DISSERTATION

Titel der Dissertation

Analysis of Collaboration and Agility in Internet Traffic for Malware Detection

Verfasser

Dipl.-Ing. Andreas Berger

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 786 880
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Assoz. Prof. Dipl.-Ing. Dr. Wilfried Gansterer, Privatdoz.

Abstract

Internet criminals have proven to be highly creative when it comes to inventing novel ways to extort money from their victims, and can deploy them in short time. This is largely related to the fact that most criminal Internet activity builds upon a vivid ecosystem, which offers readily available service platforms for running new malicious operations. Therefore, malicious Internet activity often shows community structure, e.g., when multiple hosts are instructed to attack the same target at the same time, or when one physical server is reused by multiple malicious services. We refer to such structured activity as *collaboration* patterns, which are indicative for malicious service platforms. Similar to benign services, these platforms are designed to be highly scalable, as more reliable operations for a multitude of victim hosts translate directly into more revenue for the criminals. However, miscreants are confronted with continuous countermeasures by governmental institutions and network security enterprises. In order to compensate for server takedowns and access blocking by network operators, their service infrastructure has to be always "on the move", and needs to employ a certain level of *agility* in its operations. For instance, malware services change the domain names under which they are reachable, and use different physical servers over time. Furthermore, malware uses Peer-to-Peer communication topologies which are continuously dynamically reorganized and therefore compensate automatically for hosts which are taken offline. This thesis addresses the problem of revealing collaborating groups of services and hosts which *jointly* engage in such evasive operations.

I describe two complementary approaches which consider different types of agility: (i) The analysis of Domain Name System (DNS) traffic addresses the *lookup* of malicious services. We derive an efficient modeling apparatus which describes the DNS mappings between domain names and IP addresses, and update the derived model automatically over time. Any mappings which do not match the model represent agile variations and are therefore candidates for evasive actions. We reveal collaboration patterns in these variations, and reliably expose domain names and IP addresses which relate to malicious activity. (ii) Large-scale profiling of end-to-end Internet *connections* captures the typical Internet usage patterns of each individual monitored host. The detection approach is based on a statistical, data-adaptive description of the network resources which are contacted by these hosts, and reveals unusual, rare connections, independently of any higher-layer information as, e.g., the actual protocol or packet payload. Both approaches ultimately map suspicious, agile Internet activity to graphs, which can then be efficiently mined for substructures which relate to malicious service platforms. In addition to revealing the individual malware sites themselves, the structural information represented by these graphs allows a human analyst to understand which elements of the malicious infrastructure are vital for its functioning, and should therefore be blocked preferentially.

Being driven by the idea of revealing patterns of malicious collaboration, a sufficiently large number of monitored hosts is required. Therefore, the proposed analysis approach is designed to scale to large networks, with possibly hundreds of thousands of hosts. It is lightweight in terms of processing requirements, and the main analysis components typically run in real-time. A prototype implementation was evaluated using multiple traffic data sets from large networks. The DNS analysis component revealed several previously unknown malicious sites, with a false positive rate of less than 0.0001%. It is more sensitive than similar approaches and requires less data at the same time. The complementary analysis of end-to-end connections is able to reduce the complexity of Internet connection graphs by up to four orders of magnitude, and proved to be an important enabler for graph-based malware detection, which allowed us to detect, e.g., injected malware traffic with perfect precision (1.0) and high recall (0.9). Finally, we demonstrate how one can jointly use both analysis approaches to reveal additional malicious activity.

Kurzfassung

Internet-Kriminelle sind äußerst kreativ im Entwickeln neuer Einkommensmöglichkeiten und können diese in kürzester Zeit einsetzen. Dies wird durch ein lebendiges Ökosystem an kriminellen Diensten ermöglicht, auf das neue Angriffe kurzfristig aufsetzen können. Die Wiederverwendung solcher Diensteplattformen führt dazu, daß der dadurch verursachte Internet Verkehr oft Zeichen von *Kollaboration* aufweist, z.B. wenn ein Ziel von mehreren Rechnern gleichzeitig angegriffen wird, oder wenn ein bestimmter Server von mehreren kriminellen Diensten verwendet wird. Ähnlich wie gutartige Dienste sind diese Plattformen hochskalierbar ausgeführt, da hohe Verfügbarkeit für möglichst viele Opfer von Internetkriminalität sich direkt in höheren Erträgen ausdrückt. Jedoch sind Kriminelle mit dauernden Gegenmaßnahmen von Regierungseinrichtungen und Unternehmen für Internetsicherheit konfrontiert. Um dadurch verursachte Serverausfälle und Zugriffseinschränkungen zu kompensieren, müssen diese kriminellen Plattformen ständig "in Bewegung" bleiben, und eine gewisse *Agilität* aufweisen. So ändern kriminelle Dienste etwa regelmäßig die Domain Namen unten denen sie erreichbar sind und wechseln die verwendeten Server. Außerdem werden oftmals Peer-To-Peer Kommunikationstechnologien eingesetzt die dafür sorgen daß einzelne Ausfälle automatisch ausgeglichen werden. Diese Arbeit beschäftigt sich mit dem Problem der Erkennung von Gruppen von Internet Diensten und Hosts die *gemeinsam* an solchen ausweichenden Manövern teilnehmen.

Ich beschreibe im folgenden zwei komplementäre Ansätze die verschiedene Typen von Agilität addressieren: (i) Die Analyse von Verkehrsdaten des Domain Name Systems (DNS) zielt auf die Namensauflösung von kriminellen Diensten ab. Es wird ein Mechanismus zur effizienten Modellierung von Zusammenhängen zwischen Domain Namen und IP Addressen entwickelt, der regelmäßige Aktualisierungen explizit unterstützt. Neue DNS Informationen, die nicht dem Modell entsprechen, werden als agile Variationen interpretiert und sind daher Kandidaten für kriminelle Aktivitäten. Das vorgeschlagene System analysiert derartige Aktivitäten weiterer Folge auf Zeichen von Kollaboration und ist in der Lage kriminelle Diensteplattformen zuverlässig zu identifizieren. (ii) Mittels Profiling von Ende-zu-Ende Verbindungen im Internet werden typische Aktivitätsmuster aller betrachteter Hosts extrahiert. Das System zur Erkennung von krimineller Aktivität baut auf einer statischen, Daten-adaptiven Beschreibung der Netzressourcen die von diesen Hosts kontaktiert werden auf, und erkennt ungewöhnliche oder seltene Verbindungen, ohne dabei auf Information von höheren Protokollschichten, wie z.B. das verwendete Applikationsprotokoll oder die tatsächlichen Nutzdaten, zurückgreifen zu müssen. Beide Ansätze stellen verdächtige Internetaktivität als Graphen dar und finden Sub-Strukturen die auf kriminelle Aktivitäten zurückzuführen sind. Zusätzlich zu der Erkennung der einzelnen kriminellen Dienste ermöglicht diese Darstellung ein Verständnis für die Kommunikationsanforderungen eines bestimmten solchen Dienstes, und unterstützt dadurch Experten bei kontrollierten Gegenmaßnahmen gegen essentielle Elemente dieser Infrastruktur.

Die Erkennung solcher krimineller Kollaboration erfordert zwangsläufig die Analyse der Verkehrsdaten einer ausreichend hohen Anzahl von Internet Hosts. Der vorgeschlagene Ansatz ist daher für die Überwachung von großen Netzen, mit bis zu mehreren hunderttausend Hosts, entwickelt. Die einzelnen Analyseschritte weisen eine niedrige Komplexität auf und erfüllen dadurch Echtzeitanforderungen. Der entwickelte Prototyp wurde mit Verkehrsdaten von großen Netzen evaluiert. Die DNS Analyse hat dabei zur Erkennung einer Anzahl verschiedener krimineller Aktivitäten geführt, wobei die False Positive Rate unter 0.0001% lag. Das System reagiert dabei empfindlicher als vergleichbare Ansätze und benötigt gleichzeitig weniger Informationen. Die komplementäre Analyse von Internet Verbindungen ist in der Lage die Komplexität der zu analysierenden Verbindungsgraphen um bis zu vier Zehnerpotenzen zu reduzieren. Dadurch konnte z.B. injizierte kriminelle Internet Kommunikation mit perfekter Precision (1.0) and hohem Recall (0.9) detektiert werden. Schlussendlich wird die gemeinsame Verwendung beider Analyseansätze diskutiert und die Erkennung zusätzlicher krimineller Aktivität diskutiert.

Acknowledgements

Ideally, a PhD thesis is the final result of a process in which motivation and commitment meet a great environment. I was lucky enough to have had the opportunity to work in such an environment, and there are numerous people who contributed in one or another way to this thesis. I would like to thank Marcin Davies, Sebastian Egger, and Matthias Baldauf for lots of cheerful moments which assured that I'd maintain the sanity of my mind while working on tough problems and being under deadline pressure (Wii Tennis!). This was complemented by the immense fun I had with the (vast!) Italian community at FTW, who were always ready to teach me new swearwords in some obscure Italian dialect. Grazie mille, non lo dimenticherò mai!

There is one person who supported me more than anybody else in my work over the last years. Alessandro D'Alconzo was never tired to discuss technical details and was always the best devil's advocate a PhD student could wish for. Thanks a million Ale!

I'd like to thank Wilfried Gansterer and Fabio Ricciato for being great supervisors, who gave me the freedom to work on the topics I liked and were always available to help me out with their experience and insight. And while I met Stefan Rührup only towards the end of my PhD, we had several inspiring discussions, for which I'm truly grateful. I'd also like to thank Eduard Natale and Mirko Schiavone, who were skilled (and patient!) enough to deal with weekly changes in implementation requirements, while we were developing the DNS analysis system.

Thanks to the open-source community, in particular to the developers of Linux, LaTeX, Python, NetworkX, Matplotlib, and Gephi, you've made my life a lot easier!

Last, but by no means least, I would like to thank Verena for plenty of moral support in bleak moments, as well as for encouragement and patience. Thank you so much!

Contents

Part I
Introduction and Background

CHAPTER 1

Introduction

The Internet is today the dominant medium for information access and distribution worldwide, and hosts everything from tiny, private sites to gigantic, commercial service networks. Large popular sites like Google and Facebook maintain enormous amounts of data and use it in rich Internet applications offered to their users. Governments increasingly employ electronic administrative services to ease the communication with, and the information of, their citizens. Financial transactions are established in the Internet as well, and the number of online banking users is steadily increasing. The advent of smart phones further boosted the Internet's importance as these "always-on, always-online" devices are deeply interwoven with our day-to-day life and business. Hundreds of thousands of "apps" for all kind of purposes are in everyone's hands, and are naturally integrated in our daily routines. The ubiquitous availability of fast Internet connections is a key enabler for all these services, and is continuing to be less and less the deal breaker for even more advanced applications. For both fixed line (Fibre to the home – FTTH) and mobile networks (Long Term Evolution – LTE) the large-scale deployment of new technologies is around the corner, and brings data rates in the order of hundreds of Mbit/s to everyone. The boundaries between "local" and "remote" are therefore increasingly blurred, which led to the widespread usage of web-based applications (e.g., Google Docs) and cloud services for storing user data. Even traditionally sealed-off networks with restrictive perimeter security employ these technologies, and see themselves slowly adapting to previously unthinkable paradigms like "Bring-Your-Own-Device" (BYOD), where employees connect their privately owned devices to company infrastructure. Internet service providers react to this development and try to get their share from this market, thereby avoiding the destiny of becoming simple "bit-pipes" with limited opportunities for future revenue. Sumptuous standardization efforts led to the development of service frameworks like the IP Multimedia Subsystem (IMS), a highly complex architecture that enriches the traditional operator infrastructure with enhanced service capabilities (e.g., video calls with guaranteed quality of service). Conceptually similar to BYOD, also this is achieved by opening up the core networks and providing interfaces for end-user applications.

These technological advances enable networked services which almost completely pervade our daily lives. Naturally, this development comes also with disadvantages, and along with legitimate services, crime prospers in the Internet. Specifically developed malicious software (*malware*) is a key driver for criminal activities. A recent report by Norton estimates the global annual cost of cyber crime as 110 Billion US$, with 556 million victims per year [4]. As a direct effect of the increasing usage of Internet services, the dependency on them increased equally, leaving us more vulnerable to adversarial actions. The secure operation and usage of these services therefore requires an apt understanding of criminal activities, as well as suitable analysis and detection mechanisms. As a motivation for this thesis, the following (incomplete) list summarizes several prominent cyber crime activities, and illustrates the severity of the problem.

- **Distributed Denial of Service (DDoS)** attacks flood victim hosts with requests, to the effect that the service becomes unreachable for everyone due to resource exhaustion. DDoS is today enough of a commonality for cyber warfare that even small and medium sized business are regularly being targeted (e.g., pizza delivery sites in Germany[1]). Over the last years, DDoS evolved from simple attacks like TCP SYN flooding to attacks on web applications, as the impact per request is significantly higher. In a recent report it was found that prices for DDoS attacks start from 2 US$ per target and hour, depending on the desired sophistication of the attack, and the countermeasures of the defenders, with discounts for longer attacks [3]. Also attacks on fixed line and mobile phones are possible now, and can be ordered for around 5 US$ per hour[2].

- **Spam** evolved from an annoyance to a multi-million dollar business in the last years, and in 2007 it was found that 95% of all emails are Spam [13]. Thereby, in particular, the money spent on defense mechanism is enormous. Anderson et al. estimate the worldwide expenditures on Spam prevention in 2011 to exceed 1 Billion US$, while the estimated income of the group of criminals who are responsible for a third of the worldwide Spam was estimated as 2.7 million US$ in the same year [8].

- **Information theft** is a wide-spread issue on the Internet today. The most prominent target is probably credit card information which created an estimated loss of 135 million £ in the UK alone [8]. There exists a vital market for such data in underground forums, with prices reportedly ranging from 10 cents to 25$, depending on the quality of the provided information (e.g., the credit card limit) [150].

  A related threat is stealing of online banking login information directly from the users' computers, e.g., by means of keylogger tools, or using advanced software that takes screenshots of displayed keypads as they are clicked by the user. The "Eurograbber" malware infected mobile devices to intercept Transactions Numbers (TANs) sent via SMS, to steal 47 million US$ from bank accounts[3].

  Also software licence keys are often stolen from malware-infected hosts, and are further being sold online. Less obviously, even account credentials for social network sites like Facebook are at risk. They are, e.g., being used for malware distribution via these channels, and thereby abuse the trusted relationships which are linked to these accounts.

- **Click Fraud** describes a variety of activities for abusing advertisement services on the Internet [49, 154]. *Advertisers* mandate *advertisement networks* to host certain ads. Websites act as publishers and display them to their visitors. For every click on a particular ad, the advertisers pay the advertisement networks, who in turn give a share of the money to the publishing website. Automatized clicks, based on malware installations, can quickly deplete the advertisers' campaign budget without any return of investment for them. Competitors take advantage in this case, as from this moment on, only their ads are being displayed. Furthermore, by using websites under the control of the fraudster, significant amounts of money can be earned.

[1] `http://www.h-online.com/security/news/item/Botnet-attacks-pizza-delivery-service-1330816.html`

[2] `http://ddos.arbornetworks.com/2012/07/ddos-attacks-targeting-traditional-telecom-systems/`

[3] `http://www.informationweek.com/security/attacks/zeus-botnet-eurograbber-steals-47-millio/240143837`

- **Phishing** is a technique to trick Internet users into handing out sensitive information like credit card numbers and online banking credentials by impersonating trusted entities. Criminals send bulk emails blindly to a large number of addresses, hoping that some recipient would respond, and thereby get "fished". Researchers estimate that 0.4% of the Internet population fall victim to phishing [62]. A variant of this concept called "spearphishing" consciously targets certain persons of which the criminals expect higher revenue in case of response. The FBI investigated 400 such cases in 2011 and found that 85 million US$ were stolen [144].

- **Blackmailing** of Internet services is common today. Typically, site operators are threatened with DDoS attacks to extort money. Often, online gambling sites are targeted by these attacks[4]. A more recent addition to this field is "Ransomware". Malware installs on the victims' computers, encrypts the users' files, and requests a ransom to be paid before access is restored. A variant of this scheme displays a fake copyright infringement message on the screen, seemingly originating from a law enforcement agency. The user is expected to pay a fine (e.g., 200 US$), and is threatened with being arrested in case of noncompliance. Symantec estimated the monthly income for a single piece of ransomware they analyzed as 394,000 US$ per month [71].

All these criminal activities have in common that rather short-term actions are meant to lead to quick financial gains. In contrast, researchers witnessed recently a new family of criminal endeavors which aim at long-term, stealthy operation to extract confidential information from, and disrupt operations of, high-profile targets. This type of activities was termed "Advanced Persistent Threats" (APTs). The level of sophistication of these malware, and the "mission-like" character of their activities suggest involvement of military bodies, and therefore governments/nations. For example, "Stuxnet" was generally believed to target Iranian nuclear facilities, and was then described as the most sophisticated malware. It was specifically developed to attack Siemens industrial software and exploited four different Windows vulnerabilities that were previously unknown [109]. In 2012, the "Flame" malware forced Iranian officials to order "*the disconnection of six of its main oil terminals from the internet, to stop the worm spreading*"[5]. Kaspersky labs stated that they consider Flame twenty times more complex than Stuxnet, and that a full analysis would probably take ten years[6].

It is not only the diversity of attacks and the technical sophistication, but also the complexity of these operations as a whole which is staggering. Just as legitimate services, malware evolved from rather simple tools to complex, diverse infrastructures. The traditional idea of a single, highly skilled hacker who breaks into a corporate site no longer holds. Rather, there exists an entire underground economy with different "professions" (e.g., developers, operators, affiliates) and "services" (e.g., hosting, client infection, credit card trading). Just as for legitimate services, there is competition between different groups, and there are tenders for malicious activities. Customers expect a certain quality of service from their purchases, e.g., a site targeted by a DDoS attack should indeed become unreachable. Suppliers which are not able to live up to these expectations are publicly discredited, and will sooner or later be excluded from the market.

The backbone of many malicious services are therefore reliable network infrastructures, e.g., well-reachable servers, which can scale to large numbers of involved Internet hosts.

[4]See e.g. `http://www.sophos.com/en-us/press-office/press-releases/2006/10/extort-ddos-blackmail.aspx`

[5]`http://www.guardian.co.uk/world/2012/may/28/computer-worm-iran-oil-w32flamer`

[6]`http://www.wired.com/threatlevel/2012/05/flame/`

▷ *Botnet* The dominant "platforms" for various malicious Internet activities are *botnets*. A ro*bot* designates an Internet host that runs an instance of a particular malware, which is remotely controllable by an adversary. The malware is typically installed without the explicit consent of the machine's owner, by exploiting a security vulnerability of the operating system, or tricking the owner to install it. Botnets constitute of thousands of such bots, and therefore possess an enormous amount of cumulative bandwidth and processing power, which is available to adversaries at no cost. Additional attack techniques are rolled out on these platforms via updates, plugins, and new releases, and thereby resemble professional (legitimate) software development. A botnet is therefore equivalent to a high-performance supercomputer, connected to the Internet with tremendous bandwidth, and in possession of all kind of sensitive user data, that is either available directly on the controlled machine, or can be retrieved from third-party services (e.g., online banking) by abusing the users' credentials. Analogous to "Software-as-a-Service", researchers coined the term "Malware-as-a-Service" (MaaS) to emphasize the notion of malware *platforms*, that can be rented on a daily basis for a variety of nefarious purposes [124].

A particularly interesting category of malware communication is Botnet *Command-and-*
▷ *C&C* *Control* (C&C), which constitutes an essential puzzle piece in this ecosystem. C&C is used to instruct bots to perform a certain task (e.g., "send 1000 Spam emails using template x" or "install latest update from URL y") and enables advanced control over the bot operation (e.g., by returning status reports after a task has been completed). C&C infrastructures are becoming increasingly complex and employ, e.g., strong cryptography for ensuring message authenticity and confidentiality. Likewise, redundant hosting strategies and load balancing are being used for hosting core services, as e.g., a Phishing site. We summarize such activity
▷ *Malicious Hosting* as *Malicious Hosting*. These do not necessarily have to be related to C&C, but share the same requirements for high availability. The efforts that criminals put into making their infrastructure more robust underline the importance that these mechanisms have for the entire ecosystem.

In this thesis, I investigate methods for detecting malware *infrastructures* by monitoring and analyzing network communication. Such techniques are highly valuable, as they can be used to better understand, e.g., botnet communication, and enable taking down the entire attack platform, instead of mitigating only single attacks. Ideally, they can even lead to the initiation of countermeasures *before* any malicious action has been carried out. Therefore, such approaches are both *sustainable* and *proactive*, and can equally serve academia and industry, as, e.g., network operators and hosting providers.

1.1 Terminology

This thesis is concerned with the analysis of data exchanges on the Internet, and is therefore rooted in the theory of computer networks. The main objective of such networks is the es-
▷ *Connectivity* tablishment of *connectivity* among a set of computers [132]. In general, a computer network consists of *nodes* (i.e., computers) and *links* (e.g., coaxial cables), and enables the establishment of connectivity between any pair of nodes. As there cannot exist links between all nodes on the Internet, most computers are *indirectly connected* and links are *shared* for carrying the data exchanges of many nodes. Peterson and Davie further distinguish between
▷ *Host* nodes which *use* the network (*hosts*) and nodes which *implement* the network (*switches*)
▷ *Packet* [132]. Internet hosts communicate by exchanging *packets* and are in the focus of this thesis.

A fundamental prerequisite for establishing connectivity is the ability to uniquely identify each host. For this purpose, each host gets assigned an Internet Protocol *address* (IP
▷ *IP Address* address). The dominant IP address version in use is IP Version 4 (IPv4), which uses four-Byte Integer values as addresses. Typically, these addresses are displayed in the decimal *Dotted-Quad* notation, in which each of the four Bytes is represented separately by a dec-

imal value, e.g., `1.2.3.4`. Groups of hosts in the same network share the same address prefix, e.g., `1.2.3.`. *Classless Inter-Domain Routing* (CIDR) defines a notation for specifying the particular prefix length, and enables the addressing of such networks. For example, `1.2.3.0/24` specifies the network *range* which contains 256 host addresses which start with the 24-bit (i.e., 3 Byte) prefix `1.2.3`. Finally, note that one or more network ranges which are operated by the same organization are called *Autonomous Systems* (ASes).

▷ *IP Network*

▷ *AS*

By means of IP addresses, switches can identify the source and the destination of packets, and *forward* them accordingly. Such information is contained in the so-called packet *header*. In contrast, the actual message to be carried from one host to the other is called the packet *payload*. For example, the payload of a packet issued by an Internet user's computer might contain the *request* for a particular Internet site, as, e.g., Google's main web site. The *response* of the addressed Google host would consist of a packet with a payload which contains the content of this site, i.e., the text and the images which should be displayed to the user. This model of data exchanges is called *client-server* communication, where the user's computer is the client, and the Google host is the server.

Many different usages of the Internet exist, and each one has its own communication requirements. Therefore, different procedures for exchanging data were introduced. These procedures are called *protocols* and define a certain message *syntax* as well as the *sequence* of message exchanges for a particular purpose. Protocols are organized in protocol layers, where each layer has a particular function. The *transport layer* is responsible for enabling data exchanges between two processes running on two Internet hosts, where each exchange may consist of multiple packets being transmitted [132]. The two main transport layer protocols in use are the *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP). For being able to address different processes on the same Internet host, TCP and UDP carry *port numbers* (*ports*) in the header.

▷ *TCP, UDP*

▷ *Port*

At the top of the protocol layer stack are *application layer* protocols. The most prominent example is probably the *Hypertext Transfer Protocol* (HTTP), which mainly defines a procedure for transferring webpages between Internet hosts [60]. Another application layer protocol of fundamental importance for the Internet (and for this thesis) is the *Domain Name System* (DNS) protocol, which relieves human Internet users from remembering IP addresses for being able to use the Internet [112]. For this purpose, the global DNS infrastructure implements a distributed database with the main purpose of translating Fully Qualified Domain Names (FQDNs) to IP addresses. Specifically, DNS clients send a query (*Q*) for an FQDN (e.g., `www.google.com`) to a recursive resolver (*RDNS*) which is usually run by the local network operator. By interfacing with the global DNS infrastructure, the resolver then attempts to find a matching address record (*A-Record*) for the queried FQDN. The IP address included in the A-Record finally enables the client to contact the requested site. We refer to these relations between FQDNs and IP addresses as *DNS mappings*.

