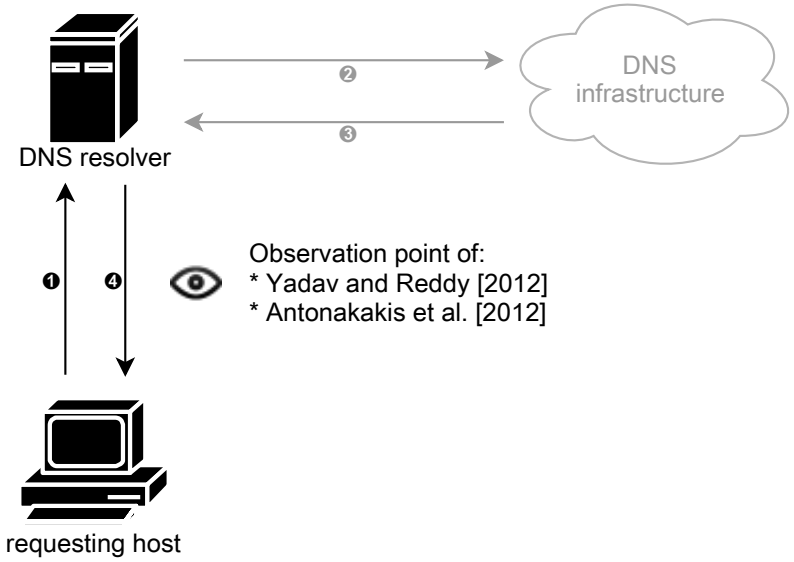

Figure 3.2: *DNS observation point of Yadav and Reddy [2012] and Antonakakis et al. [2012].*

The rationale behind their strategy is that a malware-infected machine will generate a overwhelming amount of NXD traffic trying to resolve AGDs not registered by the botmaster. Finally, they will hit a registered AGD, being able to discover the current IP address of the C&C server. On this basis, the system Yadav and Reddy implemented, keeps track of the DNS queries generated by each querying host (thus the necessity of having visibility over its IP address) trying to identify the aforementioned pattern. Beside measuring the randomness of domain names, they then identify registered AGDs as those *registered* domains queried after a sequence of failing DNS requests.

The approach described is really interesting, as it explores for the first time the use of NXD traffic for fast botnet detection. Nevertheless, it has some drawbacks, partially overcame in following research works. First, it requires access to DNS traffic data with visibility over the IP addresses of the querying host. This kind of data is difficult to gather, as we will explain later. Moreover, the proposal of Yadav and Reddy requires the collection

of precise timelines of the DNS queries performed by each user. Although theoretically possible, working with such precision may be not trivial when dealing with real-world high-volume networks, for which sampling traffic data may be the only viable option.

Antonakakis et al. [2012] propose Pleiades, a system that partially overcomes the limitations of the work proposed by Yadav and Reddy. Pleiades analyzes a feed of DNS traffic collected *below* the DNS resolvers (see Figure 3.2) with the goal of (1) detect DGAs active in the wild, (2) identify AGDs and label them as being coined by specific DGAs. Similarly to what proposed by Yadav and Reddy, Pleiades leverages the assumption that DGA-based botnets generate large amounts of NXD traffic. Nevertheless, its use of such information seems more reasonable and applicable under less constraining hypotheses. Because of this, Pleiades can be considered the state-of-the-art in DGA-related threats detection.

Pleiades collects the domains that result in NXD responses. It then clusters these domains according to two orthogonal similarity criteria: (1) the domain name strings have similar statistical characteristics[2] (e.g., similar length, similar level of "randomness," similar character frequency distribution), (2) the domains have been queried by overlapping sets of hosts. These two resulting sets of clusters are then combined with a cartesian product-like operation and filtered to remove noise. The outcome of this unsupervised operation are clusters containing *unregistered* domain names that are likely automatically generated by the same algorithm running on multiple machines within the monitored network.

The clusters of domains are used to fingerprint the automatic generation process, which lies behind them, so to be able to build a supervised classifier that assigns a previously-unseen *registered* domain to one of these clusters,

[2]Interestingly, these statistical characterstics are not computed over *domains* but over *groups of domains*. The reason behind this is the same we discussed when introducing Yadav et al. [2010]. Antonakakis et al. extract statistical characteristics from randomly-sampled groups of domains.

or none. When the first event occurs, an active rendezvous point has been detected.

Both Yadav and Reddy and Antonakakis et al. employ NXD responses. However, the NXD responses alone carry little information that can be directly used to identify the families of AGDs. Specifically, an NXD response only holds the queried domain name plus some timing data. Therefore, as also noticed by Perdisci et al. [2012], the NXD criterion requires knowledge of the querying hosts (i.e., the bots). Indeed, Antonakakis et al. [2012] uses the client IP to group together NXDs queried by the same set of hosts. Yadav and Reddy [2012] instead groups together DNS queries originated by the same client to define the correlation between distinct requests.

Unfortunately, relying on the IP addresses of the querying hosts has some drawbacks. First, it is error prone, because little or no assumptions can be made on the IP (re-)assignment policies employed on each network, as well as on the use of NATs and proxies. Secondly, and more importantly, obtaining access to this information is difficult. In fact, it requires access to local DNS resolvers. This can be problematic for researchers but also for practitioners who want to operate systems similar to those described by Yadav and Reddy and Antonakakis et al.. Last, the need for information about the querying hosts raises privacy issues and, consequently, leads to non-repeatable experiments (Rossow et al., 2012), as datasets that include these details are not publicly available. Moreover, the requirement constraints the deployment of detection tool to the lowest level of the DNS hierarchy, preventing a large-scale, high-level use of the proposed solutions.

3.4 Goals and Challenges

After considering the above motivations, we conclude that a method to find families of AGDs yielded by the same DGAs, without specifically requiring access to low- or top-level DNS data is necessary. In the remainder of this

work, we describe Phoenix, the system that we propose to accomplish this goal.

AGDs appear completely random at sight, but creating automated procedures capable of modeling and characterizing such "randomness" is not trivial. This is particularly difficult when observing one domain at a time, as one sample is not representative of the whole random generation process. On the other hand, the task of choosing the optimal group of AGD samples to extract the characteristics of the DGA is also challenging. How to group AGDs samples together? How to avoid spurious samples that would bias the result?

To collect data about DGAs, the only observation point is the DNS infrastructure. Collecting and dealing with DNS traffic presents some challenges on (1) where to place the observation point and (2) how to process the collected data. Indeed, DNS traffic is characterized by large amounts of records, each carrying little information. In addition, the amount of information that can be collected varies depending on where the sensors are deployed in the DNS hierarchy (e.g., low-level sensors have access to the querying hosts, but are difficult to deploy, whereas higher-level sensors are easier to deploy but their visibility is limited to the stream of queried domains and their IPs). Moreover, the domain-to-IP relations are highly volatile by their nature, with both dense and sparse connections; this impacts the space and computation complexity of the algorithms that are needed to analyze the DNS traffic.

Last, little or no ground truth is usually available regarding DGAs. When available, such ground truth is limited to the knowledge of a specific implementation of DGA as a result of a specific reverse engineering effort (see Stone-Gross et al., 2009a). Therefore, this knowledge is outdated once released and, more importantly, not representative of the whole domain generation process but only of a limited view.

Chapter 4

System Description

In this chapter we introduce Phoenix, the system we propose to fulfil the goals we stated in Section 3.4.

Following a top-down approach, we describe the high-level modular architecture of the system and we state the roles covered by the different modules. Then, we describe with detail the internal functioning of each module. While doing this, we explain and motivate the rationale behind every strategy adopted.

We conclude the chapter by briefly describing the implementation of our prototype of Phoenix.

4.1 Overview of Phoenix

Phoenix is divided into three modules, as summarized in Figure 4.1. The core of Phoenix is the **DGA Discovery** module, which identifies and models AGDs by mining a stream of domains. The **AGD Detection** module receives one or more domain names, plus the respective DNS traffic, and uses the models built by the **DGA Discovery** module to tell whether such domain names appear to be automatically generated. If that is the case, this module also labels those domains with an indication of the DGA that is most likely behind the domain generation process. The **Intelligence and**

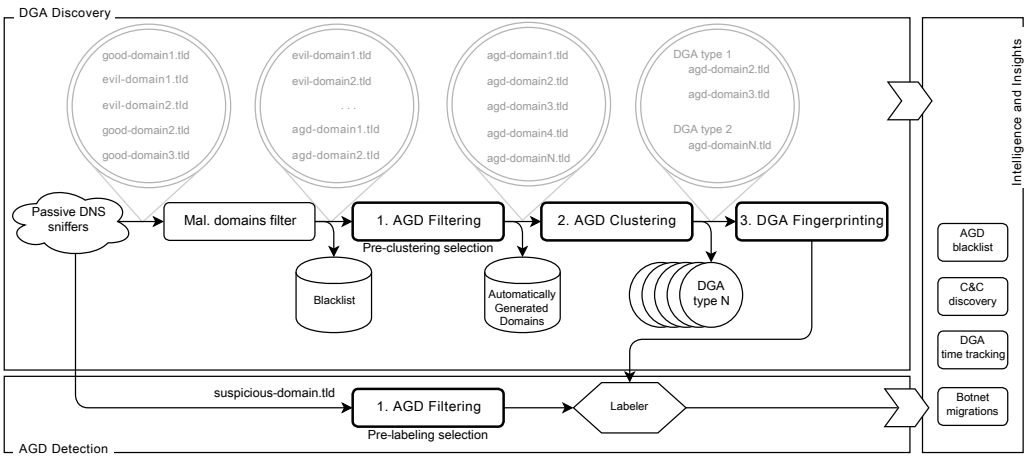

Figure 4.1: *Overview of the modules of* Phoenix.

Insights module aggregates, correlates and monitors the results of the other modules to extract meaningful insights and intelligence information from the observed data (e.g., whether a DGA-based botnet is migrating across ASs).

4.1.1 DGA Discovery Module

This module receives as input (1) a stream of domain names that are known to be malicious and (2) a stream of DNS traffic (i.e., queries and replies) related to such domains. This information is publicly accessible and can be obtained easily from (1) a blacklist or domain reputation system and (2) a passive DNS monitor. No assumption is made about the type of malicious activity the domains are used for (e.g., phishing websites, spamming campaigns or drive-by download websites, botnet communication protocols).

This module uses the following 3-step pipeline to recognize domains that are used in DGA-based botnets as rendezvous points between bots and C&C servers.

Step 1 (AGD Filtering): This step extracts a set of linguistic features from the domain names. Its goal is to recognize the (domain) names that appear to be the results of automatic generations. For instance, this step will distinguish between `5ybdiv.cn`, which appears to be an

AGD, and `yahoo.cn`, which appears to be a HGD. We make no assumptions about the type of DGA that have generated the domains, although we do assume that at least one exists. The output of this step is thus a set of AGDs, possibly generated by different DGAs.