▷ *DNS*

▷ *FQDN*

▷ *DNS Mapping*

FQDNs are in the format `A.B.C.D`, where *D* is called the Top-Level Domain (TLD), *C* the 2^{nd}-Level Domain (2LD), *B* the 3^{rd}-Level Domain (3LD), etc. DNS implements domain name aliases, i.e., there can exist a sequence of mappings from a name to a *canonical name* (CNAME). In any case, a single name can map to multiple A-Records, which usually all map to the same redundant service. Each of these A-Records includes a Time-To-Live (TTL) value after which the RDNS must delete it from its local cache, and repeat the recursive fetching as soon as a new request for this domain arrives. Note that DNS clients are not forced to query the RDNS of their local network, but can use any publicly available one in the Internet.

The following example shows a typical DNS request response. As the requested FQDN `www.google.com` could successfully be resolved to a set of IP addresses, this represents a so-called *NOERROR* response. NOERROR responses contain mappings between FQDNs and hosting IP addresses, and are the foundation of our approach.

Example DNS Query Response

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3584
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0

;; QUESTION SECTION:
;www.google.com.           IN  A

;; ANSWER SECTION:
www.google.com.     110 IN  A   173.194.113.176
www.google.com.     110 IN  A   173.194.113.179
www.google.com.     110 IN  A   173.194.113.180
www.google.com.     110 IN  A   173.194.113.177
www.google.com.     110 IN  A   173.194.113.178
```

As shown in the example, `www.google.com` resolves to many (i.e., five) addresses. Furthermore, these addresses often change over time. Consequently, the actual A-record returned to the requesting host depends not only on the query, but also on the time. This simple procedure enables *redundant hosting* of Internet services. Instead of using only a single server address, the site's content is replicated, and is therefore available at many addresses. Therefore, there is no single machine at risk to be overloaded by many requests, and the *load* is *balanced* between many servers. This strategy is often being implemented by *Content Distribution Networks* (CDNs), which provide professional hosting for a large variety of services on their IP addresses. As a consequence of this ambiguity, it is in general not possible to relate a packet to a specific service, just by looking at the IP address.

▷ *CDN*

As links are typically shared, the communication of large numbers of computers can be captured by installing *probes* on these links, which are the basic foundation of *network monitoring* [18]. Probes capture packets and forward them to an analysis system which extract a set of *traffic features* and infer, e.g., basic communication statistics like latency and loss. More complex analyses implement common communication protocols and are therefore able to read and understand (i.e., *decode*) the packets' payload. This procedure is commonly called *Deep Packet Inspection*. However, protocol decoding is costly, and therefore the maximum packet rate which can be processed in real-time is inherently limited. Furthermore, Internet criminals conceal their communication by using custom protocols which cannot be decoded by the analyzers, or *encrypt* the packet payload.

▷ *Network Monitoring*

An alternative to packet-based monitoring is *flow* analysis [38]. Instead of processing each packet, only overall statistics of a series of packets between two hosts (i.e., a *flow*) are considered. E.g., a simple flow information could contain only the source and destination address of the corresponding packets, as well as the *number* of exchanged packets. Clearly, a significant amount of traffic information is being lost by considering only flows. However, this combination of filtering and aggregation enables monitoring of large networks and is therefore widely supported by standard network equipment [37].

▷ *Flow*

1.2 Published Work

Several parts of the work presented in this dissertation have appeared in the following publications:

- Andreas Berger and Eduard Natale. Assessing the real-world dynamics of DNS. In *Proceedings of the 4th international workshop on Traffic Monitoring and Analysis (TMA)*, pages 1–14, Vienna, Austria, 2012
- Andreas Berger and Wilfried N. Gansterer. Modeling DNS agility with DNSMap. In *Proceedings of IEEE INFOCOM Workshop on Traffic Monitoring and Analysis (TMA)*, pages 387–392, Turin, Italy, April 2013

- Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Oliver Jung. Locality matters: Reducing internet traffic graphs using location analysis. In *Proceedings of the Performance and Dependability Symposium (PDS) at the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, Budapest, Hungary, June 2013

- Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Antonio Pescapè. Detecting malware activity from agile DNS mappings using graph analysis. 2013. Submitted to IEEE Transactions on Dependable and Secure Computing

For clarity, the individual publications are noted again in the introductions of the corresponding parts of this thesis, i.e., in §II (see page 41) and §III (see page 89).

The following publications are not contained in this thesis, but led to the shaping of the fundamental ideas and consider several related aspects.

- Andreas Berger, Ivan Gojmerac, and Oliver Jung. Internet security meets the IP multimedia subsystem: An overview. *Security and Communications Networks*, 3:185–206, 2009

- Andreas Berger and Mohamed Hefeeda. Exploiting SIP for botnet communication. In *Proceedings of the 5th Workshop on Secure Network Protocols (NPSEC)*, pages 31–36, Princeton, NJ, 2009

- Andreas Berger, Jacopo Cesareo, and Alessandro D'Alconzo. Collaborative network defense with minimum disclosure. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, Houston, TX, 2011

- Stefan Ruehrup, Pierfrancesco Urbano, Andreas Berger, and Alessandro D'Alconzo. Botnet detection revisited: theory and practice of finding malicious P2P networks via internet connection graphs. In *Proceedings of the INFOCOM workshop on Traffic Monitoring and Analysis (TMA)*, pages 435–440, Turin, Italy, 2013

Chapter 2

The Internals of Internet Crime

Over the last years, Internet crime underwent significant changes. The progressing complexity of the individual steps in the criminal value chain led to an increasing specialization of the involved parties. Franklin et al. were the first who studied this underground economy, and investigated the demand and offerings on this market [67]. They found many different "products" (e.g., credit card numbers) and "services" being traded there, which require a complex "industry" behind. This study was complemented by the work of Sood and Enbody who survey the commoditization of using *Crimeware-as-a-service* (CaaS) offerings in the underground market [146]. In the following, the fundamental purposes and requirements of this ecosystem are discussed. The network footprint of the involved techniques motivates the detection approaches presented in this thesis.

Malware Infection Many criminal activities start with the compromise of an Internet user's computer. Traditional, direct attacks against exposed, vulnerable services are nowadays often avoided due to the prevalence of firewalls and Network Address Translation (NAT), which causes the targeted hosts to be unreachable from the Internet [132]. Today, the dominant *infection vectors* are therefore *drive-by downloads*. Drive-by downloads occur when users visit Internet sites which host, or link to, malicious code, which is in turn downloaded to the users' machines (see [136] for a detailed study). Such unintentional downloads are facilitated by software *exploits*, which typically target vulnerabilities in the user's Internet browser, or in common third-party plugins like Flash, Java, and Adobe PDF. Once run, they cause the download and installation of a certain piece of executable malware, which in turn initiates some kind of criminal activities.

The development of exploits is today a "profession" on its own, with significant revenue for the developers[1]. Exploits are often bundled in so-called *exploit kits* and are being sold on the underground market. For example, current prices for the Phoenix exploit kit start at 2,200 US$ [108]. Optionally, buyers can also order a hosting plan, with prices for the infamous Blackhole exploit kit starting at 50 US$ per hour, with the seller taking care of server maintenance and exploit updates [69]. Grier et al. conducted a detailed study of these *exploit-as-a-service* offerings and found that 47% of an initial set of malicious domains lead, via redirects, to a site running an exploit kit. The median uptime of these sites was only 2.5 hours, after which these domains were not reachable anymore [74]. It is important to note here that the sites in the redirect chain are not necessarily knowingly participating in this plot. For example, recently the highly popular site `www.mysql.com` was compromised, and redirection code was embedded to infect visiting hosts[2]. The more sites are compromised, the more Internet hosts are redirected to the exploit kit. Single

[1] `http://krebsonsecurity.com/2013/01/new-java-exploit-fetches-5000-per-buyer/`

[2] `http://www.pcworld.com/article/240609/mysqlcom_hacked_to_serve_malware`

sites which are cleaned from the redirection code can be quickly replaced by new ones, for maintaining a steady stream of new victims. The final exploit site, however, requires reliable hosting as its availability directly translates into money.

Another "profession" deals with using these exploits to install malware on Internet hosts. These are in general neither the same criminals who developed the exploits, nor the ones who actually later command the malware. Rather, there exist *Pay-per-install* (PPI) programs as platforms for trading such services. Clients order installs of a particular piece of malware, and pay a certain sum, typically per 1,000 installs [32]. Conversely, PPI affiliates sell their install services via these sites. A variety of methods exist, from sending Spam that includes links to exploit sites, to direct installs on vulnerable computers, to packaging with illegal software downloads [74]. A recent variant, Traffic-PPI, aims at steering traffic to sites running exploit kits, and criminals are being payed per successful installation. For example, Wondracek et al. conducted a study on platforms for trading traffic for adult Internet offerings. They found that by spending 160 US$ for traffic being redirected to their servers, they could have infected 20,000 hosts [155]. The security measures of the traffic trading sites they investigated were low in general, and mostly non-existing. For example, no checks were made whether the site to which users are directed via embedded advertisements was actually safe.

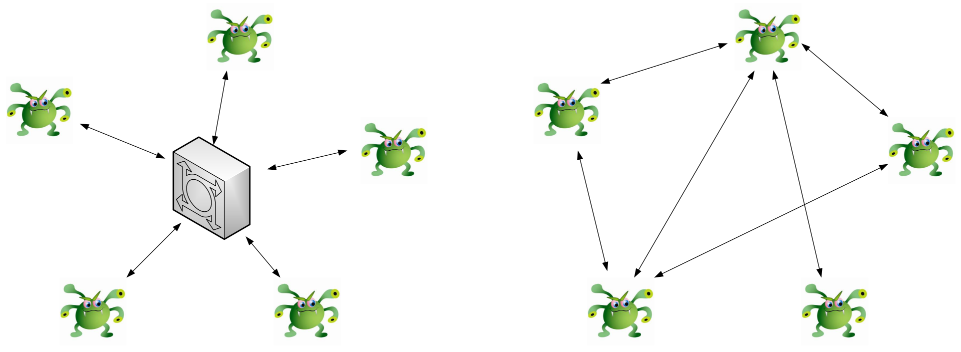
Malware requires this infrastructure not only for hosting exploit kits, but in addition also for many other nefarious purposes. Konte et al. studied the dynamics of scam hosting infrastructures (used, e.g., for Phishing campaigns) [98]. Scam links are usually distributed via email Spam or other unsolicited communication, and are rather short-lived. The authors found that the IP addresses to which the scam sites resolve change rather quickly over time. However, IP addresses are being reused by multiple site names, and therefore reappear often. In a related effort, Stone-Gross et al. discuss the underground economy of fake antivirus software [147]. They note that a functional hosting infrastructure is vital for these criminals, as the victims must be able to reach a website for making their purchase.

Command and Control The initial infection of large numbers of Internet hosts with any kind of malware is often only the first step. Ultimately, criminals aim at extracting information (e.g., login credentials, credit card data) or rent infected hosts out to other parties (e.g., for DDoS campaigns) in return for a rental fee. This implies that some kind of *communication channel*, between the infected hosts and the criminal controlling them, is required. As introduced in the previous section and in accordance with previous work, we refer to such hosts as *bots*, and call their controller *botmaster*. A group of related bots is thus commonly called a *botnet*, and their communication channel with the botmaster is known as *Command-and-Control* (C&C). As C&C enables reconfigurations, updates, launching of new attacks, and reporting, it is of tremendous importance to botmasters.

Dagon et al. provide a taxonomy of botnet communication and define three metrics [46]: (i) the *size*, i.e., the number of bots; (ii) the *diameter*, i.e., the average communication path length between two bots; and (iii) the *redundancy*, i.e., the alternative means of communication a botnet has available. The more alternative paths P_i a botnet can use, the more resilient it is to mitigation actions, i.e., blocking of individual paths. Given a detection probability ϵ_i for each path, the authors define an upper bound for the probability that all n paths can be detected, and further blocked, as

$$\prod_{i=1}^{n} \epsilon_i \leq (1-\alpha)^n$$

where α is the probability that all communication nodes in a path are cleaned. Improved detection techniques let α increase, and botmasters therefore attempt to increase n in order to maintain the botnet's resilience. However, a large number of paths requires more


(a) Centralized botnet topology

(b) P2P botnet topology

Figure I.1: Common botnet topologies.

maintenance effort, and a large diameter increases the communication latency. The C&C structure has therefore a direct impact not only on the resilience and robustness, but also on the botnet's utility. This is further being described in term of effectiveness, efficiency, and robustness in [45], and was extended later by Zhang et al., who note that also the cost for botnet construction and maintenance, as well as the scalability of the C&C model is important [161]. Dagon et al. found diurnal changes in malware activity, due to the fact that victim machines are typically turned off during night [47]. Malware communication therefore also needs to be robust against these constant changes.

Two main C&C models exist [50]. By far the most prevalent is centralized C&C, where all bots connect to a particular C&C server (Fig. I.1a). The main advantage of this scheme lies in the high effectiveness and efficiency, which also entails a low command latency. The limited robustness against mitigation actions has been addressed by using redundant central servers, which often change quickly over time. This is often done by rotating DNS mappings of C&C servers, and is called "Fast-Flux", which we further discuss in §4.1.2. The second existing C&C model is based on Peer-To-Peer (P2P) networking (Fig. I.1b). Although being highly robust against counteractions, it entails a high maintenance effort due to P2P *churn*: bots which are joining or leaving require the continuous reorganization of the P2P overlay. The command latency is higher due to the longer average path length (as compared to the centralized C&C model), and a botmaster runs the risk that an infected machine is switched off again by its owner before a bot could receive new commands.

2.1 Malware Communication Examples

The following examples illustrate several techniques used for malware communication. Although this list is not exhaustive, it serves as a motivation for the following chapters.

The *Srizbi* botnet originally contacted C&C server IP addresses that were hardcoded in the malware binary. After a takedown attempt by the security firm FireEye, the bots fell back to generating domain names using a built-in algorithm that was seeded with the current time of day, and started querying the DNS for these names. Knowing in advance which domains are going to be queried, the botmaster registered some of these domains, and reestablished Srizbi's C&C communication [114].

The estimated number of hosts infected with the *Rustock* malware was 1.1 to 1.7 million in 2010. The botnet was responsible for 48% of the worldwide Spam, before it was taken down in a concerted action [104]. Rustock used two tiers of C&C servers, of which one with 26 different proxies was relaying the communication between bots and the actual servers. The malware resolved C&C server addresses using DNS, and fell back to a set of hardcoded

IP addresses when this failed [31]. Note that Rustock did not contact the IP addresses returned via DNS, but rather used them as seed for a specially crafted algorithm to generate the actual C&C server IP addresses.

The *Storm* malware was mainly used for email spamming, phishing, and DDoS attacks [83]. It used a customized version of the Overnet P2P file-sharing protocol for communication, using randomly selected UDP ports. The messages between individual bots were encrypted. A new command (e.g., containing new Spam templates) could be published by any of the hosts in the P2P network, and was then automatically distributed throughout the botnet. In addition, Storm used Fast-Flux, for further increasing the resilience to countermeasures.

The *Conficker* botnet received much attention by both scientific and non-scientific publications, especially in the beginning of 2009, and even led to the formation of a dedicated working group[3]. The botnet's size was then estimated to be somewhere between 500,000 and 9,000,000 individual bots. The criminals behind Conficker released several revisions of the malware, which are known as *Conficker.{A,B,C,D,E}*. All variants use centralized C&C, although later versions implement a P2P protocol used exclusively for bot software updates. The notable difference to Storm is here, that Conficker does not possess an initial list of peers, but requires probing for finding other bots. The ports used for P2P communication are randomized[4]. The C&C messages are sent on ports 80 (HTTP) and 445 (Microsoft CIFS), clearly chosen so that the botnet's traffic is hidden in the vast amount of legitimate communication using these ports. Conficker makes extensive use of Fast-Flux and uses fresh DNS mappings every few hours. The details of this procedure offer an excellent insight into the rationale of botnet design [135].

Zhang et al. provide an in-depth analysis of botnet communication and design a "hardened" botnet [161], which is highly robust against countermeasures which aim at disrupting its communication. Their approach, Bot-Enclave, provides a decentralized, hierarchical C&C control structure with three types of botnet nodes, namely *commanders*, *sub-leaders*, and *bots*. The proposed structure changes over time, i.e., commanders are replaced by newly infected machines and new sub-leaders are regularly chosen. On the network layer, Bot-Enclave nodes are assumed to be connected using P2P or random topology models. Sub-leaders control groups of bots, which may overlap for redundancy reasons. All communication in the botnet is authenticated and encrypted, and commands need to be received from multiple sub-leaders before they are executed, in order to anticipate that particular sub-leaders can be captured and manipulated. However, while this approach is highly effective and robust, Bot-Enclave causes plenty of traffic due to regular reorganization, authentication, re-keying, and command redundancy.

For completeness, note that malware communication is under continuous development, and criminals are highly creative when it comes to inventing new, unconventional mechanisms. For example, some botnets abuse Twitter for C&C[5]. Nagaraja et al. speculate about more advanced techniques and investigate botnet communication via probabilistically unobservable channels [115]. Their *Stegobot* uses steganography for hiding messages in images, which are then delivered via (existing) social network services. As another example, we discuss the potential abuse of Voice over IP (VoIP) infrastructures for botnet communication in [24]. At the core of these platforms is the Session Initiation Protocol (SIP). SIP is a versatile and generic mechanism for looking up communication peers, and is expected to play a key role in next-generation telecommunication services [23]. Botnets could abuse these readily available infrastructures, and take advantage of using a highly reliable, distributed platform for free. As this kind of communication requires no network traffic to be

[3]`http://www.confickerworkinggroup.org`

[4]The randomizer is seeded with the local IP address and the current time of day.

[5]`http://ddos.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/`

exchanged directly between two bots (or a bot and the botmaster), it cannot be revealed easily by network monitoring. Detection and mitigation of this kind of communication should therefore be done by the respective service provider (e.g., Twitter), and is considered out of scope for this thesis.

CHAPTER 3

Methodology and Scope

Defensive approaches against misapplications of the Internet are being researched since the beginnings of computer networking [52]. In general, we differentiate *host-based* countermeasures and *network-based* ones. Host-based approaches are installed directly on the host to be protected (e.g., virus scanners). Network-based systems analyze the Internet traffic to and from the host, and, e.g., block illegitimate communication (using firewalls). In this thesis, I focus on the latter approaches, and base my work on two main observations about malware communication requirements, to which I refer to as *collaboration* and *agility* in the following.

1. **Collaboration**: the perilousness of Internet malware stems from the collaboration of a large number of hosts in a common task, and their ability to dynamically commence new tasks. Most prominently, botnets require coordination among the bots, and communication of C&C commands and statistics. Bots are therefore linked either directly (using P2P networking), or indirectly (using centralized C&C servers). The analysis of such collaboration patterns can therefore support botnet C&C detection. Similar collaboration patterns can be observed for other types of malware, e.g., when many victim hosts contact the same Phishing site. From the viewpoint of the criminals, these hosts "collaborate" for the success of the Phishing campaign, and are therefore characterized by a specific networking profile. Likewise, multiple external hosts can collaborate in hosting the same malicious FQDNs. ▷ *Collaboration*

2. **Agility**: malware needs to strike a balance between stable "service" availability, and continuous reconfiguration. Modern malware activities are based on service platforms, and need therefore to be able to evade detection mechanisms and sustain take-down actions by legal authorities. Static configurations are doomed to fail as soon as critical components (e.g., the only C&C server) are blocked, and bots become uncontrollable. Therefore, criminal Internet activities employ a certain level of *agility*, which involves different aspects of their communication patterns. Examples include the quick rotation of FQDNs of C&C servers ("Fast-Flux", see §II), peer-to-peer (P2P) techniques that constantly reconfigure the overlay network, and redundant malware hosting strategies with different server IP addresses being used at different times. Measuring agility patterns addresses the basic need of malware to communicate, while being forced to constantly change its networking footprint. ▷ *Agility*

The basic idea is therefore to use network monitoring to (i) reveal traffic patterns which represent a certain degree of agility, and (ii) further discover groups of hosts that are directly or indirectly linked via such connections. However, there is a hitch: some benign Internet traffic is highly agile itself. The prevalent use of Content Distribution Networks (CDNs) and cloud services has replaced the previously dominant strong relation between a service (e.g.,

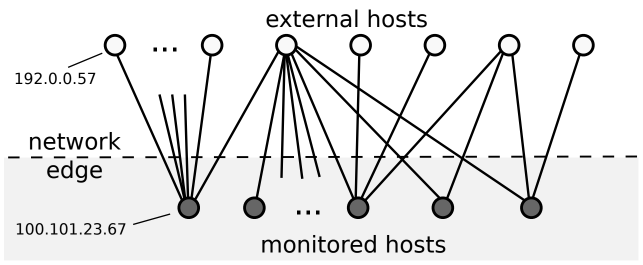


Figure I.2: Scenario overview: connections between monitored (internal) hosts and external hosts are observed. Traffic data is captured at the network edge, and is further analyzed for signs of malware communication.

Google search) and the network resources (i.e., IP addresses) hosting it. Hosting a service using CDNs implies that there is no such relation anymore, as services are dynamically assigned to IP addresses based on availability and current load conditions. Therefore, even for benign services, there are regular changes in the relations between FQDNs and external hosts. The notion of agility is therefore not a binary but rather a continuous one, which depends on the particular service and IP addresses in question. A fundamental requirement for a detection approach is therefore a modeling apparatus, that is able to *describe* the normal level of agility to expect for a specific service/IP address, and thereby *enables* the detection of agility deviations.

Detection of patterns of collaboration further requires that sufficient malware samples are available in the monitored network, so that group activity patterns can be observed. Internet service providers (ISPs) suggest themselves for this kind of network monitoring, as they both possess the required technical means as well as an adequate customer population. Furthermore, the main reason for the flourishing of malware is the fact that the criminals can operate mostly without limits. They are not confined to any legislative boundaries and are truly international in their activities. It is so far unclear who has the responsibility (and the authority!) to monitor, and eventually disrupt, e.g, botnet operations. While there are examples of elaborate, concerted counteractions[1,2] [133], typically they focus on the "server-side" infrastructure of botnets, and do not affect the bots themselves. Recently established initiatives in Germany[3] and Australia[4] are supported by the respective governments and a large number of local ISPs. They exploit the unique monitoring opportunities of ISPs, as well as their established, trusted customer relationships, to inform users about potential malware infections and offer remediation tools and technical support. A study by Wood and Rowe suggests that customers are even willing to pay for security protection plans [157], which could make it profitable for ISPs to take over this responsibility.

Therefore, ISP networks are defined as the intended deployment scenario for the analysis approaches presented in this thesis. In particular, I consider network monitoring architectures as shown in Fig. I.2, where an ISP monitors the communication between monitored hosts (i.e., customers) and external hosts (i.e., "the Internet"), and is therefore able to "see" all communication that traverses the network edge.

▷ *Monitored Host*

[1]http://www.securityfocus.com/brief/855

[2]http://nakedsecurity.sophos.com/2011/03/24/one-week-later-rustock-and-pharmacy-express-still-flatlined

[3]www.botfrei.de

[4]http://www.icode.net.au

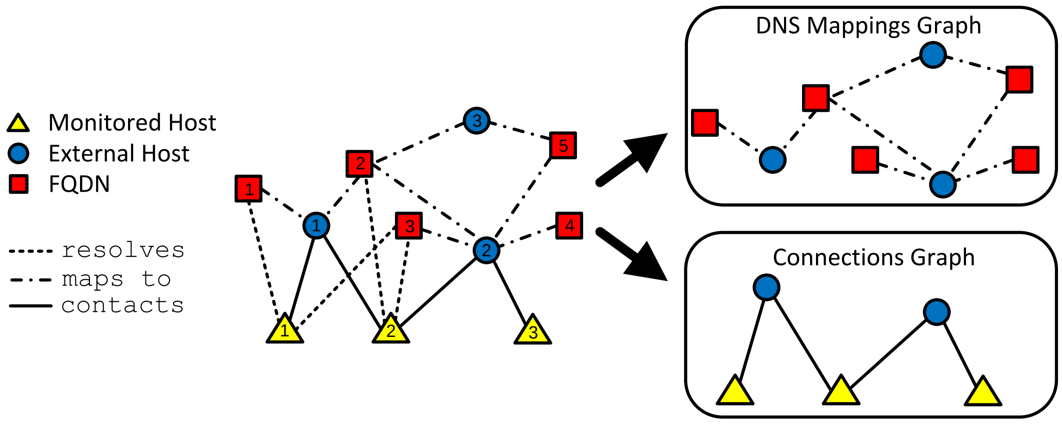


Figure I.3: Basic concept: we consider relations between Internet entities (FQDNs, internal/external hosts). Changes in these relations represent a certain degree of agility, which we analyze structurally as graphs for revealing evidence of collaboration.

3.1 Methodology

In the following, I sketch the fundamental methodology applied for this work, which sets the scene for further in-depth discussion in §II and §III. The basic foundation of my approach is the exclusive analysis of traffic features which can be readily extracted, and do not require sophisticated extraction techniques like Deep Packet Inspection (DPI). These suffer from performance limitations which make them unfit for the intended, ISP-based monitoring scenario. Furthermore, and this being the main reason for sticking with simple traffic features, higher level traffic data as, e.g., HTTP payload is often obfuscated or encrypted, and botnets can be easily reconfigured to hide from detection approaches which depend on such corresponding patterns.

Based on a careful evaluation of the existing literature (see §4), I identify two traffic features which represent now the main pillars of this thesis. On the one hand, the analysis of *DNS traffic* (see §II) is feasible even for vast numbers of monitored hosts, and cannot be obfuscated without impairing its function. Just as many other Internet services, malicious services make extensive use of the DNS, and may therefore be detected. On the other hand, I consider the *communication endpoints* of Internet connections (i.e., IP addresses) (see §III), a traffic feature that is equally hard to manipulate as these addresses are fundamentally required for forwarding packets on the Internet.

The relations between (monitored and external) Internet hosts on the one side, and FQDNs and (external) Internet hosts on the other side, can be described by graphs. The nodes of the graph represent Internet hosts and FQDNs while the graph's edges describe different kinds of relations between these nodes. Note that these graphs contain static snapshots of the aggregated traffic exchanges of a particular monitoring epoch, e.g., one hour.

Fig. I.3 illustrates this basic concept. Monitored hosts *resolve* the FQDNs for certain services by issuing DNS queries. The DNS infrastructure responds with one or more IP addresses which *map* to this FQDN (at this moment in time). Ultimately, the monitored hosts *contact* the services by establishing a connection to these IP addresses. The representation of this information as graphs enables us to reveal *groups of hosts and FQDNs* which are densely connected to each other, and therefore form a graph *community*. Members of the same community are therefore involved in an activity which shows collaboration patterns, as they occur when multiple hosts are contacting the same malware platform.

However, also monitored hosts contacting benign services result in the same collaboration patterns. For example, many hosts resolving and contacting a popular web site like `www.google.com` would likely end up being assigned to the same community. Therefore, we require a technique to *filter* these graphs, so to remove benign services, and reveal only patterns of *malicious* collaboration.

For being able to make this distinction, we exploit the fact that malware activity implements dynamic methods for *evading* detection and mitigation measures. For example, Botnet C&C infrastructure is often located in networks which are normally not contacted by a particular host, and therefore these connections stand out when, e.g., a particular server is suddenly contacted by many hosts which previously did not contact this location. Likewise, due to the platform-like character of malicious services, a changing set of different malicious FQDNs maps to the same external hosts. In the graph representation, this results in the appearance of new nodes and edges over time. We summarize these dynamics as network traffic *agility*.

By monitoring the Internet traffic from the vantage point of a network operator, we can track significant changes in this graph, and thereby find both *agile DNS mappings*, as well as *agile IP connections*. As illustrated in Fig. I.3, we address these two analysis problems separately. First, in §II, we describe a DNS-based malware detection system that analyzes the relations between FQDNs and hosting IP addresses. Secondly, we propose a detection approach for IP-level communication in §III. The structure of the corresponding (sub-)graphs (see Fig. I.3) reveals structural patterns of collaboration, as they appear, e.g., when one FQDN maps to multiple IP addresses over time, or when multiple hosts contact the same set of C&C servers. This allows us to identify monitored hosts which are victims of Internet crime. Furthermore, it provides an understanding of the network-level activity patterns and logical connections between different malicious sites, and therefore enables targeted countermeasures which consider the actual infrastructure components.