Step 2 (AGD Clustering): This step extracts some features from the DNS traffic of the domains that have passed **Step 1**. These features are used to cluster together the AGDs that have resolved to similar sets of IP addresses—possibly, the C&C servers. These *AGD clusters* will partition the AGDs according to the DNS replies observed. For example, if the AGDs `5ybdiv.cn` and `hy093.cn` resolved to the same pool of IPs, we will cluster them together. Here, we assume that AGDs generated by different DGAs are used by distinct botnets, or distinct malware variants, or at least by different botmasters, who have customized or tuned a DGA for their C&C strategy. Therefore, this partitioning will, to some extent, reflect the different botnets that AGDs have been employed for.

Step 3 (DGA Fingerprinting): This step extracts the features from the AGD clusters and creates the models that define the fingerprints of the respective DGAs. The **AGD Detection** module uses these fingerprints as a lookup index to identify the type of previously-unseen domains. For instance, as clarified in the remainder of this work, `epu.org` and `xmsyt.cn` will match two distinct fingerprints.

The output of this module is thus a set of clusters with their fingerprints.

4.1.2 AGD Detection Module

This module receives in input a previously unseen domain name d, which can be either malicious or benign. The goal of this module is to verify whether d is automatically generated. For this, it relies once again on the **AGD Filtering** step.

The domain names that will pass this filter will undergo further checks, performed by the **Labeler**, which may eventually flag them as not belonging to any AGD cluster (i.e., not matching any of the fingerprints). Therefore, in this step, including domains that do not belong to some DGA is not strictly important. It is instead more important not to discard suspicious domains. Therefore, for this module only, we configure the **AGD Filtering** step with looser parameters (as described in Section 5.2.1), such that we do not discard domains that exhibit, even slightly, the characteristics that are typical of AGDs.

After the filtering, this module leverages the AGD clusters and their respective fingerprints to find the DGA, if any, that may lie behind d.

4.1.3 Intelligence and Insights Module

Once an AGD is labeled as such, it can be recognized among others as being the result of a specific DGA. Therefore, the outcomes of the previous modules allow the extraction of summarized information and novel knowledge about the botnet employing the different DGAs.

With this knowledge, the addresses of the C&C servers and lists of malicious AGDs can be grouped together in small sets that are easier to analyze than if considered into one, large set, or distinctly. For instance, if an analyst knows that 100 domains are malicious, he or she can use the label information to split them into two smaller sets: One that contains domain names "similar" to `5ybdiv.cn` and `hy093.cn`, and one with domains "similar" to `epu.org`. The top level domain is not distinctive, we use it here as a mere example. With this information, the analyst can track separately the evolution of the IPs that the two groups point to and take actions. For example, recognizing when a C&C is migrated to a new AS or undergoing a takedown operation is easier when the set of IPs and domains is small and the characteristics of the DGAs are known and uniform.

Generally speaking, these analyses, for which we provided two use cases in Section 5.4, can lead to high-level intelligence observations and conjectures,

useful for the mitigation of DGA-related threats. In this, Phoenix advances the state-of-the-art by providing researchers and practitioners with a tool that goes beyond blacklists and reputation systems.

4.2 System Details

In this section we go in details in defining the internal functioning of each module composing Phoenix and we formalize the rationale behind every strategy adopted.

From now on, when dealing with domain names (e.g., `www.example.co.uk`), we will always refer with the term *chosen prefix* to the rightmost domain label (e.g., `example`) preceding the eTLD (e.g., `co.uk`). Instead, other domain labels (e.g., `www`) will not be considered.

4.2.1 Step 1: AGD Filtering

In this step we make the assumption that if a domain is automatically generated it has different linguistic features with respect to a domain that is generated by a human. This assumption is reasonable because HGDs have the primary purpose of being easily remembered and used by human beings, thus are usually built in a way that meets this goal. On the other hand, AGDs exhibit a certain degree of linguistic randomness, as numerous samples of the same randomized algorithm exist.

4.2.1.1 Linguistic Features

For ease of explanation and implementation, Phoenix considers linguistic features based on the English language, as discussed in Section 6.1. Given a domain d and its chosen prefix $p = p_d$, we extract two classes of linguistic features to build a 4-element feature vector.

LF1: Meaningful Characters Ratio This feature models the ratio of characters of the chosen prefix p that compose meaningful words. Domains with a low ratio are likely automatically generated.

Specifically, we split p into n meaningful sub-words w_i of at least 3 symbols: $|w_i| \geq 3$, possibly leaving out some symbols, and we compute the feature as

$$R(d) = R(p) := \frac{\sum_{i=1}^{n} |w_i|}{|p|}.$$

In the case of $p = \texttt{facebook}$, the prefix is fully composed of meaningful words, producing the high ratio value

$$R(p) = \frac{|\texttt{face}| + |\texttt{book}|}{|\texttt{facebook}|} = \frac{4+4}{8} = 1.$$

In a case such as $p = \texttt{pub03str}$, instead, the value remains lower:

$$R(p) = \frac{|\texttt{pub}|}{|\texttt{pub03str}|} = \frac{3}{8} = 0.375.$$

LF2: *n*-gram Normality Score This class of features captures the pronounceability of the chosen prefix of a domain. This problem is well studied in the language form field, which is a field of linguistics, the scientific study of human language. The problem of calculating the pronounceability of a word is reduced to quantifying the extent to which a string adheres to the phonotactics of the (English) language. The more permissible the sequence of phonemes (see Scholes 1966, Bailey and Hahn 2001), the more pronounceable a word is. Domains with a scarce number of such sequences are likely automatically generated.

We calculate this class of features by extracting the n-grams of p, which are the substrings of p of length $n \in \{1, 2, 3\}$, and counting their occurrences in the (English) language dictionary. The features are thus parametric to n:

$$S_n(d) = S_n(p) := \frac{\sum_{n\text{-gram } t \text{ in } p} \text{count}(t)}{|p| - n + 1}$$

where count (t) counts the number of occurrences of the n-gram t in the language dictionary.

fa	ac	ce	eb	bo	oo	ok	$S_2 = 170.8$
109	343	438	29	118	114	45	

aa	aw	wr	rq	qv	$S_2 = 13.2$
4	45	17	0	0	

Figure 4.2: *2-gram normality score S_2 for* `facebook.com` *and* `aawrqv.biz`.

Figure 4.2 shows the value of S_2, along with its derivation, for one HGD and one AGD, respectively `facebook.com` and `aawrqv.biz`.

4.2.1.2 Statistical Linguistic Filter

Phoenix uses **LF1–2** to build feature vectors

$$\boldsymbol{f}(d) = [R(d), S_1(d), S_2(d), S_3(d)]^T.$$

It extracts these features from a dataset $\mathbb{D}_{HGD}$ of HGDs (the 100,000 most popular domains according to Alexa[1]) and calculates their mean $\boldsymbol{\mu} = [\overline{R}, \overline{S_1}, \overline{S_2}, \overline{S_3}]^T$ and covariance matrix $\boldsymbol{C}$, which respectively represent the statistical average values of the features, and their correlation. Strictly speaking, the mean defines the centroid of the HGD dataset in the features' space, whereas the covariance identifies the shape of the hyperellipsoid around the centroid containing all the samples. Our filter constructs a confidence interval, with the shape of such hyperellipsoid that allows us to separate HGDs from AGDs with a measurable, statistical error that we set a priori.

The rationale of building a filter as described is that obtaining a dataset of HGDs is straightforward and does not constrain the filtering to specific AGDs. Instead, our filter models non-AGDs by means of the generic modeling of HGDs.

[1] `http://www.alexa.com/`

Distance Measurement To tell whether a previously-unseen domain d' resembles the typical features of HGDs, the filter measures the distance between the feature vector $\boldsymbol{f}(d') = \boldsymbol{x}$ and the centroid. To this end, we leverage the Mahalanobis distance:

$$\mathrm{d}_{Mah}(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{C}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})}.$$

This distance has the property of (1) taking into account the correlation between features—which is significant, because of how the features are defined, and (2) operating with scale-invariant datasets.

Distance Threshold A previously-unseen domain d' is considered as automatically generated when its feature vector identifies a point that is too distant from the centroid: $\mathrm{d}_{Mah}(\boldsymbol{x}) > t$. To take a proper decision, we define the threshold t as the p-percentile of the distribution of the variable $\mathrm{d}_{Mah}(\cdot)$ computed over the set $\mathbb{D}_{HGD}$, where $(1 - p)$ is the fraction of HGDs that we allow to confuse as AGDs. In this way, we can set a priori the amount of errors.

As mentioned in Section 4.1.2, the **DGA Discovery** module employs a strict threshold, $t = \Lambda$, whereas the **AGD Detection** module requires a looser threshold, $t = \lambda$, where $\lambda < \Lambda$.

Threshold Estimation To estimate proper values for λ and Λ, we compute $\mathrm{d}_{Mah}(\boldsymbol{x})$ for $\boldsymbol{x} = \boldsymbol{f}(d), \forall d \in \mathbb{D}_{HGD}$, whose distribution is plotted in Figure 4.3 as ECDF. We then set Λ to the 90-percentile and λ to the 70-percentile of that distribution, as annotated in the figure.

Figure 4.4 depicts the 99%-variance preserving 2D projection of the hyperellipsoid associated to $\mathbb{D}_{HGD}$, together with the confidence interval thresholds calculated as mentioned above.

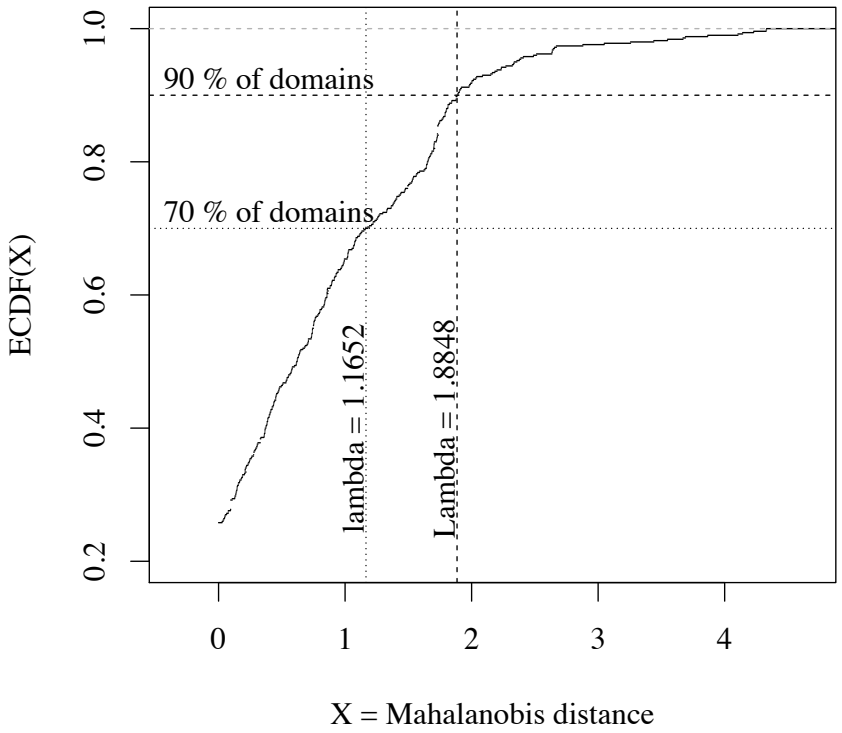

Figure 4.3: *Mahalanobis distance ECDF for Alexa top 100,000 domains with λ and Λ identification.*

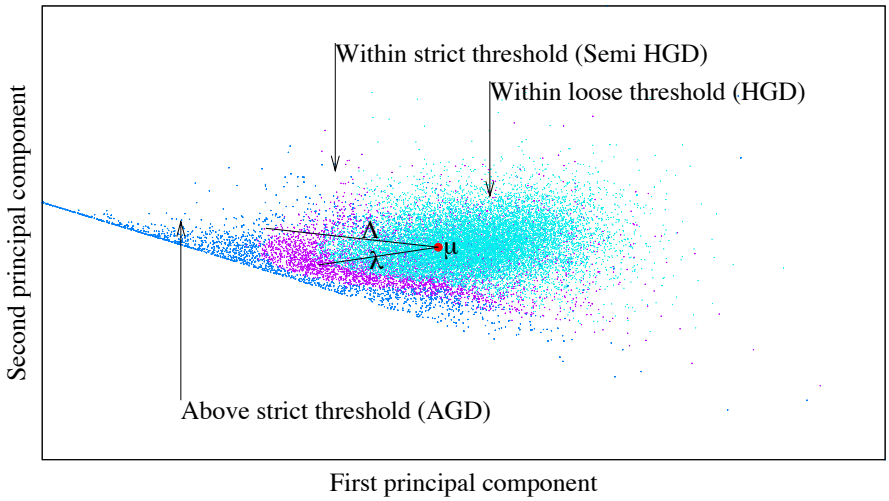

Figure 4.4: *Principal components of the Alexa top 100,000 domains hyperellipsoid with annotation of the confidence interval thresholds.*

4.2.2 Step 2: AGD Clustering

This step receives as input the set of domains $d \in \mathbb{D}$ that have passed **Step 1**. These domains are such that $\mathrm{d}_{Mah}(\boldsymbol{f}(d)) > \Lambda$, which means that (beside being malicious) are likely to be automatically generated, because they are too far from the centroid of the HGDs.

Our goal is to cluster domains according to their similarity, where we define as *similar* two domains that resolved to similar sets of IP addresses. The rationale is that the botmaster of a DGA-based botnet registers several domains that, at different points in time, resolve to the same set of IPs (i.e., the C&C servers). These are the domains that we wish to group together.

To find similar domains, we represent the domain-to-IP relation as a bipartite graph, which we convert in a proper data structure that allows us to apply a spectral clustering algorithm (see Newman, 2010) that returns the groups of similar domains. In this graph, two sets of node exists: $K = |\mathbb{D}|$ nodes represent the domains, and $L = |\,\mathrm{IPs}(\mathbb{D})|$ nodes represent the IPs. An

edge exists from a node $d \in \mathbb{D}$ to node $l \in \text{IPs}(\mathbb{D})$ whenever domain d pointed to IP l.

4.2.2.1 Bipartite Graph Recursive Clustering

To cluster the domain nodes $\mathbb{D}$, we hierarchically (thus recursively) apply the DBSCAN clustering algorithm (see Han and Kamber, 2006), which is particularly fast and easy to implement in our scenario.

Data Structure We encode the bipartite graph as a sparse matrix $\boldsymbol{M} \in \mathbb{R}^{L \times K}$ with L rows and K columns. Each cell $M_{l,k}$ holds the weight of an edge $k \to l$ in the bipartite graph, which represents the fact that domain d_k resolves to IP l. The weight encodes the "importance" of this relation. For each IP l in the graph, the weights $M_{l,k}, \forall k = 1, \ldots, K$ are set to $\frac{1}{|\mathbb{D}(l)|}$, where $\mathbb{D}(l) \subset \mathbb{D}$ is the subset of domains that point to that IP. This weight encodes the peculiarity of each IP: The less domains an IP is pointed by, the more characterizing it is.

Domain Similarity At this point we calculate the matrix $\boldsymbol{S} \in \mathbb{R}^{K \times K}$, whose cells encode the similarity between each pair of domains d and d'. We want to consider two domains as highly similar when they have peculiar IPs in common. Therefore, we calculate the similarity matrix from the weights, as $\boldsymbol{S} = \boldsymbol{N}^T \cdot \boldsymbol{N} \in \mathbb{R}^{K \times K}$, where $\boldsymbol{N}$ is basically $\boldsymbol{M}$ normalized by columns (i.e., $\sum_{l=1}^{L} M_{l,k} = 1, \forall k = 1, K$). This similarity matrix implements the rationale that we mentioned at the beginning of this section.

Domain Features and Clustering We compute the first normalized eigenvector $\boldsymbol{v}$ from $\boldsymbol{S}$. At this point, each domain name d_k can be represented by its feature v_k, the k-th element of $\boldsymbol{v}$, which is fed to the DBSCAN algorithm to produce the set of T clusters $\mathcal{D} = \{\mathbb{D}^1, \ldots, \mathbb{D}^T\}$ at the current recursive step.

Clustering Stop Criterion We recursively repeat the clustering process on the newly-created clusters until one of the following conditions is verified:

- a cluster of domains $\mathbb{D}' \in \mathcal{D}$ is too small (e.g., it contains less than 25 domains) thus it is excluded from the final result;
- a cluster of domains has its $\boldsymbol{M}$ matrix with all the elements greater than zero, meaning that the bipartite graph it represents is strongly connected;
- a cluster of domains cannot be split further by the DBSCAN algorithm with the value of ϵ set. In our experiments, we set ϵ to a conservative low value of 0.1, so to avoid the generation of clusters that contain domains that are not similar. Manually setting this value is possible because we apply the DBSCAN algorithm on normalized features.

The final output of DBSCAN is $\mathcal{D}^\star = \{\mathbb{D}^1, \ldots, \mathbb{D}^R\}$. The domains within each $\mathbb{D}^r$ are similar among each other.

4.2.2.2 Dimensionality Reduction

Each recursive execution of the clustering algorithm employed has a space complexity of $O(|\mathbb{D}|^2)$. To keep the problem feasible we randomly sample our dataset $\mathbb{D}$ of AGDs into I smaller datasets $\mathbb{D}_i, i = 1, \ldots, I$ of approximately the same size, and cluster each of them independently, where I is the minimum value such that a space complexity in the order of $|\mathbb{D}_i|^2$ is affordable. Once each $\mathbb{D}_i$ is clustered, we recombine the I clustered sets, $\mathcal{D}_i^\star = \{\mathbb{D}^1, \ldots, \mathbb{D}^{R_i}\}$, onto the original dataset $\mathbb{D}$. Note that each $\mathbb{D}_i$ may yield a different number R_i of clusters. This procedure is very similar to the MapReduce programming model, where a large computation is parallelized into many computations on smaller partitions of the original dataset, and the final output is constructed when the intermediate results become available.

We perform the recombination in the following post-processing phase, which is run anyway, even if we do not need any dimensionality reduction—that is, when $I = 1$ and thus $\mathbb{D}_1 \equiv \mathbb{D}$.

4.2.2.3 Clustering Post-processing

We post-process the set of clusters of domains $\mathcal{D}_i^\star$, $\forall i$ with the following **Pruning** and **Merging** procedures. For simplicity, we set the shorthand notation $\mathbb{A} \in \mathcal{D}_i^\star$ and $\mathbb{B} \in \mathcal{D}_j^\star$ to indicate any two distinct sets of domains (i.e., clusters) that result from the previous DBSCAN clustering, possibly with $i = j$.

Pruning Clusters of domains that exhibit a nearly-one-to-one relation with the respective IPs are considered unimportant because, by definition, they do not reflect the concept of DGA-based C&Cs (i.e., many domains, few IPs). Thus, we filter out the clusters that are flat and show a pattern-free connectivity in their bipartite domain-IP representation. This allows to remove "noise" from the dataset.

Formally, a cluster $\mathbb{A}$ is removed if

$$\frac{|\text{IPs}(\mathbb{A})|}{|\mathbb{A}|} > \gamma,$$

where γ is a threshold that is derived automatically as discussed in Section 5.2.2.

Merging Given two independent clusters $\mathbb{A}$ and $\mathbb{B}$, they are merged together if the intersection between their respective sets of IPs is not empty. Formally, $\mathbb{A}$ and $\mathbb{B}$ are merged if $\text{IPs}(\mathbb{A}) \cap \text{IPs}(\mathbb{B}) \neq \emptyset$. This merging is repeated out iteratively, until every combination of two clusters violates the above condition.

The outcome of the post-processing phase is thus a set of clusters of domains $\mathcal{E} = \{\mathbb{E}^1, \ldots, \mathbb{E}^Q\}$ where each $\mathbb{E}^q$ (1) exhibits a domain-to-IP pattern and (2) is disjunct to any other $\mathbb{E}^p$ with respect to its IPs. In conclusion, each cluster $\mathbb{E}$ contains the AGDs employed by the same botnet backed by the C&C servers at IP addresses $\text{IPs}(\mathbb{E})$.

4.2.3 Step 3: DGA Fingerprinting

The AGD clusters identified with the previous processing are used to extract fingerprints of the DGAs that generated them. In other words, the goal of this step is to extract the invariants of the DGAs. We use these fingerprints in the **AGD Detection** module to assign labels to previously-unseen domains, if they belong to one of the clusters.

Given a generic AGD cluster $\mathbb{E}$, corresponding to a given DGA, we extract the following cluster features:

CF1: C&C Servers Addresses defined as IPs($\mathbb{E}$).

CF2: Chosen Prefix Length Range captures the lengths of the chosen prefixes allowed for the domains in $\mathbb{E}$. The boundaries are defined as the lengths of the shortest and longest chosen prefixes of the domains of $\mathbb{E}$.

CF3: Chosen Prefix Character Set C employed for the chosen prefixes of the domains, defined as

$$C := \bigcup_{e \in \mathbb{E}} \text{charset}(p_e),$$

where p_e is the chosen prefix of e. It captures which characters are used during the random generation of the domain names.

CF4: Chosen Prefix Numerical Characters Ratio Range $[r_m, r_M]$ captures the ratio of numerical characters allowed in the chosen prefix of a given domain. The boundaries are, respectively, the minimum and the maximum of $\frac{\text{num}(p_e)}{|p_e|}$ within $\mathbb{E}$, where $\text{num}(p_e)$ is the number of numerical characters in the chosen prefix of e.

CF5: Public Suffix Set The set of eTLDs employed by the domains in $\mathbb{E}$.

To some extent, these features define the aposteriori linguistic characteristics of the domains found within each cluster $\mathbb{E}$. In other words, they define a model of $\mathbb{E}$.

4.2.4 AGD Detection Module

This module receives a previously-unseen domain d and decides whether it is automatically generated by running the **AGD Filtering** step with a loose threshold λ. If d is automatically generated, it is matched by the **Labeler** against the fingerprints of the known DGAs on the quest for correspondences.

In particular, we first select the candidate AGD clusters $\{\mathbb{E}\}$ that have at least one IP address in common with the IP addresses that d pointed to: $\text{IPs}(d) \cap \text{IPs}(\mathbb{E}) \neq \emptyset, \forall \mathbb{E}$. Then, we select a subset of candidate clusters sharing the same features **CF1–5** of d. Specifically, the length of the chosen prefix of d, its character set, its numerical characters ratio, and the eTLD of d must lie within the ranges defined above.