Note that it is not necessarily required that each monitored host engages in DNS activity *and* IP-level communication with a particular external host. For example, monitored host "3" in Fig. I.3 does not resolve any FQDN and still contacts external host "2". Likewise, monitored host "1" resolves FQDN "3", but does not contact IP address "2" to which this FQDN maps. Some malware activity is detectable only either in the DNS mappings graph or the connections graph, while other malicious activity can be revealed because of the combination of DNS requests to suspicious sites and peculiar IP connections.

As a fundamental principle, the design of the presented approaches acknowledges the always-changing nature of Internet traffic, and continuously updates the derived models of "normality". It implements therefore the change detection paradigm, where changes are reported once and are then integrated in the model building process. Note that this implicitly assumes that malware activity is more agile than legitimate one, and therefore triggers recurrent change events, which make the malware detectable by the following collaboration analysis. The presented techniques go to great lengths to assure that "normality" in legitimate Internet traffic is well represented, and that a priori assumptions about what is normal are as limited as possible. Rather, the system should adapt to variations, and adjust its working point as it goes on.

The design is driven by the idea to provide evidence for the existence of malware platforms as a whole, and focuses less on the detection of every single infected machine. We envision a system that is able to provide an early first alarm, that already includes a topological overview of the suspicious connections. Such systems, which favor broad coverage over detailed, per-host analysis, should consider the integration of a human analyst in the detection process. Following this first alarm, targeted investigation should then be initiated, which is out of scope for this work.

3.2 Evaluation of Results

The detection of criminal Internet activity requires the differentiation of benign and malicious traffic patterns, and is thus a *classification* problem. For evaluating the results of our proposed detection systems we therefore employ the standard measures for classification system evaluation. In particular we consider *binary* classification tasks and data sets which include *events* that are either *relevant* (i.e., malicious) or *not relevant* (i.e., not malicious). For example, an event can correspond to a TCP packet being observed by a network probe or to a DNS request response being captured.

In particular, we distinguish the following types of classification results:

- **True Positives** (TP): the number of events which are *relevant* and which were *correctly* classified as relevant.
- **False Positives** (FP): the number of events which are *not relevant* but were *wrongly* classified as relevant.
- **True Negatives** (TN): the number of events which are *not relevant* and which were *correctly* classified as not relevant.
- **False Negatives** (FN): the number of events which are *relevant* and which were *wrongly* classified as not relevant.

Note that the assessment of these figures requires the availability of *ground truth*, i.e., the knowledge about the correct classification of any single event in a considered data set. This is often difficult to achieve for real-world problems, and is further being discussed for network-based malware detection problems in §4.3.

Building on these measures, two important concepts are commonly used for evaluating classification results. The *specificity* or *True Negative Rate* (TNR) of a classifier is defined as TN/(TN+FP). Similarly, the *sensitivity* or *True Positive Rate* is defined as TP/(TP+FN). Another important concept is the *fallout* or *False Positive Rate* (FPR) which is defined as FP/(FP+TN), and which is among the most commonly used concepts for evaluating and comparing the performance of malware detection systems. Together, these concepts are used for creating Receiver Operating Characteristic (ROC) curves (see, e.g., [58]) which show the overall classification performance. ROC curves are graphical plots which compare FPR (on the X-axis) and TPR (on the Y-axis) for several parameter configurations (e.g., classification threshold settings). Ideally, a classifier would yield TPR=1.0 and FPR=0.0 for adequate parameter configurations, in which case the ROC curve would pass through the upper left corner of the plot.

However, as discussed by Davis and Goadrich, ROC curves have limitations for highly skewed data sets [51]. They show that *Precision* and *Recall* (PR) plots generally give a more informative picture of a classifier's performance. Precision is defined as TP/(TP+FP) and the definition of Recall is equivalent to sensitivity. Similar to ROC curves, PR plots compare Recall (on the X-axis) with Precision (on the Y-Axis). Ideally, both Precision and Recall equal to 1.0 for adequate parameter configurations, in which case the PR curve passes through the upper right corner of the plot.

Furthermore, Axelsson noted an important aspect of using FPR for assessing network traffic classification results [14]. He pointed out that, for practical reasons, the *absolute* number of false positives is often more important than FPR in this context. Even very low FPRs may correspond to a high number of false positives. Therefore, Axellson suggests that traffic analysis systems should be tuned such that the absolute number of FPs is low enough in order to enable the (manual) investigation of the results by a human operator in realistic time.

3.3 Outline

In §4, the current state-of-the-art of malware detection and complex network analysis is discussed. This serves as a motivation for the two main parts of the thesis, §II and §III. In the first main part §II, we discuss the analysis of DNS agility, with a particular focus on the agility of *benign* DNS activity. Based on this analysis, a novel, versatile, and highly sensitive detection system is proposed, which allows for revealing malware activity in real-time using graph analysis. The second main part §III describes a novel idea for large-scale malware detection, based on an efficient, data-adaptive profiling technique for capturing the normal, baseline activity of Internet hosts. Similarly to the DNS analysis approach, we employ graph analysis for revealing malicious, collaborative activity. Finally, we discuss the joint usage of both systems in §IV. Throughout the thesis, the proposed approaches are evaluated in an extensive set of experiments using large traffic data sets from ISP networks.

CHAPTER 4

State of the Art

In this chapter, the related work from the fields of malware detection and complex network theory is discussed. First, a review of a variety of malware detection approaches is provided in §4.1, which ultimately motivates the focus on *network-based detection approaches*. One of the main objectives of this thesis is the investigation of the analysis of collaboration patterns using graph representations. Therefore, I review the available literature of *complex networks* in §4.2, in order to identify the properties and limitations of relevant graph analysis algorithms. Finally, the evaluation of network-based detection approaches is discussed in §4.3. The remainder of this thesis is based on the findings of these analyses, and is driven by the idea of a two-step detection system, where the initial network-based detection approach is specifically designed with the following graph analysis in mind.

4.1 Malware Detection

Ever since the first appearance of malware, detection approaches have been proposed, so to avoid its installation on the victim host, or enable its removal. A basic requirement for any such system is the reliable differentiation of malware and legitimate software. This initially led to the development of *signatures* which are today still a cornerstone of many detection approaches. For example, virus scanners analyze code binaries and match them against a large corpus of semi-automatically generated signatures. These signatures can be derived using *static analysis* of malware binaries, i.e., analysis of the malware program without executing it. Often, this requires the use of disassembler tools for deriving a human-readable representation of the instructions that comprise the malware program. However, malware developers and distributors reacted to these techniques by obfuscating the binaries and introducing ambiguity in the code that heavily complicates static analysis [113]. For example, the execution path of a malware sample can depend on system parameters like the current time or the operating system version. Today, the dominant obfuscation technique are so-called *packers* [77]. These tools encrypt the actual malware payload and decrypt it only in-memory on the victim machine. By regularly changing the encryption keys, it becomes impossible to identify a given malware sample without decrypting it, and the generation of reliable signatures becomes significantly more complex. A related challenge for detection systems is *malware polymorphism*, i.e., code which changes its syntactic form, but not the execution semantics [66], and therefore avoids malware signatures very effectively[1].

The limitations of static analysis led to the development of alternative approaches. *Dynamic analysis* executes malware and extracts distinct activity patterns from its runtime behavior. Thereby, it benefits from being able to observe the actual execution of the pro-

[1]`http://www.nist.gov/itl/upload/Symantec-Comments-to-BotNet-FRN-11-14-11.pdf`

gram. While static analysis can only provide signatures that are based on the *syntax* of the code, dynamic approaches can therefore rely on its *semantics*, i.e., the actual operations it performs while running. Several systems have been developed (e.g., *Norman sandbox*[2], *bitblaze*[3], and *ether*[4]), and have demonstrated their potential in a great many of research publications. For example, Bayer et al. give a comprehensive overview of current malware behavior [17]. They describe how they use their "Anubis" system for automatic analysis of Windows binaries. Each received binary is analyzed for five minutes at most. During this time, the invocation of system calls and network traffic is recorded and is afterwards compared to other traces. The authors note that this type of analysis executes malware in a virtualized environment and is therefore computationally expensive.

Numerous dynamic analysis approaches have been presented, many of which address highly specific problems of these systems which often stem from the fact that there exists an arms race between miscreants and security researchers. Analysis systems like Anubis are built on top of system emulators or virtual machines, and is has been demonstrated that software is able to detect that it is running in a such environments [137, 128]. Consequently, malware authors implemented these checks and do not commence malicious activities in suspected analysis environments, or even initiate misleading behaviour so to distract researchers. Malware has also been observed to deliberately delay its execution, as it is widely known that analysis frameworks have only limited time available for each binary they process. Execution-stalling code introduces delays before the actual malicious operations are initiated, and is therefore difficult to detect [97]. A complete summary of the large field of dynamic malware analysis is out of scope for this thesis, and the reader is referred to the comprehensive survey of Egele et al. [55].

In addition to being used by security specialists for dissecting malware operations, dynamic analysis can also be deployed directly on the Internet end host to be protected, similarly to the widely deployed virus scanner products. Kolbitsch et al. propose such a system in [96]. They extract behavior graphs from the system call activity of a malware instance, and show that these graphs can be used to build accurate malware profiles. Their system efficiently matches the actual runtime activity of unclassified software against the profiles, and determines if it corresponds to known malware activity. This system therefore relies on *semantic signatures*, as opposed to the typically used syntactical ones. It introduces a certain overhead though (between 5% and 39% in the initial experiments presented in [96]), and, being running on the end-host, suffers from the fact that malware might compromise the analysis system itself.

Another branch of related detection approaches focuses on the execution of malware in a controlled analysis environment to extract *communication models* [99, 140]. For example, Cho et al. manually reversed the C&C protocols of the *MegaD* botnet, and infiltrated its communication [35]. They revealed the underlying network architecture and found that multiple servers are being used for different purposes. While this kind of analysis is highly valuable, it is also very costly in terms of time spent for understanding the protocol, and is therefore suitable only for specific malware instances which infected a large number of hosts, and are persistent over time. Therefore, Wondracek et al. propose an *automatic* system for protocol reverse engineering [156]. A more advanced system was later proposed by Wurzinger et al. [158]. The authors employ change point detection strategies to discover significant alterations in the outgoing traffic of a particular host (e.g., an increase in connections as observed during a scanning procedure or a large number of SMTP packets as seen during a spamming attack), represented as feature vectors and called the *behavior profile* of a host. In particular, the CUSUM algorithm is used to detect relevant deviations

[2]`http://www.normanshark.com`
[3]`http://bitblaze.cs.berkeley.edu`
[4]`http://ether.gtisc.gatech.edu`

of the normal, expected networking behavior. When a change point is detected, the system analyzes the incoming traffic to the particular host, which was recorded in the n seconds before this change happened ($20s \leq n \leq 100s$). By clustering multiple of these traffic "snippets", common token sequences are found, which are then used to generate *command models*. Together with the *response models*, created by computing the average of the individual behavior profiles, bot signatures are derived. The system relies on visible networking activity *immediately* after a command was received. Randomized delays or "time-bombs" can therefore circumvent this approach. Jacob et al. further progressed in a similar direction, and complement host-based analysis with network communication analysis [89]. Based on a set of labeled malware connections, their "JackStraws" system can infer C&C message templates. As host-based execution patterns are known, non-malicious communication, initiated by the malware binary, can be excluded from the results. An interesting addition to such systems is the work by Neugschwandtner et al. [118]. Their approach aims at triggering failover C&C strategies by blocking access to the primary C&C servers. Malware binaries are run in a contained environment, and multiple execution paths are revealed by continuously evaluating the effects of blocking certain communication. Finally, Perdisci et al. execute malware binaries in a contained environment and focus on automatic clustering of HTTP communication patterns, which enable the generation of traffic signatures for each identified malware family [131].

Both dynamic analysis of the malware binary, and malware execution in contained environments have their own shortcomings. First of all, one has to possess a particular malware binary before being able to analyze it. This is a challenging task, given the appearance of large numbers of malware binaries every day, due to, e.g., do-it-yourself development kits which allow for the quick creation of a new, custom piece of malware [123]. Besides inherent performance limitations of the analysis platforms, another main problem lies in the fact that these approaches rely on the assumption that an executed malware binary behaves exactly as if it would be installed on a victim machine. This is not always the case, and criminals are implementing strategies for detecting such analysis environments. Furthermore, such analysis approaches cannot provide an assessment of the malware activities *at large*, as, e.g., in a particular operator network. This is, however, required for, e.g., understanding how many hosts are infected, and which countermeasures should be deployed for protecting the victim users. Finally, the execution of malware binaries raises ethical questions, and requires tight containment strategies which can impair the analysis [99]. In the following, we therefore discuss *network-based* detection approaches, which are based on passively monitoring malware "in the wild", and enable an understanding of the "big picture" of malware activities of potentially thousands of hosts, over extended periods of time.

4.1.1 Network-based Malware Detection

In contrast to the previously discussed approaches, purely network-based detection systems have no control of, and no insight in, the execution flow of the malware binary. Therefore, such systems are in general less precise, and are prone to misclassify the network activity of individual hosts. The strength of network-based approaches rather lies in their ability to reveal large-scale malware activity, as, e.g., a massive botnet with thousands of hosts. Rather than, e.g., identifying the precise C&C protocol in use, such systems are better used for understanding that there *exists* some suspicious activity in the network, and therefore *enables* a suitable reaction, as, e.g., targeted mitigation actions or more costly in-depth analysis, if required. The following literature review focuses on such approaches.

A number of systems focus on malware communication via particular protocols/ports. Goebel and Holz analyze Internet Relay Chat (IRC) data and reveal nicknames which are related to malware activity [72]. IRC servers are often used for botnet C&C, where each bot joins a certain chat channel on a particular server, for uploading status information and receiving new commands. Strayer et al. gave a comprehensive overview of the state of the art [148] of botnet communication detection, and presented a novel detection approach. While their own work and experiments were targeted exclusively at the detection of centralized C&C botnet communication using the IRC protocol, especially the discussion of their classifier and correlation stages are universally relevant, and laid the ground for later approaches. The system clusters traffic data using flow characteristics (e.g., the protocol type and the flow duration) and employs machine learning for revealing groups of malicious hosts. However, the system can be defeated by bots which mimic benign traffic features when communicating.

Gu et al. propose "BotSniffer" [76], a detection system for botnet C&C traffic over IRC and HTTP. The system is based on spatial-temporal correlation and similarity of the traffic patterns of Internet hosts, as they occur when many hosts contact the same set of C&C servers using, e.g., equally sized messages, or when they attack the same victim hosts using DDoS. The authors consider *homogeneous crowds* of hosts, which exchange the same messages with a certain server. Benign services with similar traffic patterns are being removed using a set of whitelists. Another proposal by Gu et al. is "Botminer" [75]. Their core contribution is the correlation of both results from the analysis of the botnet activity (*A-plane* – e.g., sending Spam emails) and the C&C traffic (*C-plane*). Based on that, their proposal can detect IRC, HTTP, and P2P botnets. The core input data in use are so called *C-flows*, consisting of several average numbers for the number of flows per hour, the number of packets etc. The proposed system does not take sequences of events into account, but considers each one in an isolated manner. This was later addressed by host-based approaches like the one of Wurzinger et al., which we discussed in the previous section [158].

Karasaridis et al. discuss the challenges of detecting C&C communication in [93]. They present a detection system which is based on flow data (as opposed to packet data) and is therefore suitable for deployment in large (ISP) networks. Based on initial, labeled data, e.g., from Spam analysis, the proposed system reveals C&C servers and infected hosts. The approach builds on the assumption that the same malware binary would behave similarly on different hosts (e.g., use the same port numbers). The authors note that botnet activity is highly dynamic, and bots switch to new C&C servers every few days.

Wang and Yu develop a system for detecting botnet communication by using traffic aggregation [152]. Their approach focuses on botnets with centralized C&C topologies which communicate via TCP protocols. In order to improve the detection results, a number of high-volume flows (like email traffic) are not taken into consideration, as they are not commonly used for botnet communication. Furthermore, flows with high data rates (like HTTP downloads) are filtered out, as large file downloads are assumed to not represent typical botnet behavior. To further improve the results, both static and dynamic whitelists are used. The system aggregates the remaining m traffic flows and classifies them according to destination address and port. Accordingly, the normalized compression distance (NCD) is used to find similarities in the payloads of the analyzed traffic. From the resulting m-dimensional distance matrix, Wang and Yu compute a weighted variant of the local clustering coefficient for each node and thus find clusters of flows with similar payloads. Similarly, to address encrypted traffic, the flows are clustered according to the sequences of packet sizes and packet interarrival times. Those groups of flows showing high values for both clustering coefficients are finally suspected to originate for botnet communication. However, the approach has a number of deficiencies. As the authors note themselves, encryption paired with randomized message padding and random idle periods between two messages would seriously impair the detection results.

Gao et al. present an online approach to measure the number of outgoing connections (i.e., the outdegree) of a large number of hosts to a set of destinations during a given time window [70]. They call such hosts *stealthy spreaders* and demonstrate how they can reliably quantify large-scale scanning events when the number of flows that relate to scanning activities accounts for more than 2% of all flows. The presented approach specifically targets distributed, collaborative scanning events, where each host only scans a moderate number of destinations and which are typical for botnet activity. The system has an extremely low memory footprint of only 24 kB for monitoring 1 million distinct source addresses and operates fast enough to be used for online traffic analysis. However, the presented system comes with a number of limitations. Although the authors recognize the dynamics of Internet traffic and employ change detection strategies to detect anomalous outdegree variations, they employ only two consecutive time intervals for their analysis. Also, there is no continuous retraining intended, as the proposed system uses a static set of training data. Finally, the focus on scanning activity and the restricted view on *unsuccessful* connections only, limit the applicability of this approach for this thesis. Especially the decision not to store source information in favor of efficient counting hinders the usage for malware countermeasures, as the system's result can only tell that there is some suspicious activity in the network, but cannot provide information which allows one to investigate the details of this activity.

The "DISCLOSURE" system by Bilge et al. is based on the analysis of Netflow data, which is readily available in many large networks, and therefore allows for straight-forward deployment in real-world environments [26]. It benefits from observing multiple malware specimens of the same kind, whose correlated activity patterns may disclose their existence. The approach requires no insight into the actual payload of the malware communication, and is rather based on overall statistical information, as, e.g., the flow sizes and the connection inter-arrival times. Similarly, the "CoCoSpot" system by Dietrich et al. is mainly based on message length sequences [54]. Both systems require labeled traffic data for training the detection algorithm, and are therefore restricted to the detection of malware activity which is similar to labeled data. This is a general restriction of approaches which are based on machine learning techniques, and requires a thorough understanding of the quality of the used data set, as well as a careful configuration of the system parameters [145]. Furthermore, the usage of obfuscation techniques (e.g., random packet padding [54]) or randomization of the activity of individual bots, so to achieve desynchronization of the botnet's activity [26], are rather simple to deploy and may have a significant impact on the detection performance.

4.1.2 DNS-based Malware Detection

Many malware detection approaches exploit the availability of malware binaries or employ labeled traffic data. This information is used to "seed" the analysis systems which then project the knowledge about a limited number of Internet hosts involved in malware activity on a larger host population, and therefore reveal, e.g., large-scale botnets. An alternative seed is DNS traffic, which can be observed for various malware "services" (e.g., when the initial exploit triggers the download of a malware binary, or when a bot resolves the domain name of a C&C server). In contrast to most other malware communication, one can take advantage from being able to decode the communication protocol (i.e., DNS), as its specification is publicly available and DNS traffic is not (and will not be [12]) encrypted. While such DNS traffic is not malicious by itself, knowing which FQDNs represent malicious services, and on which IP addresses they are hosted, enables, e.g., the straight-forward extraction of all hosts contacting these services. In the following, we provide an overview of the existing approaches in this field.

DNS-based malware detection is based on revealing the inherent *agility* in such DNS traffic [125, 81, 9, 129]. Internet criminals need to guarantee reliable hosting of their "services", but are constantly facing countermeasures, e.g., by network operators or hosting providers. This implies that a particular FQDN cannot remain mapping to a certain IP address, but must rather change the IP address from time to time, so that the service remains reachable when the IP address is being blocked. Extremely quick changes to these mappings, with possibly hundreds of IP addresses being used for a single FQDN within, e.g., only one day, are known as *Fast-Flux* [117]. Similarly, one service is often identified by multiple FQDNs, in case an FQDN ends up on a DNS blacklist, and therefore becomes unreachable. These FQDNs do typically change over time, and are often based on *Domain Generation Algorithms* (DGAs) which dynamically generate the FQDN to be queried, based, e.g., on the current time of day [160].

Villamarin-Salomón and Brustoloni presented an early approach for DNS-based botnet detection [151]. Their system focuses on the detection of Fast-Flux. Based on the expectation that large botnets would cause exceptionally high DNS request rates, the employed methodology considers the request and reply rates of DNS servers. A modified form of the Mahalanobis distance is used to represent the difference between normal and anomalous network behavior. The authors were however not successful in reliably distinguishing regular DNS queries from those caused by bots, which resulted in many false positive classifications.

Most existing systems build on DNS data collected by a network operator. Antonakakis et al. propose a dynamic reputation system for domain names, called “Notos” [9]. The system processes DNS query responses from a passive DNS (pDNS) database and extracts a set of 41 features from observed FQDNs and IP addresses. A labeled training set of both benign and malicious FQDNs is used to derive eight domain classes (five benign, three malicious) by clustering the FQDNs according to their feature vectors. After this initial training, subsequently seen FQDNs are then assigned to one of these classes, according to their features. Ultimately, Notos derives a dynamic reputation score for each observed domain name. In a similar spirit, Bilge et al. present their “EXPOSURE” system [27], which requires less features (15) and less training data (1 week), but has no notion of the relations between domains mapping to the same set of IP addresses. Both Notos and EXPOSURE employ the Alexa list[5] for whitelisting popular domains. Both systems are rather complex and require careful tuning of the (significantly many) analysis parameters. Furthermore, both systems are based on machine learning, and therefore require significant amounts of labeled training data. Similar to the approaches discussed in the previous section, even in case such data is available, such systems require a thorough understanding of the specifics of these data for attaining meaningful results (see also [145]).

A number of approaches target specific aspects of malicious DNS activity. Yadav et al. discuss the detection of algorithmically generated domain names [160] using string analysis. Similarly, Jiang et al. investigate the failed DNS queries for such domain names, and reveal groups of hosts which share a particular failure pattern, and are therefore infected with the same malware instance [90]. As these systems are limited to a subset of malicious activity, they are complementary to the more generic approaches discusses above.

Other approaches aim exclusively at detecting Fast-Flux activity, and are therefore narrower in scope than the system we develop in this thesis [129]. Most recently, Perdisci et al. proposed “FluxBuster” [130]. The system performs a sequence of four steps: (i) All information collected about a domain name d in an epoch $\varepsilon = 1$ day is aggregated. (ii) Domains that are unlikely to represent flux activity are removed. More precisely, FluxBuster removes domains with a high average TTL, less than three IP addresses, and a low IP address diversity (i.e., IP addresses which do not belong to a large number of different /16 networks). (iii) The system finds clusters of domains which share a significant number of IP addresses. (iv) Finally, a supervised statistical classifier algorithm labels domains as flux or non-flux. At the end of an evaluation epoch, a set of 13 features is computed for each cluster. A C4.5 decision tree classifier, trained using a four months long training set and tuned with an extra one month of data, ultimately decides whether the domains and IP addresses in a cluster are related to malicious flux services. The authors state that the detection of flux domains can

[5]`www.alexa.com`

take up to 30 hours, which prevents real-time detection. Furthermore, the system requires a minimum of 30 IP addresses per domain cluster per day to detect flux activity, which limits the sensitivity of this approach.

Another branch of approaches bases on data collected at a top-level domain's (TLD) operator. Hao et al. study DNS lookup patterns by analyzing lookups to the TLDs `.com` and `.net` [80]. Thereby they concentrate on the distribution of queries from recursive resolvers, so to assess the cumulative DNS activity of entire networks. They find that malicious domains are more likely to be queried from a dynamic set of networks, and that many domains are either entirely legitimate or malicious. A related effort is described in [10]. This kind of analysis is restricted to detecting malicious activity using a subset of the existing domains, which are hierarchically directly "below" the TLD (e.g., `example.com`). Furthermore, it can observe only a small share of the queries of the actual clients, as most DNS answers are available at the recursive resolvers, and are therefore absorbed by them. These approaches are therefore complementary to the approaches presented in this thesis.