The clusters that survive this selection are chosen as labels for d.

4.3 System Implementation

We implemented a prototype of Phoenix in Python with the purpose of evaluating its capabilities. To this end, we strongly relied on the NumPy library, which offers a variety of well-tested and well-documented statistical primitives, while instead we preferred to write *ad-hoc* implementations of the data mining algorithms, to exploit domain-specific knowledge (e.g., data dimensionality).

In the following of this section, we provide sketches of the software design diagrams to clarify the mapping between the modules described in Section 4.2 and the Python classes we implemented.

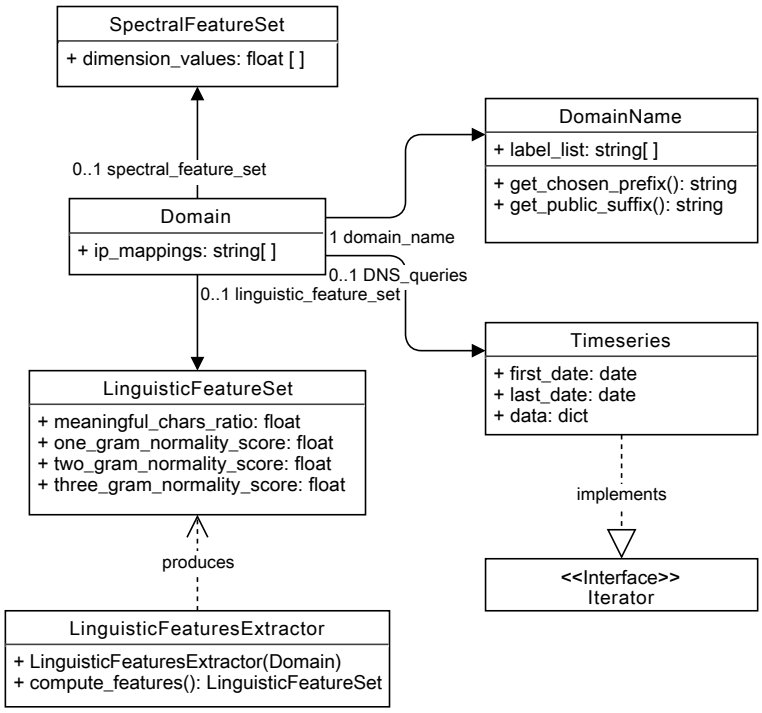

Figure 4.5: *UML diagram of* Domain *and related classes.*

4.3.1 Data Structures

The main data structures we implemented are meant to store information related to single domains and to ensembles (i.e., clusters) of domains. In the next two subsections, we go in detail in exploring the two cases.

4.3.1.1 Domain

An object of the class Domain stores the information related to a given domain d, as depicted in Figure 4.5. It contains the list of IP addresses to which domain d has resolved during its lifetime, accessible through the class attribute ip_mappings.

The attribute domain_name points to an object of class DomainName, which contains the domain name of d, parsed in labels. Moreover, its class allows to access the chosen prefix and the eTLD of d through the methods get_chosen_prefix() and get_public_suffix().

Then, an object of class Domain can be associated (through the attribute linguistic_feature_set) with an instance of LinguisticFeatureSet, which stores the linguistic feature vector $\boldsymbol{f}(d)$ of domain d, as defined in Section 4.2.1.2. These features, as we will clarify later, are computed by an object of the class LinguisticFeaturesExtractor, which is fed with a Domain and returns the corresponding LinguisticFeatureSet. Similarly, a Domain can be linked (through the attribute spectral_feature_set) to a SpectralFeatureSet object, which holds the spectral features of d computed and employed by the DBSCAN clustering algorithm during the **AGD Clustering**, fully described in Section 4.2.2.

Finally, an object of class Domain can be associated with a time series—specifically, with an instance of the class Timeseries—which stores the DNS traffic related to domain d.

The presence of the SpectralFeatureSet, LinguisticFeatureSet and Timeseries objects attached to an object Domain corresponding to domain d, depends from the context in which d is employed (i.e., **DGA Discovery**. **AGD Detection** or **Intelligence and Insights** modules).

4.3.1.2 Domain Cluster

An object of the class DomainCluster is a collection (i.e., cluster) of objects of type Domain, which are stored in the attribute domain_list as shown in Figure 4.6. The method get_ip_set() returns all the IP addresses resolved by the domains in the cluster, while the method get_ip_backbone_set() returns the IP addresses resolved by *all* the domains in the cluster.

A DomainCluster object can be associated to an object of type LinguisticDescriptor, storing the fingerprints of the domain cluster (thus of the generation mechanism behind it) as defined in Section 4.2.3. The object LinguisticDescriptor is produced by an instance of the class LinguisticDescriptorExtractor, which produces the fingerprints when fed with an instance of DomainCluster. A BipartiteGraphDescriptor object can also be associated to a newly-created cluster at the end of every DBSCAN recursive execution (see

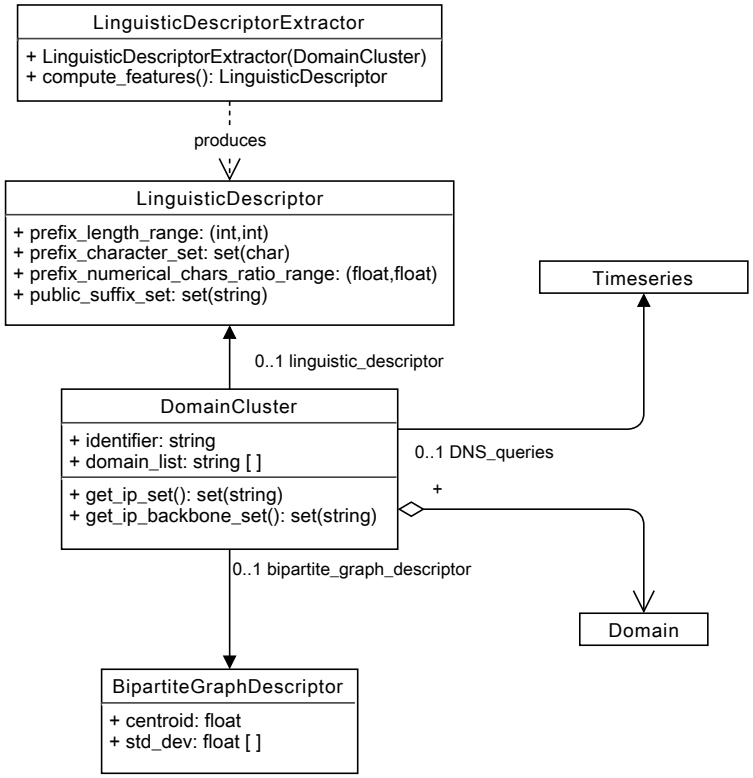

Figure 4.6: *UML diagram of the class* DomainCluster *and related classes.*

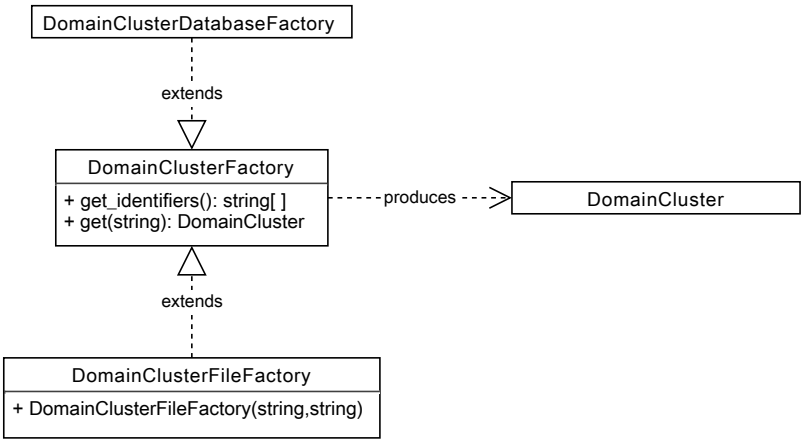

Figure 4.7: *UML diagram of class* DomainClusterFactory *and its extensions.*

Section 4.2.2). It stores the mean and the standard deviation of the spectral feature values of the domains clustered together.

Finally, a DomainCluster object can be linked to a cumulative Timeseries object accounting for the DNS traffic, which *cumulatively* interested the domains contained in the cluster.

Instances of the class DomainCluster are produced by objects of different classes extending the DomainClusterFactory abstract class, following a factory design pattern (see Figure 4.7). Specifically, objects of class DomainClusterDatabaseFactory build DomainCluster instances retrieving data from a database (the Mongo[2] no-SQL database, in the specific case), whereas objects of type DomainClusterFileFactory import data from CSV files, whose format is defined through one of the input parameters.

4.3.2 Features and Fingerprints Extractors

Both the LinguisticFeaturesExtractor and the LinguisticDescriptorExtractor classes rely on submodules to compute the linguistic features and fingerprints, as shown in Figure 4.8. Specifically, the LinguisticFeaturesExtractor relies on

[2] `http://www.mongodb.org/`

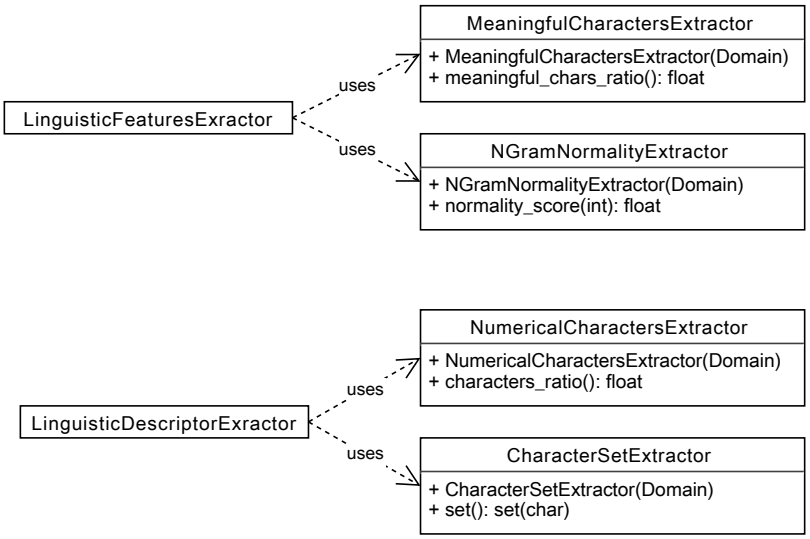

Figure 4.8: *UML diagram of the LinguisticFeatureExtractor and LinguisticDescriptorExtractor dependencies.*

MeaningfulCharactersExtractor to compute the linguistic feature **LF1**, and on NGramNormalityExtractor to compute the feature class **LF2** over domains. **LF1–2** are those defined in Section 4.2.1.1. The LinguisticDescriptorExtractor, instead, relies on the CharacterSetExtractor to compute the fingerprint **CF3** and on the NumericalCharactersExtractor to compute the fingerprint **CF4** over domain clusters. Computing **CF1**, **2** and **5** is straightforward and handled directly by the LinguisticDescriptorExtractor. The fingerprints **CF1–5** are those described in details in Section 4.2.3.

4.3.3 Modules

After introducing the basic data structures we employed and some of the tools we implemented to create them, we now describe how these components are used to build the high-level modules, which Phoenix needs to function.

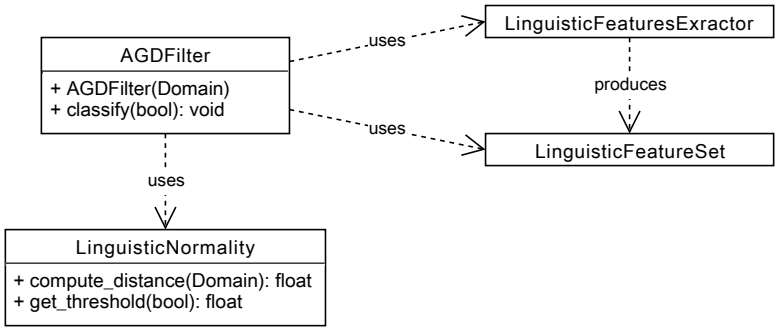

Figure 4.9: *UML diagram of the AGDFilter class and dependencies.*

4.3.3.1 AGD Filter

The class AGDFilter implements the logic behind the **AGD Filtering** described in Section 4.2.1. An object of such class is built by receiving in input a Domain object, which is assigned a label as the result of the invocation of the classify(bool) method, as shown in Figure 4.9. This label determines whether the domain seems automatically or humanly generated, depending on a loose or strict threshold (Λ or λ, specified as parameter in the method call). In order to perform this operation on domain d, an AGDFilter object leverages a LinguisticFeaturesExtractor to extract the linguistic feature vector $\boldsymbol{f}(d)$. Then, a LinguisticNornality instance is employed to compute the Mahalanobis distance $\mathrm{d}_{Mah}(\boldsymbol{f}(d))$ of domain d from the centroid of the HGD dataset. The same instance is also used to verify whether the value of $\mathrm{d}_{Mah}(\boldsymbol{x})$ is within λ, Λ or neither.

4.3.3.2 DGA Discovery

The **DGA Discovery** module is implemented by means of the DGADiscoverer class, depicted in Figure 4.10. The constructor of the class accepts a DomainCluster as input (i.e., the cluster of domains known to be malicious), while the method compute_clusters() outputs a list of objects, still of type DomainCluster, which partitions the original domain cluster grouping domains

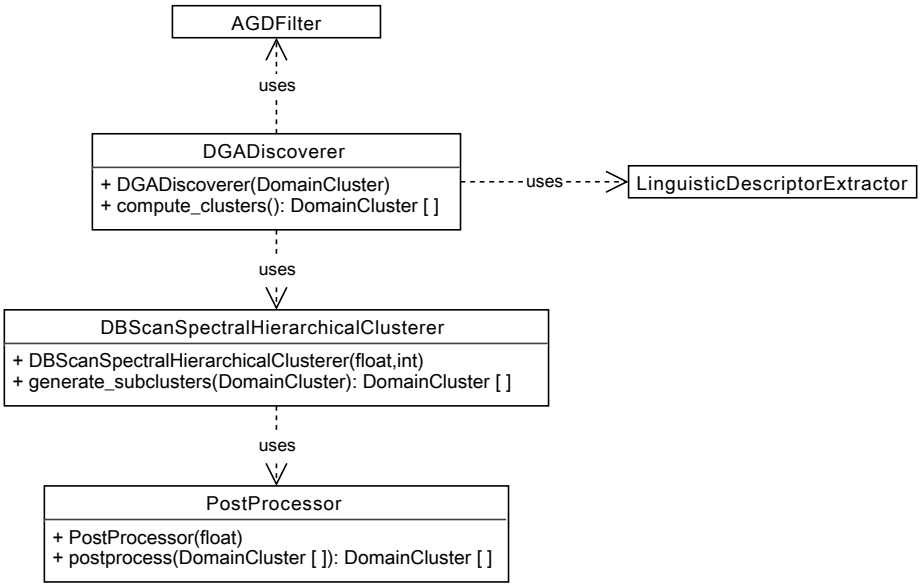

Figure 4.10: *UML diagrams of the classes involved in the* ***DGA Discovery*** *module.*

similar to each other. Using the notation we used in Section 4.2.2.1, the method returns $\mathcal{E} = \{\mathbb{E}^1, \ldots, \mathbb{E}^Q\}$, the set of clusters containing the AGDs employed by the different botnets.

The DGADiscoverer class employs the AGDFilter for the **AGD Filtering** and the class DBScanSpectralHierarchicalClusterer for the **AGD Clustering**. Last, the **DGA Fingerprinting** is performed via the LinguisticDescriptorExtractor.

A DBScanSpectralHierarchicalClusterer object is initialized with the parameters tuning the clustering algorithm: ϵ and the minimum size of a resulting cluster. This module, uses an instance of the PostProcessor class, which is configured with the proper value of γ and performs the tasks specified in Section 4.2.2.3.

The implementation of DBScanSpectralHierarchicalClusterer leverages the SciPy package (Jones et al., 2001) to efficiently handle sparse matrices. It then features an *ad-hoc* implementation of the DBSCAN algorithm, which relies on the domain-specific knowledge of single-dimensionality. Under this

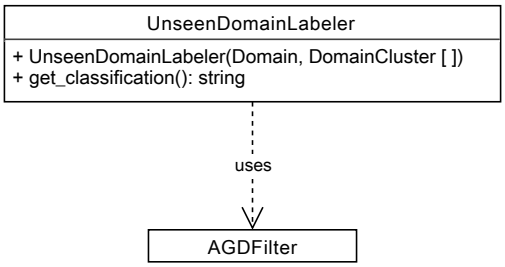

Figure 4.11: *UML of the classes involved in the* ***AGD Detection*** *module.*

assumption, in fact, the clustering algorithm can be implemented with temporal complexity of $O(n\ logn)$, instead of $O(n^2)$, where n is the number of samples to be clustered.

4.3.3.3 AGD Detection

The **AGD Detection** module is implemented by the UnseenDomainLabeler class, as depicted in Figure 4.11. Its constructor takes as input a previously unseen domain, which is to be classified (i.e., an instance of the Domain class), and a list of DomainCluster objects (namely, $\mathcal{E}$, the output of the **DGA Discovery** module). The method get_classification() returns the labels, if any, assigned to the previously unseen domain.

Again, the AGDFilter module is employed to realize the **AGD Filtering** step, when now the threshold parameter is set to the loose value λ.

Chapter 5

Experimental Evaluation

Validating the results of the Phoenix is by no means trivial, precisely because it produces novel knowledge. The reason is that we do not have a ground truth available that would allow us to validate the correctness of Phoenix quantitatively. For example, no information is available to our knowledge about the membership of a malicious domain to one family of AGDs. If such ground truth were available, then there would be no need for Phoenix. Therefore, in lack of an established ground truth, we proceed as follows. We validate *quantitatively* the internal components of each module (e.g., to verify that they do not produce meaningless results and to assess the sensitivity of the parameters), and *qualitatively* the whole approach, to make sure that it produces useful knowledge with respect to publicly-available information.

In the reminder of this chapter, first we describe the evaluation datasets we employed and the setup we used to run our prototype of Phoenix. Then, we describe the different validation experiments we performed, we give the results we obtained and their interpretations.

5.1 Evaluation Dataset and Setup

The **DGA Discovery** module of Phoenix requires a feed of DNS traffic and a reputation system that tells whether a domain is generally considered as mali-

cious. For the former data source, we obtained access to the Internet Systems Consortium's Security Information Exchange (ISC/SIE) framework[1], which provides DNS traffic data shared by hundreds of different network operators. For the latter we used the Exposure (see Bilge et al., 2011) blacklist, which included 107,179 distinct domains as of October 1st, 2012.

To validate the components of Phoenix we required other real-world datasets. To this end, we relied on publicly-available implementations of the DGAs used by Conficker (Leder and Werner, 2009) and Torpig (Stone-Gross et al., 2009a), which have been among the earliest and most widespread botnets that relied on DGAs for C&C communications. After Conficker and Torpig, the use of DGAs kept rising. With these DGAs we generated five datasets of domains, which resemble (and in some cases are equivalent to) the domains generated by the actual botnets: 7,500, 7,750 and 1,101,500 distinct AGDs for Conficker.A, Conficker.B and Conficker.C, respectively, and 420 distinct AGDs for Torpig. Moreover, we collected the list of 36,346 AGDs that Microsoft claimed in early 2013 to be related to the activity of Bamital[2].

We ran our experiments on a 4-core machine equipped with 24GB of physical memory. All the runs required execution times in the order of the minutes.

5.2 DGA Discovery Validation

In the following two sections, we cover the evaluation of the **AGD Filtering** and the **AGD Clustering** modules, which contribute to the **DGA Discovery**.

5.2.1 Step 1: AGD Filtering

The AGD filter is used in two contexts: by the **DGA Discovery** module as a pre-clustering selection to recognize the domains that appear automatically

[1] `https://sie.isc.org/`

[2] `http://noticeofpleadings.com/`

generated within a feed of malicious domains, and by the **AGD Detection** module as a pre-labeling selection. For pre-clustering, the strict threshold Λ is enforced to make sure that no non-AGD passes the filter and possibly biases the clustering, whereas for pre-labeling the loose threshold λ is used to allow more domains to be labeled. Recall that, the **Labeler** will eventually filter out the domains that resemble no known AGD. We test this component in both the contexts against the AGD datasets of Conficker, Torpig and Bamital.

The filter, which is the same in both the contexts, is best visualized by means of the ECDF of the Mahalanobis distance. Figure 5.1 shows the ECDF from the AGD datasets, compared to the ECDF from the Alexa top 100,000 domains. The plot shows that each dataset of AGDs has different distribution from the one of the dataset of HGDs. Moreover, applying a Kolgomorov Smirnov statistical test to each AGD dataset versus the HGD dataset yields p-values close to zero: This confirms that our linguistic features are well suited to perform the discrimination.

The same analyses show that Conficker and Torpig AGDs have the same linguistic features, which differ from the linguistic features of Bamital AGDs (and, of course, from the ones of non-AGD domains). We may argue that the DGAs that are responsible of such datasets are also different, whereas Conficker and Torpig rely on DGAs with very similar linguistic characteristics.

Then, we verify which fraction of AGDs passes the filter and reaches the **AGD Clustering** (Λ) step or the **Labeler** (λ). The results obtained are reported in Table 5.1 and show that roughly half of the domains would not contribute to the generation of the clusters: The conservative settings ensure that only the domains that exhibit the linguistic features more remarkably are used for clustering. Ultimately, the majority of the true AGD domains will be labeled as such by the **Labeler**. Overall, Phoenix has a recall of 81.4 to 94.8%.