Choi and Lee propose "BotGAD", a system which targets the DNS group activity patterns of the monitored (client) hosts [36]. The approach is based on the idea that malware infected hosts would periodically query the same (malicious) domains. The authors evaluate the approach experimentally and reveal 20 previously unknown botnets.

Hu et al. analyze the global IP usage patterns of Fast-Flux botnets by conducting measurements from 240 geographically distributed network locations [85]. They mainly aim at differentiating malicious "fluxy" activity from legitimate CDNs. Therefore, they continuously query a set of 5,169 suspicious domains to see how the returned DNS mappings develop over time. Most notably, they find that legitimate CDNs are trying to mostly return an IP address that is geographically close to the querying client, addresses involved in malicious activity are much more uniformly distributed over the world.

Virtually all analysis approaches use publicly available blacklists[6] for result verification and algorithm training. However, these lists include domain names from a variety of sources, and the domain's activity must therefore not necessarily be related to dynamic DNS usage. Therefore, although a malicious domain is detected by a system, it is unclear if it was found because there is any observable dynamics in its activity, or just because the classifier was wrongly trained. Conversely, Alexa's list of popular domains[7] is often used for training machine learning algorithms or whitelisting. However, as only second-level domains are listed, the granularity at which sites are differentiated is rather low (i.e., `a.example.com` and `b.example.com` are considered the same). Due to the fact that the quality and precision of such lists is therefore unclear, the approach presented in this thesis avoids using them where possible.

4.2 Complex Networks

The discussion of malware detection systems demonstrates a rich variety of available tools. In the focus of this thesis are network-based approaches, which have been used in the past with a large number of features extracted from traffic data. The envisioned analysis of collaboration patterns using graphs is mostly hindered by the vast sizes that these graphs usually attain. The number of graph nodes corresponds to the number of observed hosts and can quickly grow to millions for ISP networks. This section is therefore devoted to a review of relevant graph analysis techniques from complex network theory. The properties and limitations of these techniques define the requirements that have to be imposed on the data extraction and preprocessing steps of network monitoring systems. For a more complete overview of complex networks I refer the reader to the excellent book by Newman [121].

[6]E.g., `www.malwaredomainlist.org`

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

4.2.1 Terminology

▷ *Graph* Complex network theory studies the structural characteristics of graphs $\mathcal{G}(V, E)$, where $V = \{v_1, v_2, \dots\}$ is a set of *vertices* (or *nodes*) and $E = \{e_1, e_2, \dots\}$ are the *edges* connecting them. The terms "network" and "graph" are often used interchangeable. In this thesis, I use "graph" to avoid confusion with computer networks. Edges have in general no direction, i.e., the edge between v_i and v_j equals the edge between v_j and v_i ($(v_i, v_j) = (v_j, v_i)$). In contrast, the edges of directed graphs (*DiGraphs*) have defined start and end nodes (i.e., $(v_i, v_j) \neq (v_j, v_i)$). The edges of *weighted graphs* are annotated with weights w, and are used to characterize application dependent properties (e.g., the communication bandwidth between two Internet nodes). Another important class are *bipartite* graphs. They consist of two sets of nodes, and contain no edges between nodes from the same set. A *subgraph* of any type of graph consists of a subset of the graph's nodes and edges. An *induced subgraph* is built from a subset $s \subset V$, and contains all edges of the original graph for which both start and end node are contained in s.

A graph is fully described by an *adjacency matrix* with dimensions $|V| \times |V|$, that contains the weights w of the edges between two nodes. In case of non-weighted graphs, it contains zeros and ones only, where ones indicate the presence of an edge between two nodes and zeros indicate the absence of an edge between two nodes, conversely. The number of outgoing edges of a node v is called the node's *outdegree*, and the number of incoming edges is called *indegree*, respectively. The *degree* of a node corresponds therefore to the sum of indegree and outdegree. The *degree distribution* is an important means for characterizing a graph, as it provides a simple evaluation of the overall structure, and thereby reveals the type of graph one is dealing with. Several different types exist, and a multitude of complex network models have been proposed for describing them. Of particular interest for monitoring Internet connections and relations between hosts is the Watts-Strogatz graph [153]. Their *small-world model* is related to Milgram's famous experiment which is now widely known by the phrase "six degrees of separation" [111] – on average, the distance between two US citizens in a graph describing their social relationships is six. Many complex networks have been found to follow the small-world paradigm, including the relations between sites in the World-Wide-Web (WWW) [6]. Small-world networks are therefore well connected, i.e., almost all nodes are quickly reachable from all others.

The Watts-Strogatz model assumes that the node degree follows a Poisson distribution, with a certain characteristic average degree. In contrast, Barabási and Albert found that many large networks show a degree distribution that follows a power-law, and coined the term *scale-free networks* for them [15]. The fundamental concept behind such networks is the existence of relatively few *hubs* with a large degree, and a large number of nodes with a low degree (i.e., a *long tail* distribution [7]). In the context of monitoring Internet traffic, a prime example is the popularity of Internet sites: a vast number of hosts contact highly popular sites like Google and Facebook, while the majority of the remaining sites are contacted by only few hosts [107]. A graph representation of these connections between Internet hosts and services therefore resembles the scale-free model.

A variety of other graph characterization metrics, besides the degree distribution, exist. A core concept is the traversal of a graph, by following the edges starting from a particular start node. These traversals are called *walks*, and are often used to characterize the graph's structure. A *random* walk is a graph traversal where every next hop is chosen randomly, by selecting a route via a random one of the edges of the current node. Another important characterization metric is the *clustering coefficient*. It evaluates the number of "triangles" (i.e., three nodes that are each connected to the two others), and thereby measures the overall tendency to cluster together. Furthermore, advanced algorithms like PageRank [127] and HITS [94] annotate nodes with a value that corresponds to their "importance" in the graph, and thereby, e.g., allow for identifying the most important ones. The survey by Costa et al. [44]

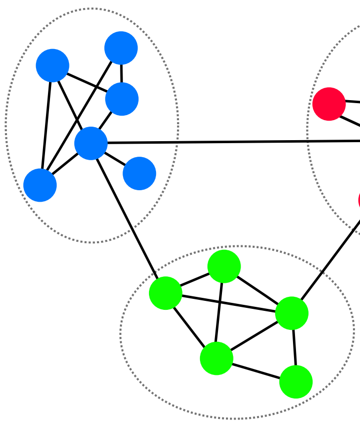


Figure I.4: Example graph with three communities.

gives an excellent overview of the field of existing measurements for the characterization of complex networks.

4.2.2 Structural Properties and Community Detection

A fundamental task for many applications, including Internet traffic monitoring, is the identification and characterization of sub-structures in a graph. The most simple scenario concerns graphs consisting of disconnected groups of hosts, i.e., where there exist pairs of nodes for which no walk between them exists. These groups are called *connected components*, and can be found in linear time (in terms of $|V|$ and $|E|$) [84]. However, in the context of network monitoring, many graphs represent the scale-free, small-world nature of the Internet. They often consist of only a single, giant connected component, which contains all nodes [91]. The further separation of this component poses a significantly harder problem than finding connected components.

▷ *Connected Components*

These sub-divisions of graph components are called *communities*, and the respective field concerned with finding them is referred to as *community detection* [44]. This problem is hard as, in general, the number of communities is a priori unknown. Furthermore, the very definition of what constitutes a community is dependent on the actual application. Fundamentally, we require a quality metric that allows us to identify where to “cut” a graph such to retrieve “meaningful” communities. Fig. I.4 illustrates the basic problem: how can we describe the relation between the nodes in the graph such that we can understand that they separate in three communities?

We require a method for deciding whether a group of nodes constitutes a community, or represents just a random graph density fluctuation. This is achieved by establishing a *null model* as a reference. A standard null model is a random graph, i.e., a graph that includes edges between nodes with equal probability, and by definition does not contain any communities. Newman adopts this model and defines the *modularity* Q of a given *partition* of a graph as

$$Q = \sum_i \left(e_{ii} - a_i^2\right)$$

where $a_i = \sum_j e_{ij}$ and i, j are indexes for graph communities [119]. The edges e_{ii} therefore connect nodes in the same community, while e_{ij} is the fraction of edges connecting two different communities i and j. In case the number of internal edges e_{ii} is not better than random, the modularity of the analyzed graph separation is low. High values of Q (approaching the maximum value of one) indicate a good partition in communities.

The modularity metric provides an intuitive understanding of the quality of the result of a particular community detection algorithm. However, there is more to it. By trying different community configurations (i.e., graph partitions) and evaluating their modularity, one can find the maximum modularity, and therefore the optimal partition. This family of community detection approaches is based on the idea of *optimization* of a certain metric, which was first shown for modularity by Newman and Girvan [120]. However, Brandes et al. found that this problem is NP-complete, and is not practically usable for large networks (e.g., the Internet) [30]. Therefore, approximate solutions for modularity optimization have been proposed [39, 119].

▷ *Louvain Method*

A particularly remarkable approximative solution is the "Louvain method", which is widely used for a multitude of different graph community detection applications [28]. A particular merit of the Louvain method is its ability to process huge networks – the authors report a runtime of 152 minutes for a graph with 118 million nodes. The algorithm starts by assigning each node to a separate community, and then continues with the following two main steps: first, for all neighbors i and j in the graph, the gain (in terms of Q) of putting i in j's community is evaluated. Node i is then placed in the community for which the highest gain was observed, or remains in its current community if no gain could be observed. This step is repeated until no further changes to the partition are being made, i.e., when a local modularity maximum has been found. Secondly, a new meta-graph is being created, where the previously identified communities are represented as nodes. The same procedure as before is repeated, so to identify communities of communities, until no further improvement of Q can be observed. The result is an approximate solution to finding the partition with maximum modularity.

While the Louvain method provides excellent approximations for the maximum modularity of many different graphs, it was found that the definition of modularity itself has a drawback. As modularity describes the quality of a partition of the entire graph, it is by definition a *global* metric. It has been shown that there exists a resolution limit, that has a particular negative impact for large graphs with many, differently sized communities [65]. Therefore, any modularity optimization approach has the tendency to merge small sub-communities, even though these should intuitively be reported as separate ones. Although several approaches try to "repair" this deficiency [138, 11], it was later shown that there is no general solution for this problem: increasing the resolution inevitably leads to the opposite problem of erroneously splitting larger communities [101]. In fact, the authors point out that the problem is not restricted to the definition of modularity alone, but probably affects all global quality metrics.

Another restriction of modularity optimization was discussed by Good et al. [73]. There exists a large number of local maxima for modularity, which are very close to the absolute maximum. Furthermore, the number of local maxima increases exponentially with the size of the graph. Therefore, it is relatively easy to find a graph partition that represents one of these, but it is in general hard to tell *which* is going to be found, given the intrinsic randomness of many community detection approaches. Two runs of the same algorithm, using the same input data and configuration, are therefore highly likely to produce different results, although each of them may individually be a high-quality approximation for the maximum modularity. Nevertheless, and despite these limitations, modularity optimization (and the Louvain method in particular) remains being used in many applications.

Lancichinetti et al. propose an alternative approach that is based on *local* optimization of a quality metric [103]. Their approach employs statistical features of communities to differentiate random fluctuations from real communities. The proposed system is called *Order Statistics Local Optimization Method* (OSLOM), and an implementation by the authors is available at `www.oslom.org`. OSLOM is applicable to directed and weighted graphs, and can reveal hierarchical community structure. However, the approach requires a significant number of iterations to produce high quality results, and is therefore not suited for very large graphs. The authors mention though that it still can be employed as a refinement step in these cases, e.g., after a first pass of the Louvain method.

Another alternative method was proposed by Rosvall and Bergstrom [141]. Their approach is based on the concept of relating graphs to geographical maps, and is accordingly called “Infomap”. The authors point out that good maps strike a balance between abstraction of minor details and conservation of important aspects. Likewise, a good (compressed) representation of a graph abstracts groups of densely connected nodes to communities. Infomap uses random walks on a graph, and assigns labels to the visited nodes. The choosing of these labels is based on information-theoretic concepts, and constitutes the main innovation of the approach: each community receives a separate label, but the individual node labels are being reused across the different communities. By virtue of this trick, the average description length for the node labels (i.e., the required number of bits) can be kept low. The best partition of a graph is then given when the description length of a random walk, constituting of a sequence of node labels, is minimal. In other words, Infomap reveals structures in which a random walk persists for longer times, until it eventually jumps to the next structure. This method has proven to be highly accurate for many applications [100]. However, as the original authors note themselves, it is not suited for all community detection scenarios. In particular, for finding community structures that consist of pairwise relations (as opposed to patterns of movement among nodes), other methods, like global modularity maximization, may provide better results.

A recent additional method is based on consensus clustering [102]. The authors base their idea on the observation that many community detection techniques involve a certain degree of randomness, and therefore produce different partitions with every run. The proposed approach implements a greedy strategy to reveal the median partition from a set of found graph partitions, i.e., the partition that is (on average) most similar to all others. While this system is not a standalone community detection approach, it addresses a fundamental drawback of most proposed techniques, and has shown to provide significant improvements.

4.2.3 Graph-based Malware Detection

Complex network theory naturally recommends itself for being employed for a variety of applications in the context of studying Internet traffic. The topological aspects of the relations of Internet entities complement the traditional analysis of point-to-point communication. Traditional network monitoring extracts information from single packets and flows, and matches these data against predefined models, reveals anomalies, and finds correlations. The notion of *structure* represents an orthogonal dimension. Collins discusses the application of graph analysis to network security monitoring, and provides an overview of the basic concept [41].

The basic idea is the detection of large groups of connected hosts, independently of the availability of concrete packet payload. In general, however, the sheer amount of information that can be extracted by network monitoring inhibits that all that information can be contained “as-is” in the graph. In order to enable efficient graph analysis, standard techniques limit either the *number of hosts* considered (i.e., the number of nodes in the graph) [159], or the monitoring *time window* (e.g., to a few minutes) per graph representation [105]. Both these approaches impair the scope of the analysis, though, and cannot be used for the analysis of rather slowly developing processes in large networks (e.g., P2P botnet communication). Alternatively, the analysis can be restricted to a subset of information that is known to be relevant a priori, e.g. by considering only certain *port numbers*, or flows with a particular packet inter-arrival time [87]. This has been shown to work well if a communication pattern, e.g. the usage of certain protocols and ports, is known. In the context of malware detection this is often not possible, as malware patterns are usually disguised as legitimate traffic, e.g. by using standard ports. Finally, in most cases it is in practice rather difficult to use any information from a packet's *payload* for filtering, due to exceeding cost of such deep packet inspection operations in large networks, and the increasing usage of encryption techniques. With respect to these techniques, malware is therefore often indistinguishable from legitimate traffic.

Nagaraja et al. present a system for detecting the topology of P2P botnets independent of exchanged data [116]. Their approach is based on the analysis of the connection graph structure, using a set of sophisticated methods involving clustering and random walks to identify densely connected subgraphs. Since no a priori distinction between legitimate and malicious connections is made, the system has to deal with the complexity problem of handling entire connection graphs and can benefit greatly from the graph reduction method proposed in this thesis.

Francois et al. propose a similar approach called "BotTrack" [68] in which they use a modified version of the PageRank algorithm to identify peer-to-peer structures. The system uses NetFlow records as input, and is thus immediately compatible with standard large network installations, which usually export this type of monitoring data. It processes exclusively the IP addresses it receives, and is in this regard closely related to the approach presented in this thesis. The main difference to our work lies in the fact that BotTrack analyzes the topological information only, without taking into account the individual differences between the graph's nodes (i.e., the hosts). This implies that the system has to process the *entire* connection graph, without any way to remove legitimate activity prior to the analysis. The authors evaluate their system using a large trace with synthetically injected "malicious" traffic, and report true positive rates between 38% and 97% depending on the injected botnet topology. The false positive rate varies between 1.9% and 6% for the different scenarios.

Coskun et al. propose a system for detecting neighbors in a P2P botnet, which they call "Friends of an Enemy" [43]. Their approach requires as additional input a *start node*, (i.e., the "enemy") which is known to be involved in malicious communication (e.g., a honeypot). The concept is based on the analysis of mutual contacts of the monitored hosts in a network. Two hosts have a mutual contact (and are therefore "friends"), if they contact the same destination IP. A *mutual contacts graph* contains then such nodes where the corresponding hosts are internal to the monitored network, and edges indicate the presence of a mutually contacted external host. Once a start node has been selected, their so-called "Dye-Pumping" algorithm distributes an initial confidence value ("dye") iteratively to adjacent nodes in the mutual contacts graph. Nodes with a certain dye level are suspected of being in the same P2P network as the start-bot. In order to control the graph complexity, the authors propose to remove a set of whitelisted nodes (e.g., those belonging to Google, Facebook, ...) and to remove destinations with large in-degree.

▷ *Dye-Pumping*

Fontugne et al. propose in [63] a method for comparing the output of different traffic analysis approaches in order to discover common results. They use a 15-minutes trace from the MAWI archive and analyze it using the sketch-based anomaly detection approach from [53] as well as with another approach based on image processing. From the analysis results, the authors construct a graph in which the nodes represent the specifically detected events (e.g. in time period t IP x sent data from source port p). The edges of the constructed graph correspond to detections by multiple analysis algorithms, i.e., when algorithms A and B both yield an event for the same traffic situation, the corresponding events are connected. The weight of these links depends on the overall similarity of events as detected by the individual algorithms. I.e., when an algorithm detects an event *iff* the other algorithms detect it too, the weight of the link between the events is set to the maximum value 1. The authors use a fast agglomerative community detection algorithm to identify those events in the graph that correspond to the same network activity. The algorithm maximizes modularity and is scalable enough to cope with huge graphs, but is prone to find local maxima only. The proposed system is interesting as it can deal with multiple granularities of monitoring data (e.g. packets, flows) and is able to analyze network activity from different vantage points. However, it remains unclear how the system performs when traces longer than 15 minutes are used.

Related to the detection of malicious traffic is the large field of traffic classification approaches, i.e., systems that attempt to derive the specific applications by analyzing traffic features. A particularly interesting approach is BLINC by Karagiannis et al. [92], which achieves highly accurate classification results even without using port numbers and payloads as sources of information. BLINC analyzes communications on the social (who talks to whom), the functional (client or server), and the application level (network flow characteristics). The latter considers so-called *graphlets* of communications, i.e., graphical visualizations of how a specific source IP address interacts with other IP addresses and which/how many ports are used in a specific time interval (here: 5 minutes). BLINC matches traffic patterns against a database of (manually derived) graphlets and thereby supports their classification. A set of thresholds controls the system and allows for manual adaptation to a specific network environment. A particular amenity of BLINC is that it is able to identify service farms, i.e., different IP addresses that provide the same service. The authors mention the potential applicability of BLINC for detecting malicious flows, but consider this mainly a possible additional use. Specifically, they differentiate malicious and legitimate flows only by exploiting highly visible peaks in some volume-based traffic statistics as well as by detection of perfect cliques, i.e., groups of hosts which all exclusively communicate with the exactly same set of other hosts (which botnets usually do not do).

Iliofotou et al. exploit the dynamicity in Traffic Dispersion Graphs (TDGs) to identify different application layer protocols in backbone traffic [87]. Their approach requires no specific information about used ports and is able to detect protocols that try to hide in other applications (*polymorphic blending* – e.g., a filesharing application that communicates via HTTP). The vertices in their TDGs correspond to the active IP addresses during a measurement window, while the edges between them represent interaction between the vertices. The authors use *edge filters* to define what is considered an interaction in a specific context (e.g., an edge could stand for the fact that at least three TCP packets on port 25 were exchanged). They employ traffic features from both single monitoring snapshots (*unary* features – e.g. the average degree) and, in addition, take feature changes over time into account. For the latter, they compare two TDGs and call the corresponding comparison values *binary* features (e.g., the presence of an edge in both graphs). Machine learning techniques are used to automatically find the dominant features that provide the best separation between the individual application layer protocols. The ground truth for the required training procedures is obtained by using a combination of signature-based and port-based filters with a set of (partially public) traffic traces. The actual classification distinguishes then between client-server and collaborative applications, and separates these two classes in a first step. For this, the authors find that two unary features (average degree and whether nodes have both incoming and outgoing connections) are sufficient for successful separation. Further separating the collaborative applications to find those using P2P protocols is harder and requires binary features (edge consistency and volatility) for sufficient accuracy. The final result of the presented approach is a manually derived classification tree, that separates collaborative and client-server protocols at the topmost level and then goes on to use other features for further separation in different application-layer protocols. To this end, a set of thresholds is used, which works equally for all employed traffic traces and correctly classifies 92% of the P2P traffic in the used traffic traces. Furthermore, the authors are successful in discovering Gnutella traffic in standard Web traffic (HTTP/HTTPS), using different pollution intensities and deviation thresholds. Although this work is focusing on traffic classification, it shares many of the challenges of graph-based malware detection. Similarly to our ideas, it employs structural traffic features to reveal the type of application layer protocol, irrespective of the payload and the actual ports being used. This reveals, e.g., P2P filesharing traffic injected

▷ *TDG*

in a dataset containing web traffic, even when both types of traffic use port 80.

Jin et al. discuss Traffic Activity Graphs (TAGs) in [91]. A TAG represents connections between Internet hosts (i.e., IP addresses), but is restricted to a pre-defined set of service ports (e.g., TCP ports 80 and 443 represent an HTTP TAG). The authors propose orthogonal nonnegative matrix tri-factorization (tNMF) for analyzing the resulting adjacency matrices, and revealing the core host interaction patterns. Similar to the community detection approaches discussed in the previous section, tNMF finds clusters of hosts in the adjacency matrix which are strongly connected. This work addresses the problem of the vast complexity of Internet connection graphs, and provides a procedure for finding the dominant characteristics. It thereby enables visualization and further analysis of the corresponding, network-wide communication patterns. The "focusing" procedure includes two steps, which both aim at reducing the graph complexity: first, one has to define the protocols one wants to investigate. Clearly, this has a significant impact, but may be harmful in the context of malware detection, where multiple ports could be used on purpose, so to conceal the criminal activities. Secondly, the tNMF procedure requires a set of parameters which control the granularity of the found clusters, and therefore needs to be tuned for each selection of service ports. These properties hinder the direct application of the system for malware detection, as typically no a priori knowledge about the specific network activity patterns can be assumed.

4.3 Evaluation of Malware Detection Results

As anticipated already in §3.2, a general problem for the evaluation of network-based malware detection approaches is the unavailability of ground truth for the analyzed network traffic data sets. Primarily the vast sizes of such data sets prohibit manual labeling of the contained traffic events. Furthermore, traffic data sets often contain privacy-sensitive information and are thus subject to non-disclosure agreements. It is therefore typically not possible to directly compare ones own results to the results of previous works, as the used data sets are not available. This fundamental problem is long known and there were attempts for deriving manually labeled, public data sets for enabling the comparison of analysis results [1]. However, the quality of such data sets has been questioned [149]. Furthermore, it strongly depends on the particular context whether an event constitutes an attack [110].

In practice, network traffic data analysis suffers from the additional problem that, in general, not all data can be collected which would be required for being able to reveal the ground truth. On the one hand, this is rooted in technological limitations as, e.g., packet loss of the network probes and limited storage capabilities. Likewise, encrypted traffic prevents the investigation of the packets' payload. On the other hand, data privacy legislation (see, e.g., [57]) forbids the broad collection of privacy-sensitive information.

However, note that collecting more traffic data typically supports revealing the ground truth, but is not always ultimately sufficient. For example, no part of a single packet which arrives at an Internet host which is under DDoS attack does necessarily differ from any other benign packet. The data which would be required to label such a packet would therefore needed to be collected at the origin, i.e., at the Internet host which sent the packet. It could consist, e.g., of the actual application which relates to this communication (e.g., an Internet browser vs. a DDoS tool). In the extreme case, this would mean that we would have to assess the intentions of the human operator of an Internet host for establishing ground truth.

Therefore, most proposed detection systems are evaluated using non-public data sets which include only a small subset of the full traffic information (e.g., in our case, only FQDNs and IP addresses). Typically used strategies for assessing the classification results include the following:

- *Usage of publicly available classification results* as, e.g., whitelists and blacklists [27, 9, 130, 26] In the context of malware detection, whitelists are often derived from online services which list highly popular web sites (e.g., `www.alexa.com`). The IP address ranges hosting these sites are often considered benign, and are therefore considered not relevant for the classification task. Conversely, blacklists are derived from web sites listed in Spam emails[8], from services which list Phishing sites[9], or from web site reputation systems[10]. They contain FQDNs or IP addresses which are considered malicious. By labeling ones own data set using these lists, once can establish (partial) ground truth.

  The main problem with this approach is that one can usually not infer *why* a specific site was white-/blacklisted. This is important, as the classification often results from a subjective impression of a specific analysis in a particular context. Indeed, for a fair comparison with the results of another detection approach, the very same activity should be present in one's own data set. E.g., in the context of detection of malicious DNS activity, it often makes a difference if a certain FQDN was observed one time or 10,000 times, depending, e.g., on the number of malware-infected hosts in the monitored network. Furthermore, any site could indeed represent malicious activity, but may still be out of scope for a particular detection system, which aims at detecting a very specific type of malicious activity. Whitelists and blacklists typically come without any context information and should therefore used only with their limitations in mind.

- *(Semi-)manual results verification* [130]. For some traffic analysis approaches, the number of true positives and false positives can be found by manually checking the classification results. This can be facilitated by looking up additional information for a limited number of results as, e.g., the packet payload of selected packets. Such information can then be compared to third-party resources, as, e.g., malware binary analysis reports. Typically, due to the large sizes of traffic data sets, the number of false negatives cannot be assessed, though. Furthermore, this type of analysis typically requires a human analyst who is an expert in the particular field.

- *Injection of known traffic patterns* [116, 43, 68], i.e., blending labeled traffic data in unlabeled data sets. This approach has the advantage that partial ground truth is available, and one knows exactly that the injected pattern should be reported as significant by the classifier. However, this technique ideally requires data sets which are entirely free of any malicious activity. Guaranteeing this is virtually impossible for large traffic data sets. Furthermore, the characteristics of the injected traffic patterns have to be carefully chosen in order to ascertain that these patterns indeed resemble realistic malicious activity.

All of these techniques have shortcomings which should be taken into account for results verification. Besides the focus on evaluating the classifier itself, one practical aspect of network-based malware detection approaches needs to be considered. As noted in §2, criminal Internet activities have certain communication requirements. For example, malicious services are often hosted on multiple IP addresses for redundancy. Not being able to fulfill these requirements has a significant impact on the utility of the malware infrastructures, and therefore on the criminals' revenue. The *parametrization* of the detection approaches influences the sensitivity for malicious activity and goes hand-in-hand with the evaluation of the classifier. Highly sensitive parameter settings limit the degrees of freedom

[8]E.g., `http://www.joewein.net`

[9]E.g., `http://www.phishtank.com`

[10]E.g., Google Safe Browsing

of malicious activity for going undetected and thereby constrain the utility of the malware platform. Consequently, and in line with [14] (cf. §3.2), we aim therefore at designing detection systems which yield low *absolute* numbers of false positives even for *highly sensitive configurations*. Furthermore, our approaches operate on graphs and therefore provide a *structured* representation of the relations between detected events, which supports further analyses.

Finally, note that the unavailability of ground truth represents a significant challenge for malware detection approaches based on (supervised) machine learning, and limits the applicability of these approaches in real-world deployments. This is discussed in great detail by Sommer and Paxson [145]. The detection approaches presented in this thesis take these limitations into account, and are designed to operate in absence of labeled training sets.

For evaluating our results, we employ a mix of all three strategies which we introduced above. Specifically, as we will further discuss in §9, we analyze our DNS detection results *manually* using publicly available blacklists and a variety of malware analysis sites on the Internet. Due to the discussed limitations, we consider fully automatic results evaluation potentially harmful (see §9.2 for an example), and therefore continue collecting evidence from multiple sources until a *expert human analyst* is convinced about the true nature of a specific event. Although this clearly entails more manual work and still does not completely match, in general, the actual ground truth, we consider this the best option we have given this difficult environment. Furthermore, we employ injection of known traffic patterns for the experiments in §13, for evaluating our system's ability to reveal the injected community patterns in a large connection graph.

4.4 Summary

We envision a malware detection system which scales to ISP networks and reveals patterns of unusually agile collaboration, which lead to the detection of service infrastructures used for Internet crime. The design of the system should not require the availability of pre-labeled data, which is often difficult to obtain. Furthermore, it should not be restricted to malware communication via certain protocols or assume the availability of clear-text traffic payload. An analysis procedure based on DNS analysis fulfills these requirements, and we can build upon an existing body of work (see §4.1.2) for detecting unusually agile DNS, and further contribute to the improvement of these techniques. On top of that, we aim at introducing graph-based collaboration analysis of agile DNS mappings, based on community detection algorithms. As we will show later, this can significantly improve the sensitivity of the analysis and the detection accuracy.

However, not all malware activity involves the usage of DNS. Most notably, C&C protocols based on P2P communication require a complementary analysis approach. Inspired by previous work (see §4.2.3), we contribute a novel approach for graph-based detection of such patterns of malicious collaboration. In particular, our work addresses the vast dimension of graphs representing Internet traffic, which severely impacts the applicability of costly analysis procedures.

The detection of collaboration patterns using graph analysis is a main goal of both our detection components. Despite many advances in the recent years, community detection in (large) graphs is still an open problem [64]. In particular, most existing approaches are not efficient enough to deal with the giant graphs we are facing in network monitoring, and/or are not reliable enough to find a large variety of differently sized communities without a priori knowledge. Especially for the purpose of detecting malicious sub-communities in Internet traffic, the problem of correctly classifying overlapping communities is relevant. Malicious Internet activity involves infected host machines (i.e., bots), which contact legitimate *and* malicious sites, and overlapping communities are therefore commonplace. Furthermore, it is not obvious what a graph community actually *is*, and what its distinctive features are which it make unambiguously clear that a group of vertices represents a community. Consequently, and based on the review of community detection algorithms, we design preprocessing techniques which aim at making the graph analysis problem as simple as possible, and thereby avoid performance limitations and results ambiguity.

The main complexity is therefore in the reliable detection of patterns of agile Internet activity, both in DNS and in the IP-level connections of Internet hosts. The better our system can understand which activity is normal by itself, the less data needs to enter the graph analysis stage. This reduction of the graph complexity is essential, as malware activity changes quickly, and we therefore require detection results in (close to) real time for being able to initiate countermeasures.

Malware activity often involves many different "professions" and "services" (see §2), which all may cause a different network traffic footprint. For example, a victim user might click on a link in a Spam email, which triggers an DNS request for a malicious site, from which the victim (unknowingly) downloads a malware binary. After installation, this binary causes some deviation in the network activity of the victim host, e.g., by resolving more malicious domain names, or initiating P2P communication. Ultimately, our goal is the joint analysis using both detection components, so to address the network communication of malware activity as a whole, which we discuss in §IV.

Part II
DNS Analysis

In comparison to other network-based malware detection approaches, the analysis of DNS traffic combines two main advantages which are highly relevant for deployments in large networks. On the one hand, DNS traffic contains relatively few private information (as, e.g., opposed to HTTP payload), and is rather low in volume. On the other hand, it allows for considering the entire host population for the analysis, instead of, e.g., sampling a subset thereof. As criminal activities in the Internet often involve the usage of DNS (see §4.1.2), the combination of these two properties enables a lightweight analysis with a broad coverage of monitored hosts.

In the following, a system for analyzing DNS data in real-time is being discussed, which allows for immediate detection of malicious (agile) activity. The system is highly sensitive, and can, in contrast to other approaches (as, e.g., [130]), detect also moderately agile malware activity with a very low number of false alarms. This is enabled by an in-depth analysis of *benign* DNS activity. We design a system which is able to understand the inherent DNS mapping agility per particular site, and thereby avoids large numbers of false positive reports caused by highly agile, benign services. Only *significant* violations of the previously derived DNS activity model for a particular IP range undergo a further analysis, in which we further exploit the fact that many malware activity shows signs of collaboration, which we detect using lightweight graph analysis. We evaluate the system using network traffic data from a large operator network, and discuss various interesting findings.

Individual parts of this work have appeared in the following publications:

- Andreas Berger and Eduard Natale. Assessing the real-world dynamics of DNS. In *Proceedings of the 4th international workshop on Traffic Monitoring and Analysis (TMA)*, pages 1–14, Vienna, Austria, 2012

- Andreas Berger and Wilfried N. Gansterer. Modeling DNS agility with DNSMap. In *Proceedings of IEEE INFOCOM Workshop on Traffic Monitoring and Analysis (TMA)*, pages 387–392, Turin, Italy, April 2013

- Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Antonio Pescapè. Detecting malware activity from agile DNS mappings using graph analysis. 2013. Submitted to IEEE Transactions on Dependable and Secure Computing

CHAPTER 5

Problem Definition

In principle, malware uses the DNS infrastructure in the identical way as legitimate clients and services, with the only difference that the contacted servers host a malicious service, i.e., a C&C server. From that alone, it is impossible to reliably classify such DNS activity as malicious, due to the lack of any clear *signal* that is indicative for malware. In other words, we simply cannot distinguish a malicious service from any other benign service if there is no difference in the DNS usage patterns. However, due to the improvements made with detection and mitigation strategies, malicious services were forced to introduce a certain degree of *agility* in their DNS usage [9]. Malware cannot use the same FQDNs and IP addresses over extended periods of time, as sooner or later these would be detected, and further be blocked or be taken offline. Malware services are therefore always "on the move", to escape countermeasures and therefore continue being profitable.

The most aggressive type of DNS mapping agility is *Fast-Flux*, which is typically used for enabling reliable, centralized C&C communication, and may involve hundreds of different IP addresses in many different networks, for hosting only one FQDN [85]. The basic functioning of Fast-Flux is illustrated in Fig. II.1. Malware-infected clients query the global DNS infrastructure for the FQDN of a C&C server. These queries resolve to a *changing set of IP addresses* of other infected clients (flux *agents*), which serve as proxies to one or more C&C servers. This agility in the DNS mappings can be observed using traffic monitoring.

However, not all malware services require the additional obfuscation layer introduced by flux agents, but have other operational constraints. As we pointed out in §3, many malicious services require reliable hosting of certain services, which are not necessarily related to C&C. These services often "flux" less, but use only a small set of IP addresses for redundant hosting, and *share* these IP addresses with other malicious services. For example, a Phishing campaign would typically try to "phish" using many different FQDNs, which resemble the names of the targeted sites (e.g., `gmai.com` instead of `gmail.com`, and `facebok.com` instead of `facebook.com`). Therefore, this activity does not stand out because of a large number of IP addresses being used over time, but rather due to the fact that an inhomogeneous set of new FQDNs are jointly using a set of IP addresses. Typically, these IP addresses are either new, or have been used for hosting different FQDNs previously. Therefore, the *signal* we are looking for when considering malware activity detection, is a certain degree of DNS mapping agility, which is caused by the fact that, for malicious services, the mappings between FQDNs and IP addresses are typically undergoing continuous changes, with some changing faster and some slower.

A fundamental feature being used by many malware detection approaches (see, e.g., [27, 82]) is the *number* of IP addresses being used per FQDN. However, many benign sites also use large numbers of IP addresses, for reasons of load balancing and hosting redundancy. The particular characteristics of a particular FQDN, as well as of an IP address, depends on the specific service. This is illustrated in Fig. II.2. Small sites do often reuse the same

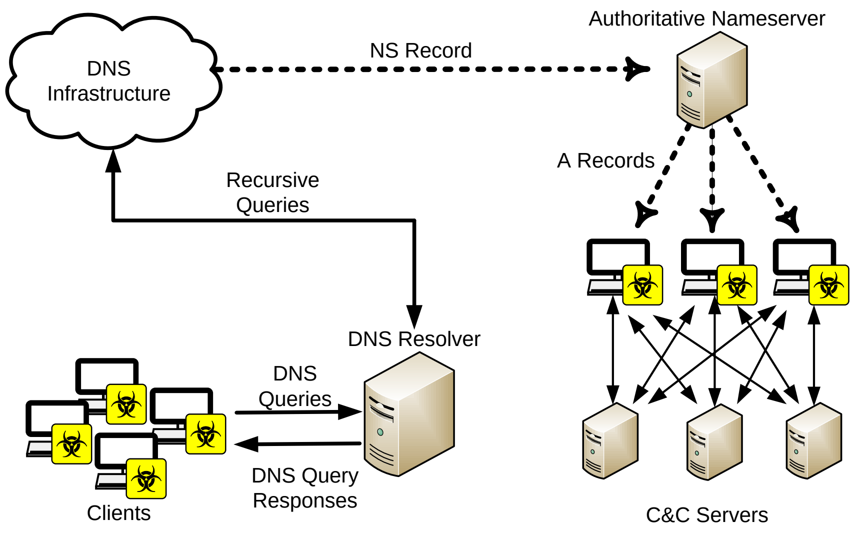
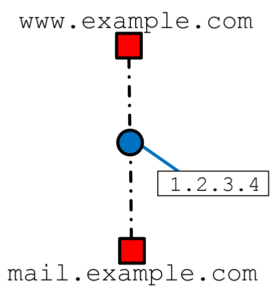


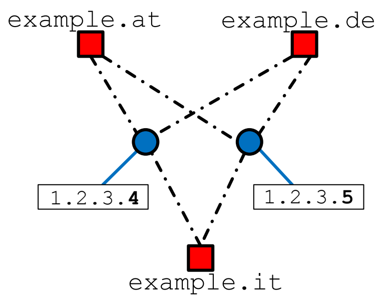
Figure II.1: Basic functionality of Fast-Flux. A set of malware-infected clients resolves FQDNs of their C&C infrastructure via DNS. The connections are proxied by a second set of malware-infected clients (flux agents), which quickly change over time, and hide the actual locations of the C&C servers.

IP address for multiple, low-volume services (e.g., the web and the mail server). Medium sites often operate multiple FQDNs (e.g., one per each country in which the company is active), and use multiple IP addresses, which often are in the same network range or even are neighbors in the IP space. In contrast, large sites typically operate multiple FQDNs for a variety of services (e.g., “example” and “sample”) and use large pools of IP addresses for hosting them. These IP addresses are often in different networks, and belong, e.g., to the Content Distribution Network (CDN) (e.g., Akamai) which was charged with hosting this site.

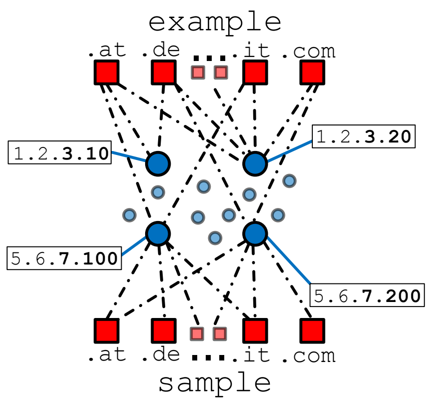
The hosting strategies of malware and benign services are therefore often highly similar. In fact, this is not surprising, as both address the same problem of guaranteeing high availability under (possibly) high load. However, the DNS activity of benign services is based on (long-term) contractual agreements, e.g., between a large service (like Facebook) and a CDN (like Akamai). Therefore, the DNS mapping agility of such sites may involve hundreds of IP addresses, in many different networks, as well as a large number of different FQDNs, but it is still constrained. Conversely, malware activity *needs* to change FQDNs and IP addresses often, and typically operates in different networks than large benign services. Therefore, we aim at *characterizing* the typical DNS usage per IP range, and identify the FQDNs and IP addresses which were either not seen at all before, or represent a significant change. By aggregating such changes over longer periods of time, we reveal groups of FQDNs and IP addresses which represent such changes, and therefore probably relate to malware activity. We represent such FQDNs and IP addresses as graphs and aim at revealing *collaboration* patterns which enable us to detect these malicious groups.



(a) Small site.



(b) Medium site.



(c) Large site.

Figure II.2: DNS usage of differently sized sites.

CHAPTER 6

How Agile are DNS Mappings?

In general, existing approaches strive to detect malicious activity by finding certain agility in DNS traffic, which can be differentiated from benign traffic. In [25], we investigate the same from the opposite direction: our aim is to find out to what degree *benign* DNS activity is *stable*. We conduct an extensive set of experiments on a DNS trace from a large operator network with several 100,000 customers. Over two weeks, we extracted and aggregated information from DNS NOERROR responses, i.e., successful DNS query responses that provided an IP resolution for the requested domain name and thereby a FQDN-to-IP mapping as described in §1.1. As a first step, we investigated the DNS activity for variety of sites manually, and report interesting findings for the individual results below, which underline the pronounced agility of benign services and demonstrate the differences between the individual requirements and deployment strategies.

Google Google's main search site is reachable under a number of FQDNs with the format `www.google.{SUFFIX}`, where *SUFFIX* can be *com* or almost any country code (e.g., *de*, *fr*, *it*, ...). However, all of these names are a CNAME alias of `www.l.google.com`[1]. This CNAME then finally resolves to a number of IP addresses, usually between one and six. In a two-hour time window, we find 71 different IP addresses being used, which are from 17 different /24 networks (resp. from five /16 networks). One day later, only 14 new IP addresses are used in addition. This indicates that it might be possible to automatically learn the set of IP addresses used for a service, given enough time. But Google search is not the only service hosted at these IP addresses. Within the same two hours, Google's image search site `images.l.google.com` use 31 of the *same* IP addresses, plus two different ones in addition. A malware detection approach must be able to cope with these "natural dynamics", so not to wrongly classify these services as malicious, just because a large pool of IP addresses is used for many different domain names.

Another interesting detail is that the site `www.youtube.com` (which is also owned by Google) maps to the CNAME `youtube-ui.l.google.com`, thereby following the same naming scheme as shown above. However, it resolves to 44 IP addresses that are not overlapping with the other IP pool identified before, and usually there are only between one and four IP addresses advertised per query. Still, *all* of the IP addresses belong to the AS "Google Inc.", i.e., Google seems not to host these services at some third-party hosting provider. In this case, knowing that a set of similar queries (i.e., `*.l.google.com`) is exclusively hosted in a single AS, could support DNS traffic classification.

Finally, a reverse lookup of the DNS name of any of the Google IP addresses always returns a DNS name in the format `{X}.1e100.net` (e.g., `fx-in-f147.1e100.net`).

[1]Sometimes, also `www{X}.l.google.com` is used, where *X* is in [2,3,4,5].

Apparently, Google implemented a common naming scheme for all their services, which might help to identify some site as belonging to this family of services[2].

Facebook Within two hours, we observed 54,691 different FQDNs with the format `{X}.{Y}.channel.facebook.com`, where *X* and *Y* are always numbers (e.g., `01371354742.67.channel.facebook.com`). However, they map to only 28 IP addresses in three different /24 networks. One day later, we observe 80,416 new domain names with the same format, still mapping to the same 28 addresses. Facebook seems here to "abuse" the DNS system for short-lived, extremely dynamic service identification, using some form of wildcard FQDN-to-IP mapping (i.e., `*.channel.facebook.com`). While it would be easy to whitelist this particular phenomenon, a universal detection approach for malicious domains must be designed such that these short-lived domains are recognized as normal, benign activity.

Amazon Amazon shows a two-fold face: on the one hand, their main site `www.amazon.com` maps to only three different IP addresses in two hours, all from their own AS "Amazon.com, Inc.". In strong contrast to Google, their European branches (*.de*, *.fr*, *.it*, ...) map to a small set of IP addresses in the AS "Amazon EU DC AS". This activity is therefore highly stable. However, for Amazon's cloud services, the observed activity is completely different. Within two hours, we observe 5,830 FQDNs of the format `{X}.profile.{Y}.cloudfront.net`[3] on 3,903 different IP addresses. Each name usually maps to 8 different IP addresses, all of them belonging to either AS "Amazon.com Tech Telecom", "Amazon EU DC AS", or "Amazon.com, Inc.".

Akamai Akamai operates one of the largest CDNs, therefore a large service diversity is to be expected. Within two hours, we observe 3,168 CNAME aliases with the format `a{X}.{Y}.akamai.net`, where *X* is a number (roughly in [1, 2000]), and *Y* is a short string (e.g., `a1254.w7.akamai.net`). In total, these CNAMEs map to 1,729 different IP addresses. Additionally we find 20 CNAMEs with the similar format `a{X}.{Y}.akamai.net.0.1.cn.akamaitech.net`. While almost all of these IP addresses belong to the AS "Akamai Technologies European AS", we find 31 other ASes of backbone operators and Internet providers. Simply counting the number of different ASes to which a domain name is associated with over time and expecting to find just malicious activity over a certain threshold, is therefore misleading.

In addition, for understanding the complications of these configurations, consider the following example: The domain name `js2.wlxrs.com` was found to be hosted in the AS "Akamai Technologies European AS", specifically on a set of servers that follow the naming scheme introduced above. It is a CNAME alias of `login.live.com`, the login site of Microsoft's *Live* service. Within the same two hour time window however, this site maps much more often to the CNAME alias `login.live.com.nsatc.net`, and points to an IP in AS "Microsoft Corp". In summary, we have to deal here with a popular service that maps to multiple CNAME aliases and points to different IP addresses in different networks. This activity is prone to be misclassified as Fast-Flux, and poses a significant challenge to malware detection approaches.

No-IP.org As a final example, we consider the popular dynamic DNS provider No-IP.org, which operates authoritative nameservers for a number of domains. Sub-domains of these can be registered by anybody at no cost. Within two hours, we observe 109 FQDNs of the format `{X}.no-ip.org` that resolve to 74 unique IP addresses in 52 different ASes. This

[2] 1e100, or $1 \cdot 10^{100}$, is called a "Googol" by mathematicians.

[3] E.g., `a5e0129bd6b7bd4ef1e4f83b979a1216e.profile.dub2.cloudfront.net`

corresponds to 69 different /24 networks, and resp. 61 different /16 networks. Clearly, such domains can be easily taken for malicious ones when looking only at their distribution over the networked world. Given that such free, disposable domains names seem like a perfect way for a miscreant to host a C&C server, malware detection approaches need to find a way to distinguish such legitimate agile DNS usage from malicious one.

6.1 Methodology

Many of the given examples exhibit different agile DNS features at the first glance, although they represent highly stable, benign services. Thus, DNS stability means different things for different services. In the following, we define a flexible metric to qualitatively assess the degree of stability of a particular service. We use the following notation: $\mathcal{Q}$ and $\mathcal{P}$ are the sets of domain names and IP addresses, respectively. We define $\mathcal{M}_{\mathcal{Q}}$ as the set of tuples $\langle q \in \mathcal{Q}, \mathcal{P}^* \rangle$, where $\mathcal{P}^* \subseteq \mathcal{P}$ is defined by $\{p \in \mathcal{P}\ s.t.\ q\ resolves\ to\ p\}$. Similarly, $\mathcal{M}_{\mathcal{P}}$ is the set of tuples $\langle \mathcal{Q}^*, p \in \mathcal{P} \rangle$, where $\mathcal{Q}^* \subseteq \mathcal{Q}$ is defined by $\{q \in \mathcal{Q}\ s.t.\ q\ resolves\ to\ p\}$.

The simplest form of a *stable, unidirectional mapping* is then given when for a tuple $\langle q, \mathcal{P}^* \rangle$ there exists only a single element in $\mathcal{P}^*$ (and analogously for tuples $\langle \mathcal{Q}^*, p \rangle$). Therefore, a *stable, bidirectional mapping* is given when any q maps to a single p, and p exclusively maps to this q. For simplicity, we refer to this as ***1:1*** stability in the following. The set $\mathcal{S}$ of 1:1 stable mappings is thus defined as $\mathcal{S} := \mathcal{M}^s_{\mathcal{Q}} \cap \mathcal{M}^s_{\mathcal{P}}$, where $\mathcal{M}^s_{\mathcal{Q}} \subseteq \mathcal{M}_{\mathcal{Q}} := \{\langle \{q\}, \mathcal{P}^* \rangle\ \ s.t.\ \ |\mathcal{P}^*| = 1\}$ and $\mathcal{M}^s_{\mathcal{P}} \subseteq \mathcal{M}_{\mathcal{P}} := \{\langle \mathcal{Q}^*, \{p\} \rangle\ \ s.t.\ \ |\mathcal{Q}^*| = 1\}$. Conversely, two FQDNs which map to the same IP address are *colliding*, and the corresponding DNS mappings are therefore not stable.

For clarity, consider the following real-world example of a 1:1 mapping: the FQDN `www.ftw.at` points *always* to the IP address `213.235.244.145`, and no other FQDN than `www.ftw.at` *ever* points to `213.235.244.145`. These types of mappings are extremely common, and should clearly never be reported by a malware detection approach. Would another FQDN also map to `213.235.244.145`, then it would collide with `www.ftw.at`, and *both* mappings would not be 1:1 stable.

In order to address more complex types of mapping stability, we define the following relaxations: $k - LD(query)$ specifies the k-level domain name of *query*, e.g., 2-LD(`www.ftw.at`) is `ftw.at`. Similarly, $j - NW(IP)$ gives the $/j$-network of an IP address, e.g., 24-NW(`213.235.244.145`) is `213.235.244.0/24`. This is motivated by the fact that both the name and the address can vary at different degrees, while still representing the same service. I.e., large Internet sites assign a network segment to single services instead of a single IP and, on the other hand, often more than one name points to a single IP address. And as we can see clearly for, e.g., the various Google services, both name and address vary often at the same time. Still, they do not vary arbitrarily: multiple IP addresses for a service are often in the same network segment, and multiple FQDNs frequently share a common suffix.

The set of stable mappings $\mathcal{S}$ is derived analogously as shown above. The names and addresses in the input sets $\mathcal{Q}$ and $\mathcal{P}$ are now the results of applying k-LD(query) and j-NW(IP). For simplicity, we refer to $k - LD : j - NW$ mappings as $k : j$, e.g., a 2-LD:24-NW mapping is abbreviated as 2:24. As an additional convention, and in line with the 1:1 mapping defined above, we use *1* for describing the case when we do not apply k-LD or j-NW at all[4]. I.e., a 1:24 mapping describes a single domain name that is exclusively hosted by an /24 network. Conversely, a 3:1 mapping describes all services that share the same 3-LD suffix and are exclusively hosted at the same IP address. Also note that 1:32, ∞:1, and ∞:32 are equivalent to 1:1.

[4]For practical reasons, it makes little sense to consider 1-LD (i.e., TLD) names or /1 networks (i.e., half the Internet) anyhow, so we accept this contradiction here.

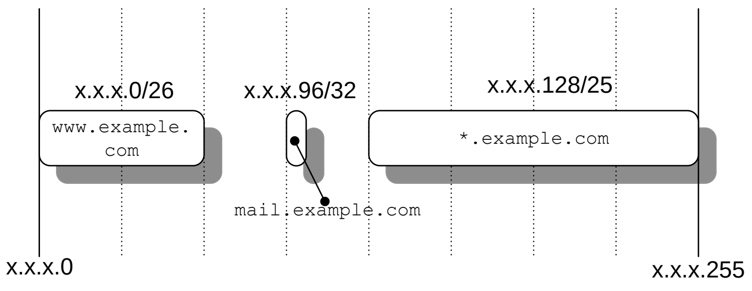


Figure II.3: Illustration of the possible usage of the /24 network from `x.x.x.0` to `x.x.x.255`.

For illustration, consider Fig. II.3. Given the shown IP address usage, we would find three different stable mappings in this range: `mail.example.com` is 1:1 stable, `www.example.com` is 1:26 stable, and all FQDNs ending on `example.com` are 1:25 stable.

The introduced bidirectional definitions are strong in the sense that they allow for only little variation of a domain's name and the IP addresses hosting them. Conversely, everything that is considered stable according to any of these definitions exhibits extremely little agility, and does therefore not match the malware model. However, as our analysis of benign sites above shows, many of them are not considered stable according to these definitions. Many of these sites share a common reason for that, namely the fact that they use a set of IP addresses from *different* networks for one or more domain names. Identifying a network mask that matches all these mappings *but no others* is therefore in general impossible. However, we can exploit the stability of the geographical position of these IP addresses, as many different ranges of a single organization are often registered at the same location. We define this additional type of stability as $1 \rightarrow GEO$ stability, i.e., a domain name is considered stable when it always resolves to an IP address at the same longitude/latitude.

6.1.1 Implementation

The basic idea is to find the set of stable mappings in a DNS trace by testing each mapping against each definition of stability as defined in the previous section. As the processing of the trace proceeds, new stable mappings are added and those contradicting the particular definition of stability are removed. Once a k-LD name or a j-NW network has been removed from the stable set, no other mapping that contains either of them can be considered stable. We implement this by using two Bloom Filters [29] for efficiently storing the already removed domain names (*bfd*) and addresses (*bfa*). The entire procedure is shown in Algorithm 6.1. Note that `BloomFilter.add(x)` also removes *x* from the set of stable mappings. `BloomFilter.rel_add(x)` additionally adds all *related* entries, i.e., those mappings that were stored previously. For example, if *stables* contains the mapping `www.example.com:1.2.3.4` and a DNS record with `www.example.com:5.6.7.8` is observed, then `www.example.com`, `1.2.3.4`, and `5.6.7.8` are removed from *stables* and are added to the Bloom Filters.

As Bloom Filters are probabilistic data structures, false positives may be reported. That is, the filter wrongly reports a certain element to be stored, while it is not. The false positive rate is a function of the filter size, the number of hash functions used, and the number of inserted elements. Knowing the total number of unique FQDNs in our trace (1,444,735), and therefore the maximum required filter size, we set the number of hash functions for *bfd* to 34 to guarantee a maximum false positive rate of $\leq 0.01\%$. Using again 34 hash functions, we set the filter size of *bfa* to 700,000 to achieve the same false positive rate. During the experiments, the maximum number of IP addresses stored in one instance of *bfa*

Algorithm 6.1: Find-k:j-mappings