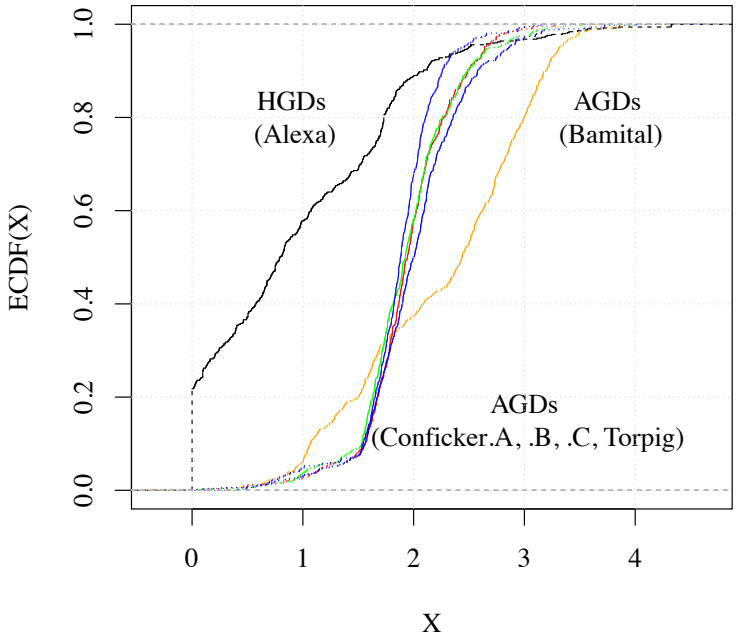

Figure 5.1: *Mahalanobis distance ECDF for different datasets.*

Malware	$d_{Mah} > \Lambda$	$d_{Mah} > \lambda$
	Pre-clustering selection	Recall
Conficker.A	46.5%	**93.4%**
Conficker.B	47.2 %	**93.7%**
Conficker.C	52.9 %	**94.8%**
Torpig	34.2%	**93.0%**
Bamital	62.3%	**81.4%**

Table 5.1: *AGD pre-clustering selection and recall.*

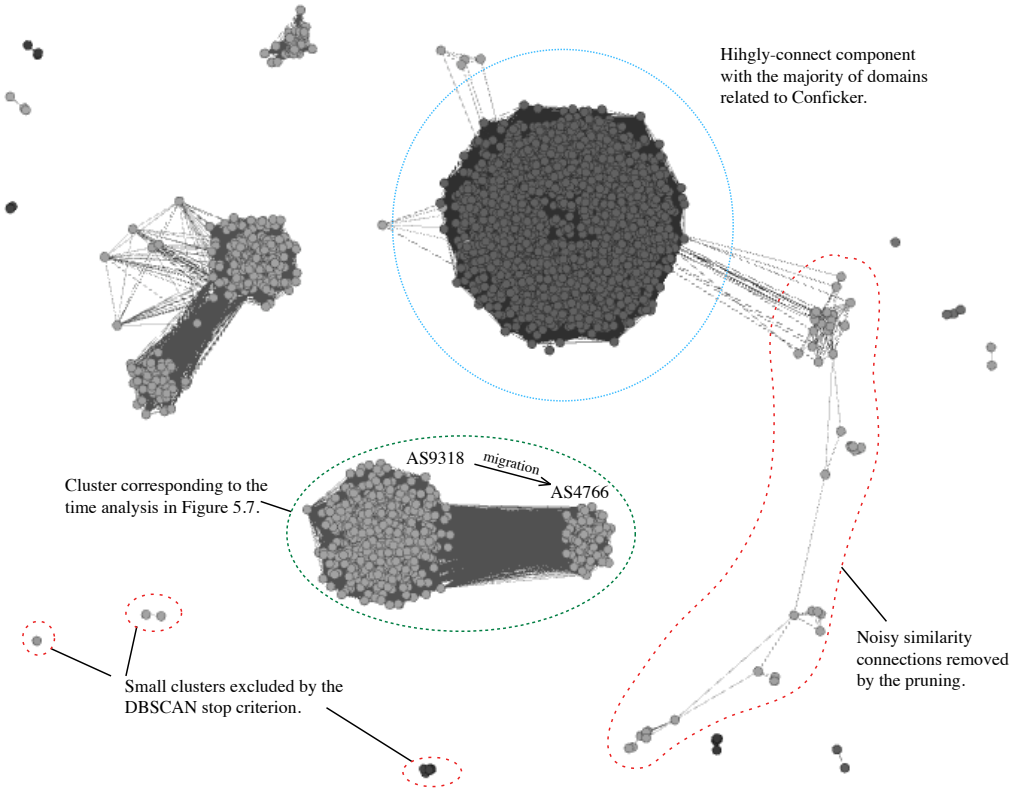

Figure 5.2: *Graph representation of the similarity matrix* $\boldsymbol{S}$ *during the first run of the DBSCAN clustering algorithm.*

5.2.2 Step 2: AGD Clustering

We ran Phoenix on our dataset and, after the first run of the DBSCAN clustering, we obtained the similarity matrix depicted in Figure 5.2 as a graph where nodes represent the domains, while edges the similarity relations. Even with one run of the algorithm, we can already see some interesting groups of domains that are similar. The purpose of the clustering algorithm is to isolate strongly-connected "communities" of domains from "noisy" and uncorrelated domains. The other annotations on the figure are clarified in the reminder of this chapter.

Cluster 6 (Bamital)

50e7f66b0242e579f8ed4b8b91f33d1a.co.cc
bad61b6267f0e20d08154342ef09f152.co.cc
62446a1af3f85b93f4eef982d07cc492.co.cc
0d1a81ab5bdfac9c8c6f6dd4278d99fb.co.cc
f1dad9a359ba766e9f5ec392426ddd30.co.cc
295e2484bddd43bc43387950a4b5da16.co.cc
501815bd2785f103d22e1becb681aa48.co.cc
341af50eb475d1730bd6734c812a60a1.co.cc
49b24bf574b7389bd8d5ba83baa30891.co.cc
a7e3914a88e3725ddafbbf67444cd6f8.co.cc

Cluster 11 (Conficker)

byuyy.biz	jbkxbxublgn.biz
kpqzk.org	tcmsrdm.org
lvzqxymji.org	fbhwgmb.info
aeyyiujxs.org	psaehtmx.info
vdrmgyxq.biz	mmdbby.biz

Figure 5.3: *Domain samples assigned to Bamital and Conficker.*

Reality Check We searched for qualitative ground truth that could confirm the usefulness of the clusters obtained by running Phoenix on our dataset. To this end, we queried Google for the IP addresses associated to each AGD cluster, with the goal of performing manual labeling of such clusters with evidence about the malware activity found by other researchers.

We gathered evidence about a cluster with 33,771 domains allegedly used by Conficker (see also Figure 5.2) and another cluster with 3,870 domains used by Bamital. Figure 5.3 shows some samples of such domains. A smaller cluster of 392 domains was assigned to SpyEye, and two clusters of 404 and 58 domains, respectively, were assigned to Palevo. We were not able to find information that could be used to label the remainder six clusters as related to a known malware.

This reality check helped us confirming that we successfully isolated domains related to botnet activities and IP addresses hosting C&C servers. The remainder of this section evaluates how well such isolation can be performed in general settings (i.e., not on a specific dataset).

Sensitivity From γ We evaluated the sensitivity of the clustering result to the γ threshold used for cluster pruning. To this end, we studied the number of clusters generated with varying values of γ. A steady number of cluster indicates low sensitivity from this parameter, which is a desirable property. Moreover, abrupt changes of the number of clusters caused by certain values of γ can be used as a decision boundary to this parameter value. Figure 5.4 shows such abrupt change at $\gamma = 2.8$.

We also assessed how γ influences the quality of the clustering to find safety bounds of this parameter within which the resulting clusters do not contain spurious elements. In other words, we want to study the influence of γ on the cluster features calculated within each cluster. To this end, we consider the cluster features for which a simple metric can be easily defined: **CF2 (Chosen Prefix Length Range)**, **CF4 (Chosen Prefix Numerical Characters Ratio Range)** and **CF5 (Public Suffix Set)**. A clustering quality is high if all the clusters contain domains that are uniform with respect to these features (e.g., each cluster contain elements with common public suffix set or length). We quantify such "uniformity" as the entropy of each features. As Figure 5.4 shows, all the features reflect an abrupt change in the uniformity of the clusters around $\gamma = 2.8$, which corroborates the above finding.

In conclusion, values of γ outside the interval $(0, 2.8)$ do not allow the clustering algorithm to perform optimally and properly separate clusters of domains.

Correctness Our claim is that the clustering can distinguish between domains generated by different DGAs by means of the representative IPs used by such DGAs (which are likely to be the C&C servers). To confirm this claim in a robust way, we evaluate the quality of the clustering with respect to features other than the IP addresses. In this way, we can show that our clustering tells different DGAs apart, regardless of the IP addresses in common. In other words, we show that our clustering is independent from the

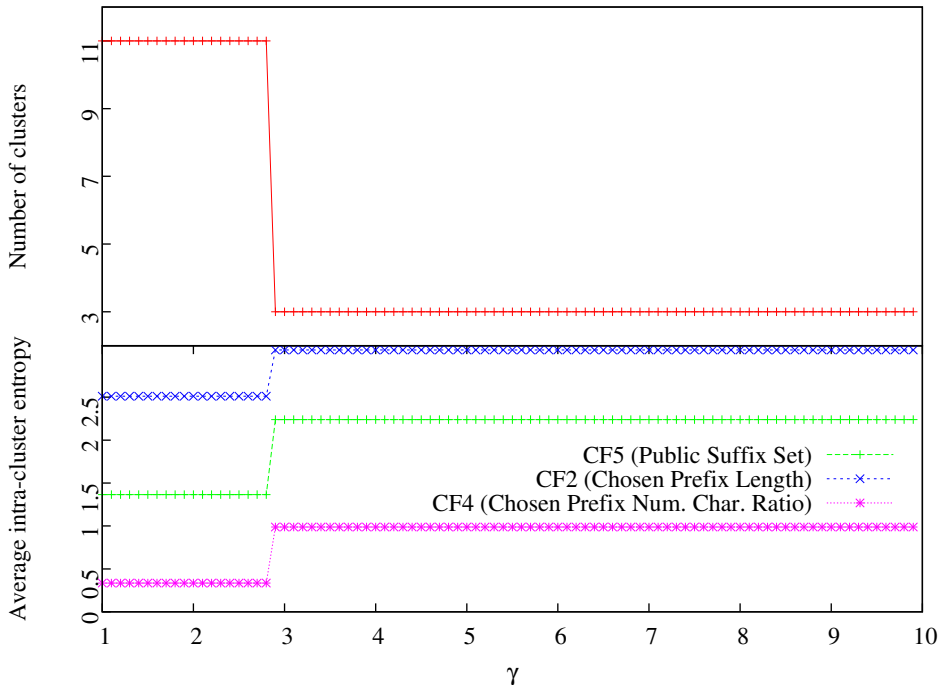

Figure 5.4: *Clustering sensitivity from parameter γ.*

actual IP addresses used by the botnets but it is indeed capable of recognizing DGAs in general.

To this end, we ignore **CF1** and calculate the features **CF2–5** of each cluster and show that they are distributed differently between any two clusters. We quantify this difference by means of the *p*-value of the Kolmogorov-Smirnov statistical test, which tells how much two set of samples (i.e., our **CF2–5** calculated for each sample in each couple of clusters) are drawn from two different stochastic processes (i.e., they belong to two different clusters). *p*-values toward 1 indicate that two clusters are not well separated, because they comprise domains that are likely drawn from the same distribution. On the other hand, *p*-values close to zero indicate sharp separation.