```
input  : DNS-Records, k, j
output: stables, bfd, bfa
stables ← Map(); bfd ← BloomFilter(); bfa ← BloomFilter();
foreach record r in DNS-Records do
    dname ← k-LD(r.name, k);
    addrs ← Get_IP addresses(r.A-Records);
    if dname in bfd then // this domain name is known to be not stable
        bfa.rel_add(addrs);
        continue;
    end
    foreach a in addrs do
        ip ← j-NW(a.ip, j);
        if ip in bfa then // this IP address is know to be not stable
            bfa.add(addrs);
            bfd.rel_add(dname);
            break;
        end
        if dname not in stables and ip not in stables then // a new stable mapping
            stables.add(dname,ip);
            continue;
        end
        if stables[dname]=ip and stables[ip]=dname then // we saw exactly this mapping before
            continue;
        end
        if stables[dname]≠ ip then // known FQDN, used other IP before
            bfa.add(addrs);
            bfd.rel_add(dname);
            break;
        end
        if stables[ip]≠ dname then // known IP, used by other FQDN before
            bfd.add(dname);
            bfa.rel_add(addrs);
            break;
        end
    end
end
```

was 313,863 (for experiment 2:1). The filters are stored after each run of the experiment, so that we can stack them together in different ways for quickly getting different views on the data, that depend on the order of the individual experiments. However, note that due to the non-zero false positive rate we occasionally misclassify single mappings.

For the 1→GEO experiment we slightly modified Algorithm 6.1 to consider ⟨longitude, latitude⟩ tuples instead of IP addresses. Note that we do not check for collisions of geographical locations, i.e., arbitrarly many FQDNs can map to the same location for being considered stable. However, any of these FQDNs is considered not stable if it maps to any other location in addition. We used MaxMind's free version of their GeoIP City database[5] for retrieving the geographical location of IP addresses.

[5]http://www.maxmind.com/app/city

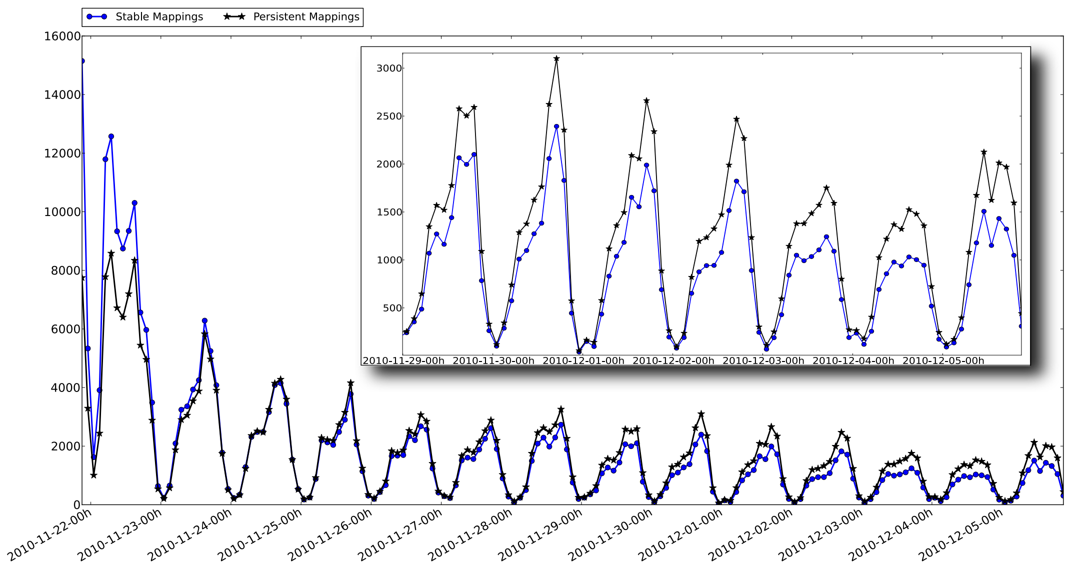


Figure II.4: 1:1 stable mappings: changes over time.

6.2 Experimental Evaluation

We analyzed the aforementioned DNS trace which was created by capturing and parsing all packets on UDP port 53 in a large European Internet provider's network. The trace includes all DNS activity from 2010-11-21, 21h to 2010-12-5, 21h (i.e., 14 days). We extract from these data the DNS "NOERROR" query responses and aggregate them on two-hour time bins using the queried name as a key. That is, for every two hours we output a list of queried FQDNs together with the number of queries they received and the list of IP addresses in the responses' A-Records. During processing, we then remove those records that received less than three queries. Furthermore, we remove the prefix `www.` from all queries, and therefore treat `www.X.com` the same as `X.com`[6]. In total, taking into account these measures, we observe 1,444,872 unique queried names which are hosted in 21,968 different autonomous systems and for which we received 1,025,858,834 individual DNS queries.

6.2.1 1:1 Stability

We found 310,788 stable 1:1 mappings in our trace, which represent 21% of the unique FQDNs in the entire trace. In other words, more than one fifth of the sites host only a single service on a single IP address. Most likely, these are small companies or private websites, that do neither require redundant hosting nor multiple services on the same IP. However, this set is constantly evolving. Figure II.4 shows the changes of the number of stable mappings over time. For better visibility, we also provide a zoom-in on the tail of the results (from 2010-11-29 to the end of the trace). After the first day of analysis, the number of newly found stable mappings decreases as popular 1:1 stable sites are already known by then. However, it is interesting to note that even at the end of the trace, i.e., after almost two weeks, the set of stable mappings grows at least by a few hundred in every time bin. Note that this set could theoretically also shrink, given there would be more mappings removed than new ones are introduced, which is not the case. In other words, there are always more entirely new mappings, than new FQDN or address conflicts (due to reuse) being found.

Of course, it is possible that a new mapping is considered stable in one time bin, but gets removed again in a subsequent one. Therefore, we assessed the number of *persistent*

[6]Many sites use the same mapping for both, as users often do not use the `www.`-prefix.

j	$\|stables\|$	$\|bfd\|$	$\|bfa\|$	Δ	**j**	$\|stables\|$	$\|bfd\|$	$\|bfa\|$	Δ
31	263,945	1,180,927	205,031	715	**27**	123,756	1,321,116	144,004	182
30	227,344	1,217,528	189,549	496	**26**	96,629	1,348,243	127,371	112
29	190,168	1,254,704	176,142	356	**25**	73,996	1,370,876	110,793	58
28	155,741	1,289,131	160,276	295	**24**	55,689	1,389,183	94,504	45

Table 1: 1:j results.

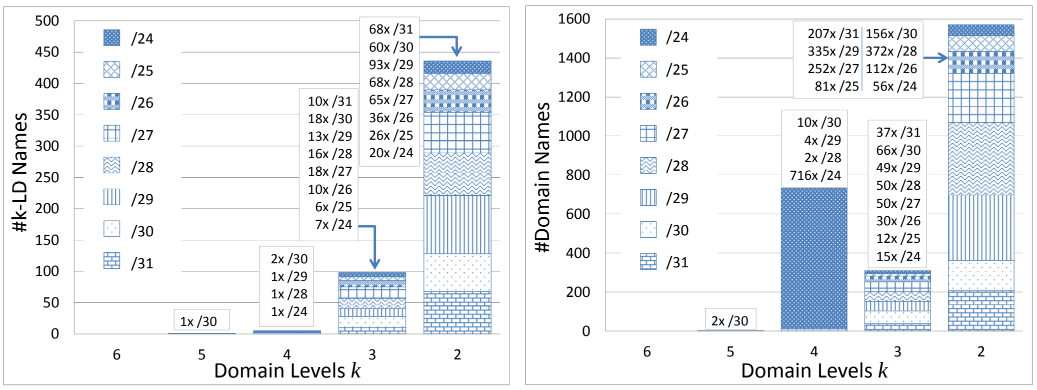
mappings at each time bin t, i.e., those that are first seen at time t, and are still considered 1:1 stable at the end of the experiment, after two weeks of traffic data. Interestingly, the number of new, persistent mappings decreases rather slowly. This suggests that a system for detecting malicious domains should be designed such that it can adapt to these dynamics, i.e., purge historic, inactive mappings to avoid future FQDN-IP conflicts, and learn at the same time about new ones. For many malware detection approaches, e.g., those based on machine learning, this implies that continuous retraining should be considered.

6.2.2 k:j Stability

After having identified the set of 1:1 stable mappings, we continue by gradually relaxing our definition of stability. We conduct three separate experiments: first we analyze the special cases 1:j and k:1. Then, we investigate several other combinations of k:j mappings (i.e., with k,j $\neq$ 1).

1:j This experiment targets FQDNs that *exclusively* use a set of IP addresses instead of a single one. We define the set of netmasks as $\mathcal{J} = \{31, 30, 29, 28, 27, 26, 25, 24\}$, and run one experiment for each $j \in \mathcal{J}$. We start with the largest value (31) and continue with 30, 29, etc. By addressing more and more addresses, our definition of stability gradually "widens". Note that, as soon as it gets too wide so that other services "pollute" the IP range, 1:j stability is not given anymore. Table 1 shows the results, adopting the terminology from Algorithm 6.1. The number of stable mappings decreases from one experiment to the next, due to overlaps in network ranges from different services. Conversely, the number of elements in *bfd* increases steadily, due to more and more address conflicts. The elements stored in *bfa* are in this case /j networks, hence the number of blocked IP addresses also steadily increases. In addition, we compute the number of new stable mappings Δ from one step to the next. As a first reference we use the results from the 1:1 experiment, and find that 1:31 matched 715 new domain names. 1:30 finds then 496 as compared to 1:1$\cup$1:31 etc. Note that the found domain names use the individual IP ranges *exclusively*, which represent in total 42,558 IP addresses. Many of them are larger services like `europe.battle.net`, an online gaming site, which is found to be 1:24 stable.

k:1 Similarly, we define the set of domain levels to analyze as $\mathcal{K} = \{6, 5, 4, 3, 2\}$, so to target FQDNs that share are common suffix but are all *exclusively* hosted at the same IP, and run one experiment for each $k \in \mathcal{K}$. We report in the following the number of new stable mappings Δ from experiment to experiment. As before, we use the set of 1:1 mappings as a first reference, and compute the number of changes in the order of decreasing k. The results are $\{3, 6, 296, 563, 2, 526\}$, i.e., in total we find 3,394 k:1 stable mappings. Note that the number of mappings is not equal the number of matched domains here, as k-LD represents a not-unique domain pattern. As it is considered new compared to 1:1, there must at least be two FQDNs matching each. The total FQDNs matched per $k - LD$ pattern are $\{9, 19, 10, 267, 3, 386, 8, 257\}$. The high count of 10,267 domains is almost exclusively



(a) k-LD names count (b) Domain names count

Figure II.5: k:j mappings: number of additional stable mappings per k,j configuration.

due to a large number of `{X}.webim{Y}.webim.myspace.com`. Furthermore, we find 643 FQDNs `{X}.jim{Y}.mail.ru` that are 3:1 stable.

k:j Finally, we run a set of 40 experiments using all combinations for $k \in \mathcal{K}, j \in \mathcal{J}$. The individual number of changes is shown in Figure II.5. On the right, the number of new k-LD patterns is shown (e.g, $\star$`.example.com`), while on the left we show the number of FQDNs matching the individual patterns. We compute the number of changes in a similar way as before, using as a first reference the set of stable mappings found by all previous experiments. Following our idea of gradually widening the definition of stability, we proceed in the following order: 6:31$\rightarrow$ 6:30 $\rightarrow \ldots \rightarrow$ 5:31 $\rightarrow \ldots$2:24. In general, the set of found domains is rather diverse, with no single, dominant service. The largest share of domains is in 4:24 and is exclusively caused by 716 FQDNs with the format $\star$`.profile.sin2.cloudfront.net`.

6.2.3 1$\rightarrow$GEO Stability

In total, we find 1,391,251 1$\rightarrow$GEO mappings, i.e., 96% of the unique mappings in our trace. Out of that, 1,053,657 were not already found by the k:j experiments. Among those are 291,977 unique $\star$`.channel.facebook.com` and 9,402 unique $\star$`.cloudfront.net`. The only two similar domain names *not* in this set are `0.53.channel.facebook.com`, which appear in two different ASes (US, Ireland), and the 716 domains $\star$`.profile.sin2.cloudfront.net` which were already found previously by the 4:24 experiment.

It is quite surprising that 1$\rightarrow$GEO stability is given for such a large number of mappings. However, this is clearly a rather coarse measure compared to the other definitions of stability. In particular, note that 1$\rightarrow$GEO stability is not sensitive to IP address reuse, i.e., assigning multiple domain names to the same IP address. In order to get an intuition of how many legitimate sites are among these mappings, we tried to resolve their domain names again. Given that since the recording of the DNS trace almost one year passed, we reasoned that any domain name that would still be resolvable to an IP address, is most likely legitimate[7]. We randomly selected 105,365 (i.e., 10%) FQDNs from the set of mappings that were exclusively found by 1$\rightarrow$ GEO and found that 103,464 still resolved to an IP address.

[7]Note that due to DNS domain parking we might wrongly assume that a site is still active, as we would be redirected to a default site. The majority of sites does not implement this though, so we consider this bias acceptable.

1,901 (i.e., 1.8%) domains were not resolvable anymore. We conclude that the majority of 1→GEO domains is therefore legitimate.

Finally, we built a set of stable mappings from the individual results of all experiments which contains 1,391,256 entries, i.e., 96% of all unique queried FQDNs are considered stable. Interestingly, these correspond to only 492,730,776 out of the 1,025,858,834 individual queries in our trace, i.e., 48%.

6.3 Discussion

The experiments show that a significant share of 21% of all mappings are 1:1 stable. In addition, only ∼1% meet any of our more relaxed k:j definitions of stability. This is surprising, as we would have assumed that more network ranges are exclusively used by a certain domain suffix, e.g., those by small and medium enterprises who would use a certain portion of a larger network. The most interesting point of learning is therefore that this is not the case, and that network masks are no reliable units of separation of Internet services. This is mostly due to the widespread use of CDNs and professional hosting providers. In contrast to traditional hosting strategies, such operators *reuse* IP addresses for multiple services, and thereby cause FQDN collisions. Furthermore, even small sites using such services employ redundant hosting, and therefore the associated FQDNs can map to many different IP addresses in multiple networks.

However, 96% of all FQDNs in our data set were found to be 1→GEO stable. This definition of DNS mapping stability is more loose and does not require the absence of FQDN collisions. Therefore, it provides a simple way for discovering services that are not entirely stable, but are at least not hosted in multiple, geographically distributed networks. We consider 1→GEO stability a lower bound for stability of particular DNS mappings, which demonstrates that, despite the widespread usage of CDNs, the vast of majority of DNS mappings show *some* stability. This does not go without saying, as many services are, e.g., hosted by Akamai, who use a highly distributed set of servers, many of which are located directly in the Internet service providers' networks [122]. In these cases, the DNS resolution of a particular service depends on the current load of the network and the Akamai servers, and is therefore expected to fluctuate over time. The fact that these fluctuations do not occur for the majority of DNS mappings in our data set is promising, as malware services are expected to employ (and require!) significantly more agile mappings. As we are not able to observe FQDN collisions though, we need to further improve our definitions of DNS mapping stability, in order to retrieve a tighter bound than 1→GEO can offer. This would enable us to detect IP reuse of changing sets of malicious FQDNs, as it occurs, e.g., for Phishing campaigns or botnet C&C.

The 4% of remaining sites which were not caught by any of our definitions of stability use highly agile DNS mappings and attract the lion's share of the traffic. We found 53,616 such FQDNs in 3,837 ASes. Among those, the two most popular sites `www.google.com` and `www.facebook.com` alone received 45,941,330 and 23,813,897 queries, respectively. The top-{10,30,50,100} FQDNs received {154,129,864, 224,851,245, 320,273,845, 373,019,662} queries. The top-10 ASes hosting these 53,616 domains were Microsoft Corp, Peer 1 Network Inc., Layered Technologies, Inc. (two separate ASes), Akamai Technologies European AS, 1&1 Internet AG, Internet Systems Consortium, Inc., Georgia Institute of Technology, MX Logic, Inc., and Deutsche Telekom AG. In total, the top-10 ASes hosted one or more IP addresses of 54% of the found domains, which attracted 57% of all individual queries. As mentioned previously, this is indeed not surprising, as it is well known that large services (e.g., Google, Facebook, Microsoft) as well as CDNs (e.g., Peer 1, Layered Technologies, Akamai) operate data centers at multiple geographical locations, and dynamically assign IP addresses depending, e.g., on the current load and the communication latency. The appearance of "Georgia Institute of Technology" among these other, very large networks stands out though. By manually inspecting the sites hosted there, we found 2,565 random-looking FQDNs. Apparently they belong to a botnet that was caught by Georgia Tech's honeynet research project[8], or was sinkholed there. This serves as a first confirmation that DNS analysis can indeed reveal malware activity, although in this case it has been already detected and contained.

Finally, an important point of learning from our experiments is the high degree of dynamics in Internet DNS traffic. Due to the multitude of different and evolving services, the DNS is constantly changing. New FQDNs and new IP addresses appear and disappear on a daily basis. The specifics of this activity depend on the type of operator that hosts a given service. A direct implication for a detection system based on modeling this activity is therefore the requirement to regularly flush information derived from old activity, else the model would represent a considerable amount of outdated mappings[9]. Equally important, yet less obvious, are two phenomena to which we refer to as *limited visibility* and *DNS wildcards*, and which we explain in the following:

Limited Visibility The popular blogging platform Tumblr assigns to its users domain names following the pattern `{account-name}.tumblr.com`. Despite the fact that there are millions of accounts, of course not all of them are frequently queried from within the monitored network, and therefore we simply rarely *see* them. Instead of storing a continuously changing list of Tumblr domains mapping to Tumblr's IP addresses, and continuously reporting changes, we rather want to extract the more useful information that `<something>.tumblr.com` is using this IP range.

Wildcards The case of Facebook demonstrated that benign Internet services sometimes use random prefixes for FQDNs (e.g., `{X}.{Y}.channel.facebook.com`). Clearly there is no value in keeping track of all these domains, but rather we want to understand the pattern which these domain names follow. Ideally, we would like to derive automatically the wildcard DNS mapping (i.e., `*.channel.facebook.com`) that Facebook most likely uses for these IP addresses, and *not* report such FQDNs as suspicious, agile activity.

These learnings serve as the basis of the modeling approach for DNS mappings described in the next chapter, which ultimately enables us to build the malware detection system discussed in §9.

[8]`http://users.ece.gatech.edu/owen/Research/HoneyNet/HoneyNet_home.htm`

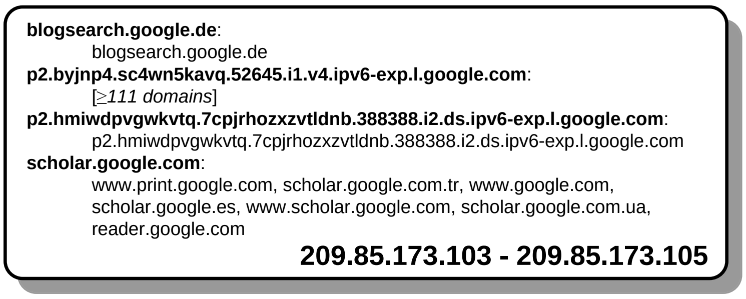
[9]See also Hao et al. [80], who found 5,711,602 new domains in only two months, although their investigation was restricted to the TLDs `.com` and `.net`.

CHAPTER 7

The DNSMap Approach

Based on the insights presented in the previous discussion, we propose *DNSMap*, a system which automatically extracts the *dominant* patterns of domain names for an IP range from DNS query responses captured from the wire, and takes the problems of limited visibility and DNS wildcards into account [22]. Furthermore, DNSMap merges neighboring IP addresses when they seem to host similar services, in order to provide an aggregated view on which family of services uses which range of IP addresses in a network. It can therefore represent *arbitrary* IP ranges and is not restricted to ranges which can be represented by netmasks. Therefore, DNSMap can be used for "zooming-in" on a particular service in a larger network.

On top of that, we present an approach that allows for detecting *significant* changes in the global DNS zones, i.e., such mappings that differ significantly from the previous ones according to a divergence metric we define. Fig. II.6 shows an example of the information our system derives. In this case (created from a real-world data set), the system's output conveys the information that on three subsequent IP addresses a set of related domain names is hosted. These domains cluster in four groups, for which the system derives group *labels*. Would we now observe a new domain name mapping to this IP range, that does not "fit" to any of the identified groups, the system would report this as an unusual event. On the other hand, domain names that differ only slightly from the ones seen before, are absorbed by DNSMap, and are not reported.



blogsearch.google.de:
    blogsearch.google.de
p2.byjnp4.sc4wn5kavq.52645.i1.v4.ipv6-exp.l.google.com:
    [*≥111 domains*]
p2.hmiwdpvgwkvtq.7cpjrhozxzvtldnb.388388.i2.ds.ipv6-exp.l.google.com:
    p2.hmiwdpvgwkvtq.7cpjrhozxzvtldnb.388388.i2.ds.ipv6-exp.l.google.com
scholar.google.com:
    www.print.google.com, scholar.google.com.tr, www.google.com, scholar.google.es, www.scholar.google.com, scholar.google.com.ua, reader.google.com

209.85.173.103 - 209.85.173.105

Figure II.6: Example for an "IPBlock", DNSMap's basic representation of DNS mappings. Shown are the domain names that mapped to this range of IP addresses, grouped by domain name similarity. The system derives a label for each group, shown in bold letters.

7.1 Methodology

As we have shown in §6, the level of DNS mapping agility of benign Internet services varies widely. Websites of small companies are hosted on the same IP addresses for years, while large enterprises host their services on many different addresses. Both of them typically use sets of FQDNs which are somewhat similar to each other, following, e.g., the pattern `*.example.com`. In contrast, CDNs and hosting providers reuse the same IP address for a large variety of different sites, while in access networks sometimes dynamic (DHCP) addresses host, e.g., private websites using dynamic DNS providers. The level and the type of DNS agility therefore depends on the specific site (i.e., FQDN) and the IP address we are looking at, and is usually limited. Even large companies do only use IP addresses in a certain set of networks, though this set may be large. Private customers may use many different IP addresses due to DHCP, but these typically all belong to the same network. Malicious sites have different constraints: they need to change both FQDNs and IP addresses often, so to avoid mitigation actions and react to new "business" requirements (e.g., a new Phishing campaign which requires a different FQDN).

DNSMap's main objective consists of characterizing this agility. Instead of just counting, e.g., the number of IP addresses for a given FQDN, we infer the real "unusualness" of these DNS mappings from evaluating how normal *this* FQDN appears for a *specific* IP address. In the following, we derive a system that scales to large volumes of DNS data and enables almost instantaneous detection of highly agile activity.

7.1.1 Measuring FQDN Similarity

Our work is based on the assumption that a certain degree of "redundancy" can be found in DNS mappings. That is, we expect that colliding domain names are often similar, and that neighboring IP addresses host similar sets of domain names. As a first step to modeling this, we require a measure of similarity for domain names. Many related approaches directly use the structure of domain names, and group them hierarchically by domain suffix (e.g., [134]). While this is sufficient to discover that `www.example.com` and `www2.example.com` are *somewhat* similar, it remains unclear *how* similar they are, and how close, e.g., `mail.example.com` is. Furthermore, `www.example.`**org** would not even fall into the same group, as already the TLD "org" falls in a different group than "com", although the domains are obviously highly similar.

Therefore, we define the *Domain Divergence* (DD) $\in [0,1]$ between two FQDNs X and Y. Let X_λ be the λ-LD of X and $|X_\lambda|$ be its length (Y_λ, $|Y_\lambda|$ resp.). Let $|X|$ be the number of domain levels in X ($|Y|$ resp.). For each domain level $\lambda > 1$ we first compute a weight w_λ, based on (i) the hierarchical "importance" of λ (i.e., more significant levels with lower λ receive more weight), and (ii) related to the length of the currently compared domain level. The dampening constant α controls the rate of decrease of w_λ with increasing λ. Based on our experience, the setting $\alpha = 1$ is a good choice, which we use throughout this thesis. For each domain level, we further compute a *partial domain divergence* dd_λ between X_λ and Y_λ, using the Levenshtein Ratio (LR) which is based on counting the necessary edit operations (i.e., add, delete, replace[1]) for transforming one string into another [106]. Precisely,

$$dd_\lambda = \begin{cases} 1 & \text{if } \lambda > \textsc{Min}\left(|X|, |Y|\right) \\ \left(1 - LR\left(X_\lambda, Y_\lambda\right)\right) \cdot w_\lambda & \text{else} \end{cases} \tag{7.1}$$

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

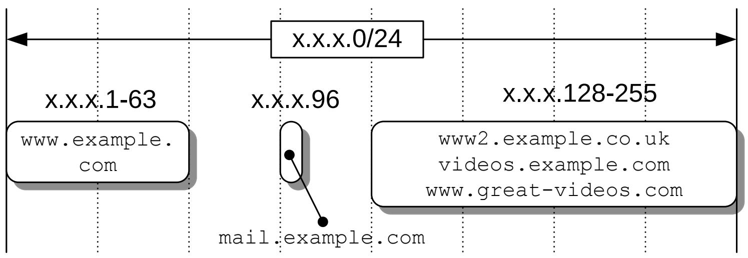


Figure II.7: Example for the information we contain in DNSMap: three IPBlocks holding five FQDNs in total.

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

$$DD := \textsc{Min}\left(\frac{\sum_{\lambda=2}^{\textsc{Max}(|X|,|Y|)} dd_\lambda}{\Omega} + \delta \cdot \beta, 1.0\right) \qquad [0,1] \qquad (7.2)$$

where $\Omega = \sum_{\lambda=2}^{\textsc{Max}(|X|,|Y|)} w_\lambda$ and δ is 0 when the TLDs of X and Y are identical, else $\delta = 1$. I.e., we assign a penalty β when X and Y have different TLDs. We set $\beta = 0.05$ for all following experiments.

7.1.2 Modeling DNS Activity

The detection of malicious DNS agility requires an up-to-date understanding of the historic DNS mappings for a particular range of IP addresses. In our approach, the basic components for holding this information are *IPBlocks*, which describe *continuous* ranges of IP addresses and the set of FQDNs mapping to these IP addresses. Fig. II.7 shows an (imaginary) example: in the /24-network from IP address x.x.x.0 to x.x.x.255, three IP-Blocks are identified. IP addresses 1-63 exclusively host `www.example.com`, IP 96 hosts `mail.example.com`, and IP addresses 128-255 host Example's UK webserver (`www2.example.co.uk`), and a video site which is reachable via `videos.example.com` and `www.great-videos.com`.

However, keeping track of *all* FQDNs mapping to an IPBlock is only the first step. Ultimately, we aim at understanding which FQDN *patterns* are being used per IPBlock, so to be able to quickly evaluate how different subsequently seen FQDNs are. In general, a particular IP range may be used by several classes of very different FQDNs. Therefore, we cluster the FQDNs according to their *Domain Divergence*, and store them in the IPBlock. For each cluster c_k, we derive a *label* L_k. By construction, $DD(L_k, f_i) \leq \Theta$, where the *domain divergence threshold* $\Theta \in [0,1]$ is a system parameter. Any new FQDN-to-IP mapping is then compared to the cluster labels, and is either accepted as "sufficiently similar", or represents a change which requires that a new cluster is created to contain it. Note that this strategy *enforces* that for all FQDNs f_j mapping to an IPBlock there exists a cluster for which $DD(L_k, f_j) \leq \Theta$. Also note that this is far more efficient than comparing a new FQDN to all FQDNs seen previously at this IPBlock. We discuss the details of the approach in the following.

Deriving FQDN cluster labels As a first step, we require a technique for computing *labels* for a set of domain names $\mathcal{D}$. We derive the generalized median strings [33] separately for each domain level, and concatenate them to build the DOMAINMEDIAN of $\mathcal{D}$, which we call the label $\mathcal{L}$ of $\mathcal{D}$. For example, the label of the FQDNs f_1=`www2.`

Algorithm 7.1: DomainCluster