The results (exemplified in Table 5.2 for **CF2**) confirm that most of the clusters are well separated, because their *p*-value is close to 0. In particular 9 of our 11 clusters are highly dissimilar, whereas two clusters are not distinguishable from each other (clusters 2 and 4). From a manual analysis of

2	e-12									
3	e-8	e-9								
4	e-12	1.00	e-9							
5	e-26	e-32	e-5	e-36						
6	e-33	e-39	e-27	e-44	0.00					
7	e-3	e-4	0.01	e-3	e-16	e-33				
8	e-20	e-11	0.14	e-12	e-55	e-276	e-5			
9	e-18	e-5	e-4	e-5	e-142	e-305	e-4	e-16		
10	e-14	e-23	e-19	e-24	e-50	e-52	e-17	e-46	e-46	
11	e-22	e-28	0.61	e-31	e-163	0.00	e-8	e-27	e-82	e-52
	1	2	3	4	5	6	7	8	9	10

Table 5.2: *p-values of the pairwise Kolmogorov-Smirnov tests between clusters, considering* ***CF2*** *as representative feature.*

these two clusters we can argue that a common DGA is behind both of them, even if there is no strong evidence (i.e., DNS features) of this being the case. Cluster 2 include domains such as `46096.com` and `04309.com`, whereas two samples from cluster 4 are `88819.com` and `19527.com`.

5.3 AGD Detection Evaluation

We want to evaluate qualitatively how well the **AGD Detection** module is able to assign the correct labels to previously-unseen suspicious domains. To this end, we first run the **AGD Discovery** module using the historical domain-to-IP relations extracted from the ISC/SIE database for the domains indicated as generically malicious by the malicious domain filter (which is Exposure in our case). Once this module produced the clusters, we validate the outcome of the **AGD Detection** against another (random) split of the same type of data extracted from the ISC/SIE database. Our system never observed such data.

The result of the **AGD Detection** is a list of previously-unseen domains, assigned to a cluster (i.e., a DGA). Some examples of previously-unseen domains are depicted in Figure 5.5 along with some samples of the clusters

Previously unseen domains

hy613.cn	5ybdiv.cn	73it.cn	39yq.cn
69wan.cn	hy093.cn	08hhwl.cn	hy267.cn
hy673.cn	onkx.cn	xmsyt.cn	fyf123.cn
watdj.cn	dhjy6.cn	algxy.cn	g3pp.cn

Cluster 9 (Palevo)

pjrn3.cn	3dcyp.cn	x0v7r.cn	0iwzc.cn
0bc3p.cn	hdnx0.cn	9q0kv.cn	4qy39.cn
5vm53.cn	7ydzr.cn	fyj25.cn	m5qwz.cn
qwr7.cn	xq4ac.cn	ygb55.cn	v5pgb.cn

Previously unseen domains

dky.com	ejm.com	eko.com	blv.com
efu.com	elq.com	bqs.com	dqu.com
bec.com	dpl.com	eqy.com	dyh.com
dur.com	bnq.com	ccz.com	ekv.com

Cluster 10 (Palevo)

uon.org	jhg.org	eks.org	kxc.com
mzo.net	zuh.com	bwn.org	khz.net
zuw.org	ldt.org	lxx.net	epu.org
ntz.com	cbv.org	iqd.com	nrl.net

Figure 5.5: *Labeling of previously-unseen domains.*

where they have been assigned to. These examples show that Phoenix is capable of assigning the correct cluster to unknown suspicious domains.

Indeed, in the first case, the domains were all registered under .cn, and it is also clear that they share the same generation mechanism. In the second case, despite the variability of the eTLD (which is commonly used as anecdotal evidence to discriminate two botnets) our system correctly models the linguistic features and the domain-to-IP historical relations and performs a better labeling.

5.4 Intelligence and Insights

In this section, we describe two use cases of the **Intelligence and Insights** module, which provides the analyst with valuable knowledge from the outputs of the other modules. The correctness of the conclusions drawn from this

module is predicated on the correctness of the two upstream modules, which has been largely discussed in previous sections.

Unknown DGA Recognition from Scarce Data Our system is designed to automatically label the malicious domains related to botnet activities. This is done by using the information of the DNS traffic related to them. Interestingly, some conclusions can be drawn on previously-unseen domains even in the unlucky case that such information is missing (i.e., when no DNS data is available).

While asking on mailing lists for information about sinkholed IPs, we received an inquiry by a group of researchers on February 9th, 2013. They had found a previously-unseen list of AGDs, which resembled no known botnet. Such list was the only information that they provided us with. Phoenix labeled these domains with the fingerprints of a Conficker cluster. This result allowed the researchers to conduct further investigation, which eventually confirmed that it was indeed Conficker.B.

In conclusion, starting from the sole knowledge of a list of domains that Phoenix had never seen before, we discovered that, according to our datasets, the only DGA able to produce domains with that linguistic features was the DGA associated with the Conficker malware.

Time Evolution Associating AGDs to the activity of a specific botnet allows to gather further information on that botnet, by using the DGA fingerprints as a "lookup index" to make precise queries. In particular, we can track the behavior of a botnet to study its evolution over time.

For instance, given a DGA fingerprint or AGD sample, we can select the domains of the corresponding cluster $\mathbb{E}_{DGA}$ and partition this set at different granularity (e.g., IPs or ASs) by considering the *exact* set of IPs (or ASs) that they point to. Given the activity that we want to monitor, for instance, the DNS traffic of that botnet, we can then plot one time series for each partition. In our example, we count the number of DNS requests seen for the domains in that partition at a certain sampling frequency (e.g., daily).

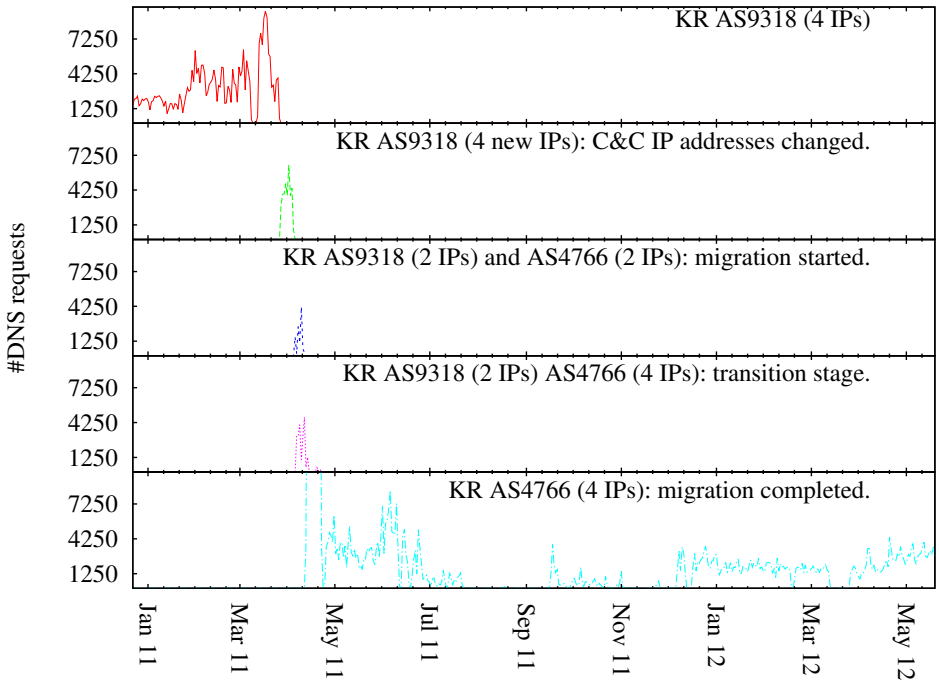

Figure 5.6: *Bamital: Migration of C&C from AS9318 to AS4766.*

The analysis of the stacked time series generated allows to draw conclusion about the behavior over time of the botnet. Figure 5.6 shows the case of (a) a migration (the botmaster moved the C&C servers from one AS to another) followed by (b) a load balancing change in the final step (the botmaster shut down 2 C&C servers thus reducing the load balancing).

In a similar vein, Figure 5.7 shows an evolution that we may argue being a takedown operated by security defenders. In particular, at the beginning the botnet C&C backend was distributed across three ASs in two countries (United States and Germany). Armed with the knowledge that the IPs in AS2637 and AS1280 are operated by computer security laboratories, we discover that this "waterfall" pattern concludes into a sinkhole. Without knowledge of the sinkholed IPs, we could still argue that the C&C was moved from some ASs to some other ASs.

The aforementioned conclusions were drawn by a semi-automatic analysis

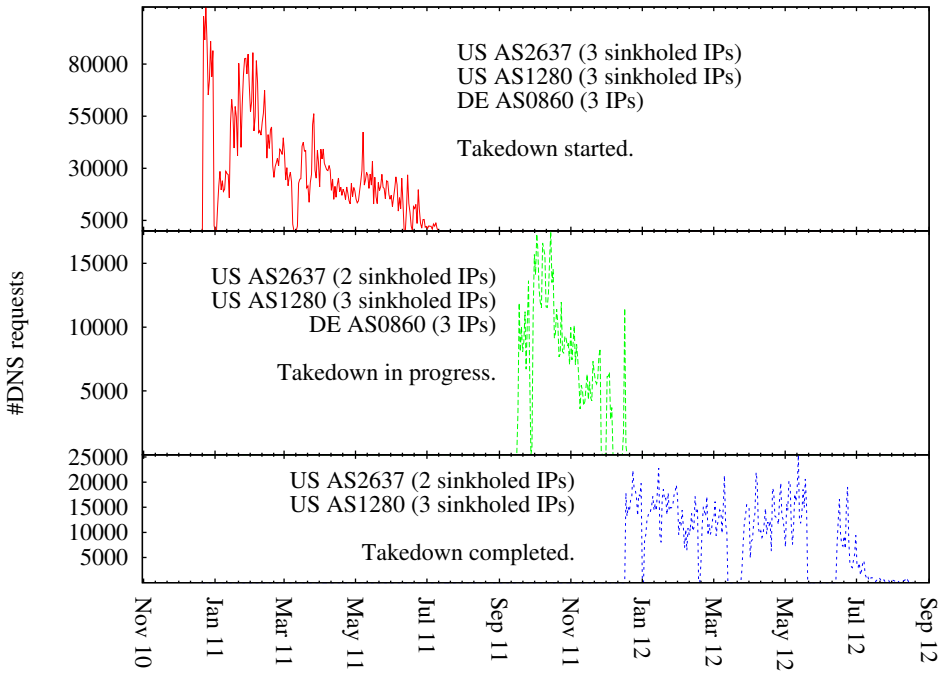

Figure 5.7: *Conficker: Evolution that resembles a C&C takedown.*

and can be interpreted and used as novel intelligence knowledge. The labels of the DGAs produced by Phoenix were fundamental to perform this type of analysis.

Chapter 6

Conclusions

In this chapter we discuss the limitations of Phoenix and we explore possible future work that may improve our system and continue our research. Finally, we draw some conclusions.

6.1 Discussion

Despite the good experimental results we described and commented in Chapter 5, Phoenix has some limitations that are hereby discussed.

Timely Detection Previous work leverages NXD responses to identify AGDs not yet registered by botmasters, as extensively discussed in Section 3.3. This allows early detection of DGA activities, since the bots yield overwhelming amount of NXD replies. Our system, instead, requires *registered* AGDs to function. Therefore, it is fed with data that takes longer collection periods. This results in a less-responsive detection of previously unseen DGAs. The advantage is that, differently from previous work, we can fingerprint the DGAs and, more importantly we lift the observation point such that Phoenix is easier to adopt. Indeed, we believe that not using NXD replies represents a strength of our work, as it makes our system easier to deploy and test under less-constraining hypotheses. In addition, given the

increasing sophistication of the malware industry, we can argue that future-generation malware may leverage mechanisms that alter the flow of NXDs, thus resulting in more stealthier DGA that would pass undetected to NXD-based techniques.

Language and Pronounceability The rationale behind the implementation of our **AGD Filter** (Section 4.2.1) is that a domain should be considered an AGD when its name appears as a randomly generated string to the eyes of English-speaking users. To this end, we compute linguistic features that are dependent from the English dictionary and capture the likelihood that a given domain name fulfils its goal of being easily remembered, pronounceable and usable by an English-speaking population. This implementation is based on a (geographic) *locality* assumption, and needs to be revisited when wishing to make Phoenix language-independent and deployable across distinct countries.

Taking into account different languages, possibly featuring sounds totally different from the English ones like the Chinese or Swedish, poses some challenges. In particular, computing language-independent features with a multilingual dictionary would flatten the underlaying distributions, rendering the language features less discriminant. To tackle this limitation, a possible solution consists in inferring the linguistic target of a given domain (e.g., via TLD analysis or WHOIS queries) so to evaluate its randomness according to the correct dictionary.

Doman encoding As of today, Phoenix can only process domain names encoded with ASCII characters. As nowadays Internet standards allow the registration of domains featuring non-latin symbols (e.g., `camtasia教程网.com`, `π.com`), this can be seen as a limitation of our system. Handling such internationalized domain names is far from trivial, as it strongly affects the underlying hypotheses behind our **AGD Filtering** module. Specifically, the definition itself of *randomly generated domain* becomes difficult to formalise in a context in which domains such as ♣→♡→♠→◇→`.com` are allowed and, possibly, benign.

6.2 Future Work

As extensively mentioned, our system relies on the Exposure system to produce a blacklist of malicious domains from a stream of DNS traffic (i.e., what we referred to as *malicious domains filter* in Figure 4.1). For the purpose of implementing a prototype of Phoenix, we found no need to reimplement Exposure and have our own reputation system run. We instead relied on Exposure website to gather updated lists of malicious domains. For the future, it should be considered to include in Phoenix the logic implemented by Exposure, which is well documented by Bilge et al. [2011], so to build a system that has no other external dependencies, beside a source of DNS traffic. Moreover, having direct access to the domain reputation system can help us tune it for our specific needs, so to speed-up our detection process.

Some of the issues discussed in Section 6.1 can be addressed as future work. In particular, a feasibility study can be conducted to assess whether it is possible to unbias Phoenix from the English language through detecting the target language of any domain name before feeding it to a specialized **AGD Filter**. Actually, this may turn out to be a wider problem, possibly shared with researchers in the field of linguistic. Addressing the other issues seems instead not straightforward as well as not priority. For example, studies should be first conducted to understand the diffusion of non-ASCII domain names and to evaluate how it changes the use of the DNS protocol.

The results produced by Phoenix could be used to build a reputation system to score the maliciousness of *domain name registrars* (i.e., the organization or commercial entities that manage the reservation of domain names). In fact, the implementation of DGA-based rallying mechanism is predicated on the possibility that botmasters can register huge amounts of domain names, possibly with automated procedures. This is possible under the assumption that some registrars are lenient or even condescending with the miscreants. A reputation system would allow to publicly pinpoint malicious registrars so to lower their reputation and credibility, as proven effective in other contexts (see for instance Stone-Gross et al., 2009b). For this and

other purposes, Phoenix should be deployed and run real-time. Moreover, its results could be made publicly available so to be employed autonomously by security researchers.

6.3 Conclusions

In this work we described Phoenix, a novel technique able to tell AGDs and HGDs apart, detect previously unknown AGDs, and provide insightful intelligence (e.g., the tracking and monitoring of AGD-based C&C domains over time). For this, Phoenix, which we have extensively evaluated, combines linguistic features and publicly-available DNS traffic information, and, contrary to the state-of-the-art, preserves client-side privacy (i.e., our approach does not rely on clients' IP, it is not affected by NAT and DHCP leases nor it requires specific deployment contexts). Furthermore, by not relying on NXD DNS replies-to-client IP correlations, we argue that Phoenix is more robust to next-generation malware evasions than the state-of-the-art systems.

Bibliography

Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of the 20th USENIX Security Symposium, USENIX Security*, volume 11, pages 27–27, 2011.

Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: detecting the rise of DGA-based malware. In *USENIX Security '12*. USENIX Association, August 2012.

Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 299–304. IEEE, 2009.

Todd M Bailey and Ulrike Hahn. Determinants of wordlikeness: Phonotactics or lexical neighborhoods? *Journal of Memory and Language*, 44(4):568–591, 2001.

Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive DNS analysis. In *Proceedings of NDSS*, 2011.

Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: Detecting botnet command and control servers through

large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

Ken Chiang and Levi Lloyd. A case study of the Rustock rootkit and spam bot. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 10–10. Cambridge, MA, 2007.

Benoit Claise. Cisco Systems NetFlow services export version 9. 2004. URL `http://tools.ietf.org/html/draft-claise-netflow-9-02x`.

Alice Decker, David Sancho, Loucif Kharouni, Max Goncharov, and Robert McArdle. A study of the Pushdo/Cutwail Botnet, 2009. URL `http://www.techrepublic.com/whitepapers/pushdo-cutwail-a-study-of-the-pushdo-cutwail-botnet/1689473`.

Julian B Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, 2007.

Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008a.

Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on Storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9. San Francisco, California, 2008b.

Jian Jiang, Jinjin Liang, Kang Li, Jun Li, Haixin Duan, and Jianping Wu. Ghost domain names: Revoked yet still resolvable. 2012.

Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python. 2001. URL `http://www.scipy.org`.

Brian Krebs. Takedowns: The shuns and stuns that take the fight to the enemy. *McAfee Security Journal*, 6:5–8, 2010. URL `http://www.mcafee.com/us/resources/reports/rp-security-journal-summer-2010.pdf`.

James Kurose and Keith Ross. *Computer Networks: A Top Down Approach Featuring the Internet*. Pearson Addison Wesley, 2006.

Felix Leder and Tillmann Werner. Know your enemy: Containing Conficker. *The Honeynet Project, University of Bonn, Germany, Tech. Rep*, 2009.

Louis Marinos and Andreas Sfakianakis. ENISA Threat Landscape. Technical report, ENISA, September 2012.

Paul V Mockapetris. Domain names - concepts and facilities. RFC 1034 (Internet standard), November 1987a. URL `http://www.ietf.org/rfc/rfc1034.txt`. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

Paul V Mockapetris. Domain names - implementation and specification. RFC 1035 (Internet standard), November 1987b. URL `http://www.ietf.org/rfc/rfc1035.txt`. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

Matthias Neugschwandtner, Paolo Milani Comparetti, and Christian Platzer. Detecting malware's failover C&C strategies with Squeeze. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 21–30. ACM, 2011.

Mark Newman. *Networks: An introduction*. Oxford University Press, Inc., 2010.

Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. Fluxor: Detecting and monitoring fast-flux service networks. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 186–206, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-70541-3.

Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early detection of malicious flux networks via large-scale passive DNS analysis. *Dependable and Secure Computing, IEEE Transactions on*, 9(5):714–726, 2012.

Phillip Porras. Inside risks reflections on Conficker. *Communications of the ACM*, 52(10):23–24, 2009.

Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. An analysis of Conficker's logic and rendezvous points. *Computer Science Laboratory, SRI International, Tech. Rep*, 2009.

Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 65–79. IEEE, 2012.

Paul Royal. Analysis of the Kraken botnet, 2008. URL `http://www.flatland.tuxfamily.org/repo/malwares/KrakenWhitepaper.pdf`.

Robert J Scholes. *Phonotactic grammaticality*. Number 50. Mouton, 1966.

Sergei Shevchenko. Srizbi domain generator calculator, 2008. URL `http://blog.threatexpert.com/2008/11/srizbis-domain-calculator.html`.

Sergei Shevchenko. Domain name generator for Murofet, 2010. URL `http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html`.

Joe Stewart. Bobax trojan analysis. *SecureWorks, May*, 17, 2004. URL `http://secureworks.com/research/threats/bobax`.

Ben Stock, Jan Göbel, Markus Engelberth, Felix C Freiling, and Thorsten Holz. Walowdac: Analysis of a peer-to-peer botnet. In *Computer Network Defense (EC2ND), 2009 European Conference on*, pages 13–20. IEEE, 2009.

Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009a.

Brett Stone-Gross, Christopher Kruegel, Kevin Almeroth, Andreas Moser, and Engin Kirda. FIRE: Finding rogue networks. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 231–240. IEEE, 2009b.

Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. *USENIX; login*, 32(6):18–27, 2007.

Sandeep Yadav and AL Narasimha Reddy. Winning with DNS failures: Strategies for faster botnet detection. *Security and Privacy in Communication Networks*, pages 446–459, 2012.

Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement*, pages 48–61. ACM, 2010.

Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. 2012.

Acronyms

AGD ..... automatically-generated domain

AS ..... autonomous system

ASCII ..... American standard code for information interchange

BGP ..... border gateway protocol

CAPTCHA .... completely automated public Turing test to tell computers and humans apart

ccTLD ..... country code top-level domain

CDN ..... content distribution network

C&C ..... command-and-control

CSV ..... comma-separated values

DBSCAN ..... density-based spatial clustering of applications with noise

DDNS ..... dynamic domain name system

DDOS ..... distributed denial of service

DGA ..... domain generation algorithm

DHCP ..... dynamic host configuration protocol

DNS ..... domain name system

ECDF ..... empirical cumulative distribution function

ENISA ..... European Network and Information Security Agency

eTLD ..... effective top-level domain

FFSN ..... fast-flux service network

gTLD ..... generic top-level domain

HGD ..... humanly-generated domain

HTTP ..... hypertext transfer protocol

HTTPS ..... hypertext transfer protocol secure

IP ..... Internet protocol

IRC ..... Internet relay chat

ISP ..... Internet service provider

ISC/SIE ..... Internet Systems Consortium's Security Information Exchange

MaaS ..... malware-as-a-service

NAT ..... network address translation

NXD ..... non-existent domain

P2P ..... peer-to-peer

SQL ..... structured query language

SSL ..... secure sockets layer

TCP ..... transmission control protocol

TLD ..... top-level domain

TTL ..... time-to-live

UDP ..... user datagram protocol

UML ..... unified modeling language

URL ..... uniform resource locator

USB ..... universal serial bus