```
input  : fqdns, $\theta$
output: clusters
1  clusters ← List();
2  label ←DomainMedian (fqdns);
3  good ← all d ∈ fqdns where DD (label,d)≤ $\theta$;
4  bad ← all d ∈ fqdns where DD (label,d)> $\theta$;
5  if good is empty then
       // not a single FQDN is well-represented by this label
6      bad 1,bad 2← k-Means (fqdns, k=2);
7      clusters ←clusters + DomainCluster (bad 1);
8      clusters ←clusters + DomainCluster (bad 2);
9  end
10 else if bad is empty then // found label for all FQDNs
11     clusters ←clusters + {(label, good)};
12 end
13 else // recursively cluster "good" and "bad"
14     clusters ←clusters + DomainCluster (good);
15     clusters ←clusters + DomainCluster (bad);
16 return clusters;
```

example.co.uk, f_2=videos.example.com, and f_3=www.great-videos.com from Fig. II.7 is www.example.com. This is intuitive, as the dominant substrings for each domain level across these three FQDNs are www, example, and com, while each of the substrings videos, great-videos, and co.uk appears only once. However, www.example.com is a rather poor label for representing these three FQDNs. This can be seen by computing the domain divergences, i.e., DD($\mathcal{L}$, f_1)=0.09, DD($\mathcal{L}$, f_2)=0.39, and DD($\mathcal{L}$, f_3)=0.58. Especially for f_3, the divergence between the label and the FQDN is large.

Rather than computing an arbitrary label for $\mathcal{D}$, we want to derive a set of *good* labels, such that for each domain $d \in \mathcal{D}$ the divergence between at least one label and d is below a certain *divergence threshold* $\theta \in [0, 1]$. Therefore, we need to derive clusters of $\mathcal{D}$, where each cluster has a label $\mathcal{L}$.

FQDN clustering procedure Algorithm 7.1 shows an unsupervised algorithm for deriving the clusters c_j for a set of domains. The algorithm does not require the full distance matrix for all domains, and therefore needs only to perform few computations of DD. It rather continues to evaluate the distances to an intermediate domain label and proceeds in a divide-and-conquer manner. Note line 6: when we cannot further divide the problem using this strategy, we use k-Means (with k=2) to derive two disjoint subsets of FQDNs and continue for each subset separately. When even that fails, i.e., when one returned subset is empty and the other one therefore contains all FQDNs, we return separate clusters for each of the remaining FQDNs (not shown in Algorithm 7.1). In this case, we still maintain the main requirement that for each FQDN f there exists a label $\mathcal{L}$ for which $DD(f, \mathcal{L}) \leq \theta$. The algorithm does not require us to specify the number of clusters a priori, but rather continues to further subdivide inhomogeneous ("bad") FQDNs and returns as many clusters as are needed to derive a "good" label for each FQDN (i.e., with DD≤ θ). An example result using real data is shown in Fig. II.8.

Note that our objective here differs from many other clustering applications. For example, the widely used DBSCAN clustering algorithm reveals dense regions (i.e., clusters) of arbitrary shape [56]. It is driven by the idea of assigning related elements to the same

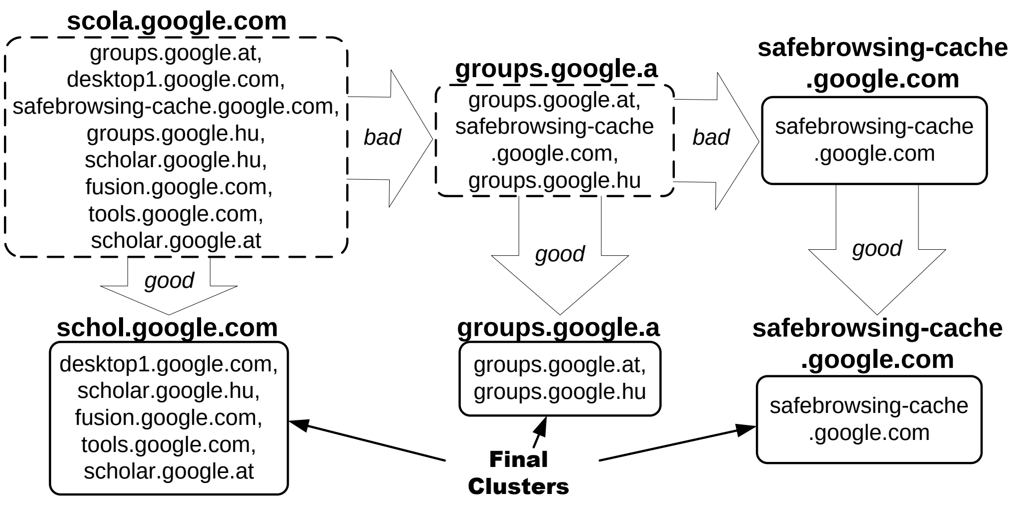

scola.google.com
groups.google.at, desktop1.google.com, safebrowsing-cache.google.com, groups.google.hu, scholar.google.hu, fusion.google.com, tools.google.com, scholar.google.at
bad
groups.google.a
groups.google.at, safebrowsing-cache .google.com, groups.google.hu
bad
safebrowsing-cache .google.com
safebrowsing-cache .google.com
good
good
good
schol.google.com
desktop1.google.com, scholar.google.hu, fusion.google.com, tools.google.com, scholar.google.at
groups.google.a
groups.google.at, groups.google.hu
safebrowsing-cache .google.com
safebrowsing-cache .google.com
Final Clusters

Figure II.8: Domain clustering example. The text in bold print are the labels of the corresponding group of FQDNs.

cluster, where "related" has a transitive meaning. I.e., an element which is close to another element in a particular cluster, should also be assigned to this cluster. As a consequence, such clusters can have a stretched shape, and their center does not necessarily represent the contained elements well. In contrast, our algorithm *requires* that this is the case. All FQDNs need to be related to their corresponding cluster's label, and our notion of "related" is therefore intransitive.

Recall that some services use DNS wildcards, for which we therefore see a large number of domains mapping to the same set of IP addresses. As these FQDNs are by definition similar, the clustering procedure will assign them to the same cluster[2]. Whenever a cluster grows larger than the maximum allowed cluster size MCL, we "collapse" the cluster, i.e., we remove all FQDNs from it and remember that no domains should be further added to it. This avoids that we bloat the DNSMap representation with a large number of these FQDNs, but still retains their characteristic pattern as cluster label.

Data processing DNSMap represents the global state of DNS mappings, as visible to our monitoring system. It provides a data structure to hold IPBlocks, which are keyed on their start IP address. In order to efficiently store, remove, and search for IPBlocks, we use a set of RBTrees (one per /8 network) [16]. The main advantage of such trees in our scenario is their ability to find the IPBlock that contains a certain IP address in logarithmic time (w.r.t. the total number of IPBlocks per RBTree), even when the requested IP is not a key in the RBTree. Additionally, the DNSMap maintains a list of contained FQDN, which are referenced by the individual blocks. Note that this ensures that there exists only one global instance for each stored FQDN, which vastly reduces the memory requirements. In the following, we discuss the main procedure of DNSMap, which is responsible for processing DNS mappings extracted from traffic data.

A new DNS mapping *fqdn:ip* is added by using the ADDMAPPING procedure (shown in Algorithm 7.2), which returns a status code and a score that describe how well it "fits" to previously seen mappings. The individual steps are the following. First, we lookup the IPBlock which contains *ip* (line 1). If exactly the same mapping is already stored in DNSMap, we return a zero score, indicating a perfect match (line 8). Else, we find the

[2]The number of clusters being created depends on the value of θ. In any case, the number of created clusters should be small, else this would contradict the assumption that the FQDNs are similar.

Algorithm 7.2: AddMapping

```
input  : fqdn, ip
output: STATUS, dd
ipb ← DNSMap.GETCONTAININGBLOCK(ip);
if not ipb then // IP not seen recently
    ipb ← IPBlock();
    ipb.ADDCLUSTER([fqdn ]);
    return (NEW, 1.0);
end
if ipb.ACCEPTSFQDN(fqdn) then // DNS mapping is known
    return (OK, 0.0);
end
closestCluster, dd ← ipb.FINDCLOSESTCLUSTER(fqdn);
if dd $\leq \theta$ then // new FQDN, similar to prev.  FQDNs
    closestCluster.ADD(fqdn);
    return (OK, dd);
end
if ipb.ISFULL then // this IPBlock reached max.  capacity
    return (FULL, dd);
end
ipb.ADDCLUSTER([fqdn ]);
return (NEW, dd);
```

cluster to which *fqdn* fits best (line 10). If it does not fit well enough (w.r.t. Θ) (line 11), and if we can create yet another cluster for this IPBlock[3] (line 15), we consider this mapping to represent a significant DNS change (line 19).

With time, the number of clusters per IPBlock would steadily increase (lines 4 and 18), and therefore the system as a whole would become less and less sensitive to DNS changes. Therefore, we provide a periodically run MAINTENANCE procedure which takes care of reclustering the contained FQDNs (using Algorithm 7.1) and removing outdated ones (i.e., FQDNs which were not recently queried from the monitored network). This is further discussed in §7.2.

7.1.2.1 IPBlocks: Merge&Split

Up to now, we considered fixed size IPBlocks, which statically represent a certain number of IP addresses. We initialize each new IPBlock using a single IP, and therefore require a procedure for growing IPBlocks such that they represent a *range* of IP addresses. This is accomplished by merging neighboring IPBlocks, given that we find that they host similar FQDNs. Conversely, we provide a splitting procedure for reverting previous merge operations, so to be able to reflect DNS changes over time. A required key concept is the similarity σ between two IPBlocks, which we define in the following.

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

[3]We restrict the maximum number of clusters for performance reasons, see §7.2.1.

Definition 1 (MERGECONDITION) *Two IPBlocks $\mathcal{A}$ and $\mathcal{B}$ are merged* iff *(i) they are direct neighbors in the IP address space, (ii) they contain IP addresses from the same Autonomous System, and (iii) $\sigma_{\mathcal{A},\mathcal{B}} > \gamma$ and $\sigma_{\mathcal{B},\mathcal{A}} > \gamma$.*

In other words, we merge two IPBlocks when at least a percentage γ of FQDNs of each IPBlock are in clusters which are similar to the neighboring IPBlock's clusters. We set $\gamma = 50\%$, i.e., merging requires at least a simple majority of similar domains.

Conversely, consider two IPBlocks $\mathcal{A}$ and $\mathcal{B}$ that were created from a third IPBlock $\mathcal{Z}$ such that $\mathcal{A}$ represents the first half of $\mathcal{Z}$'s IP addresses and $\mathcal{B}$ the second half. For each cluster in each IPBlock, we keep track of the IP addresses which were recently used by any of the FQDNs in a particular cluster[4]. A cluster is therefore *active* on a certain IP range when any of the FQDNs in this cluster used any of the IP addresses in this IP range. Let $\mathcal{A}$ contain all those domains of $\mathcal{Z}$ where the containing cluster was active in $\mathcal{A}$'s IP range (and for $\mathcal{B}$ respectively).

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

7.2 System Overview

The set of IPBlocks as a whole is a representation of "what maps where", and constitutes the foundation of our approach. In the following, we discuss a system which uses the DNSMap construction to model Internet-wide DNS activity from captured traffic data. The system constantly updates the contained IPBlocks (using the previously defined split and merge operations) to keep track of configuration changes and newly appearing services, and regularly removes outdated information.

Data Processing & Output The system receives DNS traffic (UDP port 53) from a monitoring probe, which discards responses for which no query was observed, in order to avoid traffic injections, and extracts NOERROR query responses, i.e., those responses that were answered by a DNS resolver and contain one or more DNS mappings. A preprocessing module removes duplicate DNS mappings within a time window of 1,800 seconds, as duplicates contain no additional information that would be relevant to our system, but would only increase the processing load. Each queried name and each corresponding IP address (i.e., each mapping) is added to the DNSMap by calling the main ADDMAPPING routine. Every call to ADDMAPPING which returns the status code *NEW*, triggers the textual output of the following information:

$\langle$`timestamp`$\rangle$ $\langle$`FQDN`$\rangle$ $\langle$`IP`$\rangle$ $\langle$`score`$\rangle$ $\langle$`count`$\rangle$

For example: `1321469613 static.ak.facebook.com 77.67.4.40 0.614 65.`

The $\langle$`count`$\rangle$ field holds the number of IPBlocks to which $\langle$`FQDN`$\rangle$ mapped at the time $\langle$`timestamp`$\rangle$ of the DNS change. This can be found by a simple table lookup, and is being used later by the malware detector described in §9. Note, however, that $\langle$`count`$\rangle$ is a lower bound for the real number of occurrences of $\langle$`FQDN`$\rangle$, due to the fact that we limit the storage capacity per IPBlock (line 15 in Algorithm 7.2).

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

[4]This can be trivially implemented with a bit field of length n, where n is the number of IP addresses represented by an IPBlock. In this field, a value of one means that the corresponding IP address was recently used by the cluster, while a value of zero means that it was not used.

from the monitored network, i.e., on which we have *limited visibility*. Therefore, we typically receive several 100,000 change events per day.

Maintenance operations We run asynchronous maintenance operations at configurable time intervals. In order to compress the DNS mapping information and to find continuous IP ranges which host similar services, we evaluate MERGECONDITION for all IPBlocks after every time interval Δ_{Mg}. We recluster each IPBlock using Algorithm 7.1 in advance, to ensure that the cluster labels of any two IPBlocks were created at the same time. This is important, as cluster labels can also be created by Algorithm 7.2 (lines 4 and 18). The FQDN which happens to be observed first becomes the label of the newly created cluster, and any similar FQDNs seen later are added to this cluster without changing its label. For another IPBlock, the order in which FQDNs are seen may be different, and therefore the cluster labels may differ as well. Reclustering addresses this issue, as the order in which the FQDNs were seen is irrelevant for Algorithm 7.1. Note, however, that only those IPBlocks need to be reclustered for which Algorithm 7.2 created a new cluster during Δ_{Mg}.

Furthermore, we run cleanup operations after each time interval Δ_{Ma}, to remove outdated information from DNSMap. More precisely, the following sequence of actions is performed. First, we remove all IPBlocks which do not contain any FQDN clusters. Secondly, for each IPBlock, we remove all empty clusters. And finally, we remove all FQDNs from all clusters from all IPBlocks. This implements the following removal strategy. All FQDNs are removed at most Δ_{Ma} after they were observed. Clusters which are not repopulated with any FQDNs during Δ_{Ma}, are removed, and therefore “live” longer than Δ_{Ma}. IPBlocks are removed only when they do not contain any clusters, which can happen at earliest $2 \cdot \Delta_{Ma}$ after their creation.

Following the cleanup operations, we evaluate SPLITCONDITION for all IPBlocks. Note that splitting and merging implicitly implements a divide-and-conquer scheme for iteratively finding the optimal IPBlock configuration. However, this procedure will of course only converge to stable IPBlocks if the analyzed DNS information by itself “stabilizes”. In general we assume that this is not the case, therefore the IPBlocks configuration should constantly evolve, dependent on the observed DNS data and the resulting clusters per IPBlock, i.e., depending on domain divergence threshold θ.

7.2.1 Parameters and Tuning

We summarize DNSMap’s parameters and the settings we use in Table 2. The settings of Δ_{Mg}, MCL, and MSZ mostly affect the degree of loss of the system’s DNS information compression. More aggressive settings (i.e., lower Δ_{Mg} and higher MCL and MSZ) consume more system resources (i.e., CPU and memory), and create more accurate DNSMap representations. We set Δ_{Mg} to six hours, which we found to be a good tradeoff between computational overhead (merging is costly) and memory requirements (aggregating IP addresses saves space). Our experiments (discussed in §8) revealed that the limits imposed by MCL and MSZ are reached for less than 0.1% of the IPBlocks and clusters, and have therefore limited impact. The most critical parameters are the divergence threshold Θ and the maintenance interval Δ_{Ma}. We provide guidance for tuning them in the following.

7.2.1.1 DD Threshold Θ

The configuration of Θ controls DNSMap’s ability to absorb FQDN changes, and heavily influences the system’s performance. High settings (i.e., closer to 1.0) make the system more “permissive”, and lead to a lower number of change events. Furthermore, the overall number of clusters is reduced, which leads to a lower memory footprint, and to faster processing times. Low settings (i.e., closer to 0.0) have the opposite effects, and make the

Param.	**Description**	**Setting**
Θ	DD threshold	[0.25,0.30,0.35,0.4,0.45]
Δ_{Ma}	maintenance interval	2 days
Δ_{Mg}	merge interval	6 hours
MCL	an IPBlock of n IP addresses can store at most $(1 + log(n))\cdot$MCL clusters	50
MSZ	max. number of FQDNs per cluster to store	30

Table 2: DNSMap parameters.

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

Table 3: Examples for domain divergences in Fig. II.7.

system more sensitive to differences between two given FQDNs. In the following, we investigate how low we can set Θ while still allowing for small variations in the DNS mappings.

The *Domain Divergence* provides a numerical measure for the similarity of two FQDNs which aims at reflecting the intuitive interpretation of a human analyst. Table 3 shows the domain divergences between the FQDNs from Fig. II.7, and gives a first “hands-on” idea of the behavior of this measure. For our application, we definitely want to consider, e.g., d_1 and d_2 similar, which requires $\Theta \geq 0.30$. On the other hand, we clearly want to differentiate, e.g., d_3 and d_5, therefore $\Theta < 0.63$.

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

By construction, the weight of the Levenshtein ratio diminishes with increasing domain level λ (see Eq. (7.1) on page 58). Therefore, we investigate the comparison of two FQDNs which differ only w.r.t. 3-LD. The influence of the unweighted string difference dd_3/ω_3 on DD depends on the fixed w_3, as well as on the relative length of 3-LD. Therefore, we define this length as $\zeta = l_3/(l_2 + l_3)$. Fig. II.9 illustrates the influence of a string difference on 3-LD on the total domain divergence DD between two FQDNs, assuming that both contain three domain levels and do not differ at all w.r.t. 1-LD and 2-LD. For example, in order to consider `www.exampledomain.com` and `mail.exampledomain.com` similar, we would need to set $\Theta \geq 0.19$ ($dd_3/\omega_3 = 1.0$, $\zeta = 4/17 = 0.24$). On the other hand, `xxxxxx.example.com` and `yyyyyy.example.com` would require $\Theta \geq 0.39$ ($dd_3/\omega_3 = 1.0$, $\zeta = 6/13 = 0.46$).

Based on this initial analysis, we evaluate the results for $\Theta = [0.25, 0.30, 0.35, 0.4, 0.45]$

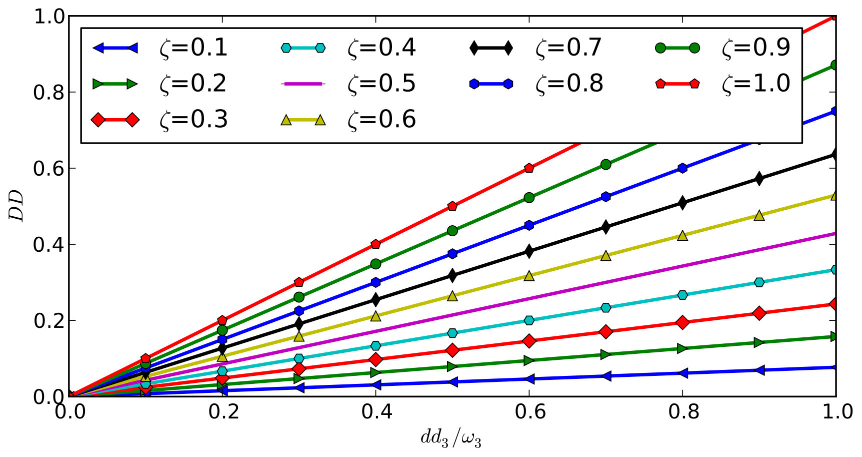


Figure II.9: Domain divergences DD for FQDNs with three domain levels which differ only w.r.t. 3-LD. Note that $\zeta = 1.0$ is an unreachable upper bound (as always $l_2 > 0$).

in the following experiments. Lower settings are to be preferred, as long as they do not severely impact the performance and number of false positives.

7.2.1.2 Maintenance Interval Δ_{Ma}

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

CHAPTER 8

System Evaluation

We evaluate DNSMap using data set DNS–DS1 from a large network operator (see Table 4). It contains two weeks of DNS queries, with ∼13 million unique FQDNs and ∼1.6 million unique IP addresses. We pursue two main goals: first, we analyze the performance of DNSMap and investigate if it is able to cope with large amounts of DNS data. Second, we evaluate the quality of DNSMap's output, and discuss its ability to model DNS mappings. All experiments are done with a Python implementation of DNSMap and a standard desktop PC with 16 GB of main memory.

8.1 Performance

Fig. II.10 shows the performance of DNSMap for processing DNS–DS1, using different settings for Θ. For 48 hours of DNS data we require only ∼1-2 hours of processing time, with a linear trend throughout the entire two weeks. Interestingly, and in contrast to our initial expectations, the system does not necessarily run slower with lower Θ. We found that often the main difference in running times originates from the time required for the evaluation of SPLITCONDITION. This can be seen from the marked regions in Fig. II.10, which show a relative drift happening every two days of trace time, which corresponds to Δ_{Ma}. Higher values of Θ cause more IPBlocks to be merged over time, and therefore reduce the overall required number of evaluations of SPLITCONDITION. However, larger IPBlocks may also have more clusters, and therefore the time required by call increases. Overall, the setting of Θ has only limited effect on the processing time, and easily allows for realtime processing.

Next, we investigate the volume of DNSMap's output in terms of IPBlocks and change events over time. Fig. II.11 shows the results for $\Theta = 0.35$. After an initialization period of 48 hours, the system reaches a stable working point which, from then on, only adapts to changes in traffic volume (i.e., the number of IP addresses seen) depending on the time of day. The stable number of IP addresses also explains the linear trend in processing performance. Note that the number of IPBlocks is not significantly lower, which indicates that the vast majority of IP addresses in the Internet do not host similar services as their neighbors. The number of change events also depends on the time of day, and is relatively

Name	*Time frame*	*Total Mappings*	*Unique FQDNs*	*Unique IP addresses*	*Unique Clients*
DNS–DS1	11/22/2010–12/05/2010	3.2B	13M	1.6M	∼500k
DNS–DS2	11/15/2011–11/21/2011	2.4B	5.4M	1.2M	∼1M

Table 4: Data sets from large access provider.

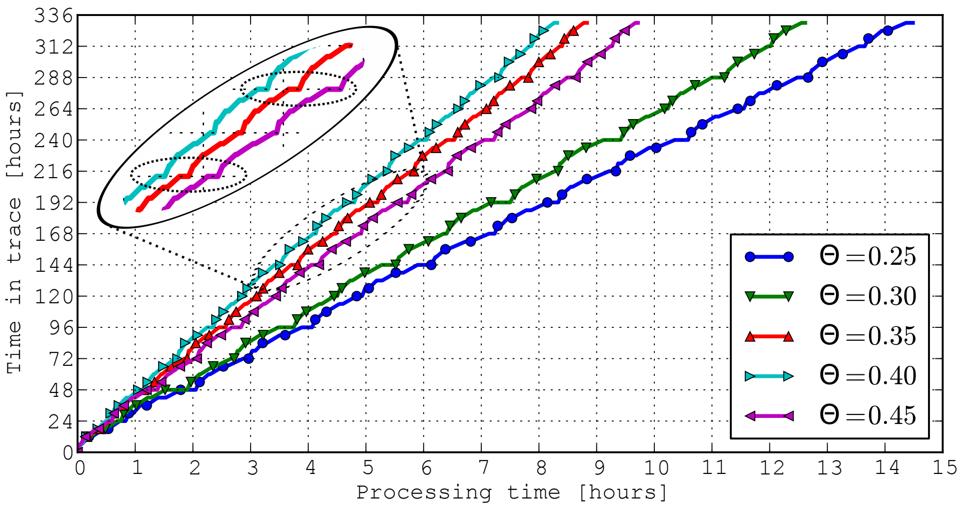


Figure II.10: Processing time for DNS–DS1, using different Θ.

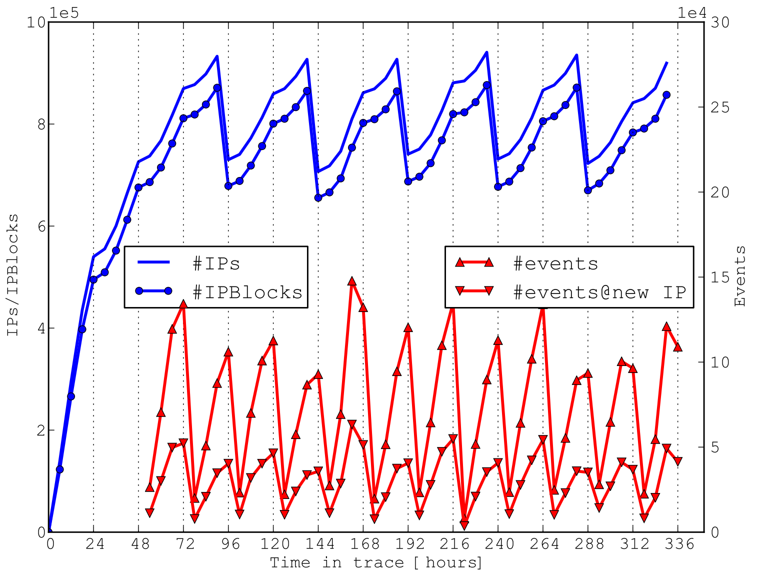


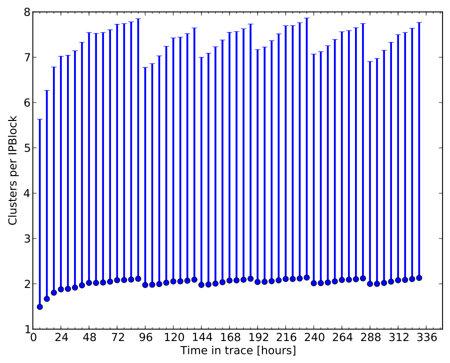
Figure II.11: Number of IP addresses/IPBlocks/events created by DNSMap over time, for DNS–DS1, with $\Theta = 0.35$. By configuration, the first events are produced after 48 hours. Note the double scale.

stable throughout the entire two weeks of data. Around a third of all changes relate to new IP addresses (i.e., IP addresses which were at this time not contained in DNSMap), which underlines the high dynamicity of Internet DNS traffic.

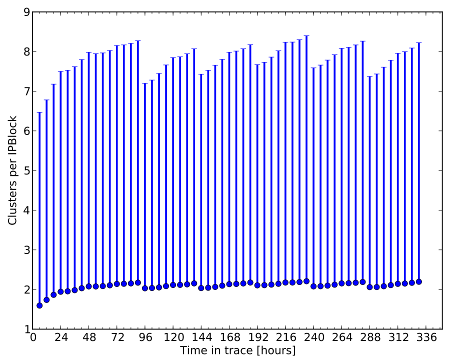
8.2 Benign Service Agility

The main objective in the design of DNSMap is the ability to describe the agility of IP address ranges, and to find a set of labels for the FQDNs which map there. Ideally, most IP ranges would host similar FQDNs over time, and can therefore be distinguished from highly agile malware activity. In the following, we evaluate this using our data set DNS–DS1 (see Table 4). We set $\Theta = 0.35$ and analyze DNSMap's output from processing the entire two weeks of DNS data.

A first indication of the overall DNS agility is given by the final number of clusters per IPBlock. Many clusters would indicate many dissimilar FQDNs per IP, which would



(a) Clusters per IP over time



(b) Clusters per IPBlock over time

Figure II.12: Mean number of clusters per IP address (left) and IPBlocks (right) for processing DNS–DS1 with θ=0.35. The error bars show the (positive) standard deviation.

contradict with the assumption that we can extract stable FQDN patterns per IPBlock. After processing the entire two weeks of data, DNSMap stored ∼850k IPBlocks and ∼3.4M FQDNs. The mean number of clusters per IPBlock was 2.2, with a 95th's percentile of 6.0 clusters. Therefore, only 5% of the IPBlocks required more than six clusters for containing all FQDNS mapping to them in (at least) the last four days ($2 \cdot \Delta_{Ma}$). 75% of all IPBlocks just had a single cluster. The detailed development of the number of clusters over time is shown in Fig II.12. Overall, there is evidently significant stability in the extracted FQDN patterns.

We use DNSMap for analyzing the service deployment strategies for a set of selected ASes. We cover large services (Google, Facebook), CDNs (Akamai, Level 3, Peer 1), a cloud provider (Amazon), a hosting provider (Hetzner), and a large online dating site (VKontakte). The IPBlock representation and the Domain Divergence metric allow us to compare the difference of IPBlocks within an AS, and ultimately find out which IP ranges are used for hosting similar FQDNs. Two IPBlocks $\mathcal{A}$ and $\mathcal{B}$ are similar if they contain similar cluster labels. For each cluster label in $\mathcal{A}$ ($\mathcal{B}$) we find the label in $\mathcal{B}$ ($\mathcal{A}$) such that the DD between the two labels is minimal. The difference score for $\mathcal{A}$ and $\mathcal{B}$ is then defined as the mean of these divergences. For the sake of better representation, we restrict ourselves to 100 *consecutive* IPBlocks per AS. Fig. II.13 shows the results, with the number of IP addresses per shown AS being shown in the captions of the figures. The diagonal represents the difference of each IPBlock to itself, and is therefore always zero. Note that high numbers of IP addresses implies that large IP ranges host similar services, and were therefore merged to single IPBlocks.

The differences between the individual plots reveal the different hosting purposes and requirements of the considered networks. **Google** (a) hosts a rather homogeneous set of services, and therefore the differences between the IPBlocks are low. Note the various blue spots in the figure, which indicate similar IPBlocks that are not immediately consecutive to each other. This behavior is expected from a large service as Google, which typically hosts FQDNs following a certain pattern (i.e., `*.google.com`). The only exception is the small region in the upper left (IPBlock 1-10), which appears to partially host a completely different set of FQDNs. Indeed, these are single IP addresses inside Google's range which host a variety of small, third-party sites. This is interleaved with single IP addresses hosting Google's DNS servers (i.e., `ns{N}.google.com`).

Similarly to Google, **Facebook** (b) mostly hosts a homogeneous set of FQDNs (e.g., `www.facebook.com`). However, there are exceptions. The dark blue square at the upper left hosts a number FQDNs following the patterns `outmail00{N}.snc4.facebook.com` and `outcampmail00{N}.snc4.facebook.com`. Furthermore, the two IPBlocks 58 and 59 host `hphotos-ash2.fbcdn.net`.

The CDN providers **Level 3** (e) and **Peer 1** (f), as well as the hosting provider **Hetzner** (g) host a rather diverse set of services, which is expected from such networks. The notable exception is **Akamai** (c), which partially contains a surprisingly homogeneous set of IPBlocks. We found that these ranges are exclusively used by large services (Facebook, Twitter, Symantec).

Amazon (d) is a particularly interesting case, which shows a two-fold face. The large (mostly) blue square exclusively contains FQDNs with the format `*.cloudfront.net` and represents Amazon's cloud services. The remaining IPBlocks host a highly diverse set of FQDNs, and resemble CDN or hosting provider activity. Note the high number of 3,532 IP addresses which are represented by these 100 IPBlocks. This indicates that the system was able to successfully merge many neighboring IPBlocks with hosted similar (CloudFront) FQDNs.

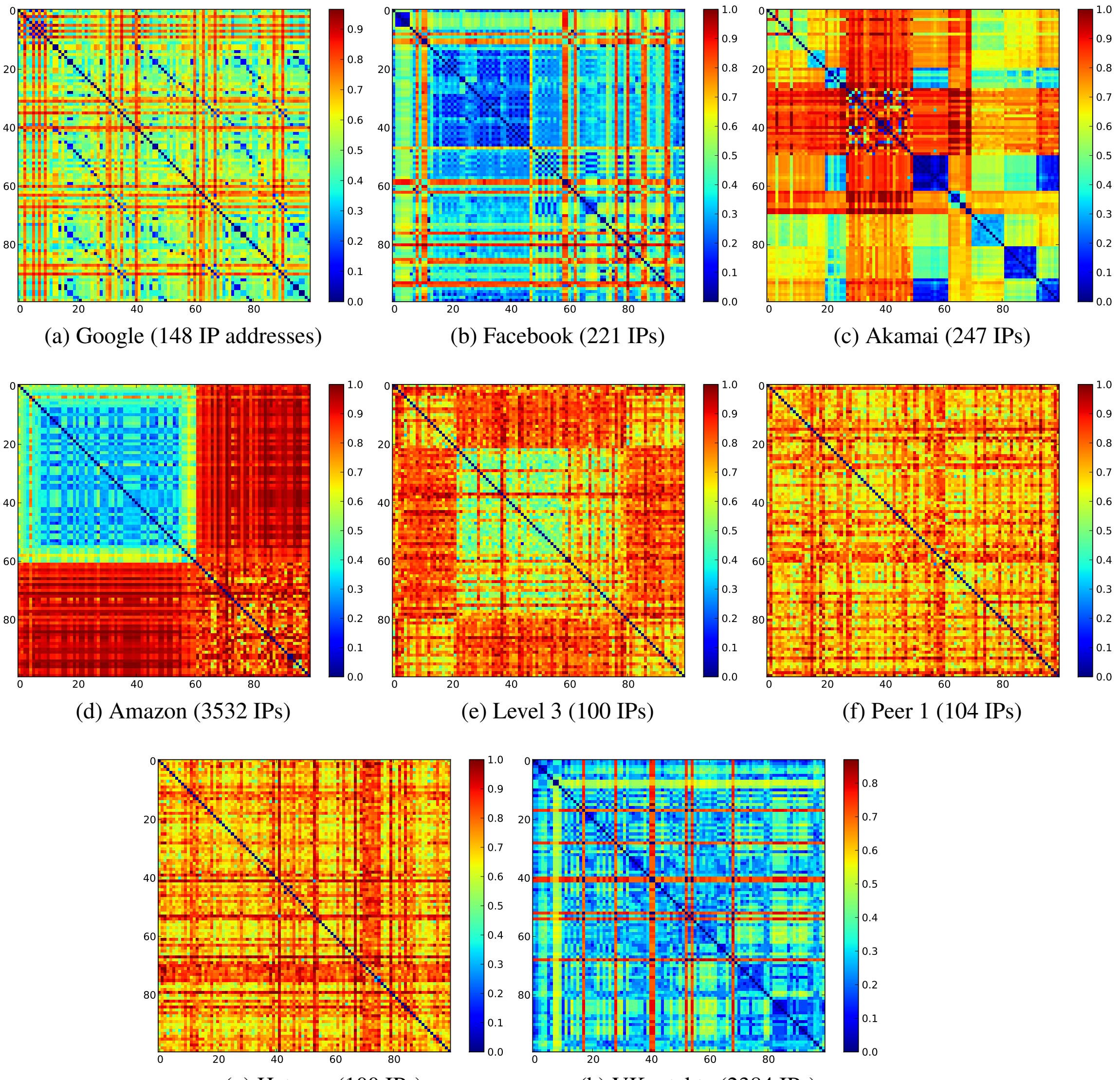
VKontakte (h) almost exclusively hosts FQDNs with the suffix `vkontakte.ru`. Some of the few exceptions are, e.g., IPBlocks 53 and 55, which both host the single FQDN `vk.cc`. Note the large number of 2,384 IP addresses, which again indicates successful merging of a large number of IPBlocks.

Apparently, even in large networks, there exist subranges with a distinct hosting profile, and DNSMap is able to reveal and model them. However, so far we investigated DNSMap's status at a particular moment in time. Therefore, we discuss in the following the development of the representation of a number of particularly busy IP ranges over time. In particular, we pick the /24 network of each of the ASes considered above, for which we observed the most IP addresses being used. We define the *difference* between two IP addresses as the average DOMAIN DIVERGENCE between the cluster labels of the containing IPBlocks. We show one comparison per IP-address pair, and the difference between two IP addresses from the same IPBlock is therefore zero.

Fig. II.14a shows the differences for Akamai, w.r.t. the hosted FQDNs, after six days of DNS-DS1[1]. The diagonal represents the difference of each IP to itself, whitespace indicates that the corresponding IP has not been seen by DNSMap. Three different sub-ranges are clearly visible, which are all rather homogeneous by themselves, but quite dissimilar from each other: (A:10-80) addresses ending on 10-80 host a large variety of different FQDNs of smaller services; (B:105-160) host various Facebook services, plus a small set of other (large) sites (e.g., Symantec); (C:200-250) almost exclusively host Facebook sites. This apparent dissimilarity indicates again that we are able to identify sub-ranges with specific hosting patterns. Fig. II.14b shows the same information after processing all 14 days with DNSMap. Interestingly, the relative differences between the IP addresses are almost unchanged. Note, however, that some slightly larger blocks appeared along the diagonal: the availability of more DNS data over time apparently caused DNSMap to merge neighboring IP addresses. Finally, Fig. II.14c shows the difference between both snapshots. We find the containing IPBlocks for each IP at both times, and compute their differences. Block A shows some limited variation in the hosted FQDNs, which is expected for a range of smaller sites which are observed rarely. Still, most IP differences are well below $\Theta = 0.35$, therefore most of these services would not trigger a change event. Blocks B and C are remarkably stable. Even after eight days, the cluster labels are highly similar.

Analogously, the same results are shown for the other analyzed networks in Fig. II.15 to Fig. II.19. In the following, we discuss the most relevant findings.

[1]We chose six days to allow for some initial training for more representative results.


(a) Google (148 IP addresses) (b) Facebook (221 IPs) (c) Akamai (247 IPs)

(d) Amazon (3532 IPs) (e) Level 3 (100 IPs) (f) Peer 1 (104 IPs)

(g) Hetzner (100 IPs) (h) VKontakte (2384 IPs)

Figure II.13: Difference scores for 100 consecutive IPBlocks of selected ASes.

In Google's /24 network (Fig. II.15), we find only six IPBlocks, representing 120 IP addresses in total. The IPBlocks host similar services, and there are no significant differences between the snapshots after six and 14 days, respectively. Facebook's range (Fig. II.16) appears similar, although far less (i.e., 36) IP addresses are being used.

Hetzner's address range shows a completely different picture (Fig. II.17). Apparently, not a single IPBlock contains more than one IP address. Again, this is expected from a hosting provider, serving many small sites. While there is some difference between the six and the 14 days snapshot, it is still limited, and for most of the IP addresses below $\theta = 0.35$. Note again that the difference in time between the two snapshots is 8 days, and that the problem of limited visibility (see §6.3) is especially pronounced for such hosting providers. Still, there exists some stability in the DNS activity, which can be revealed by DNSMap.

For VKontakte (Fig. II.18), our system created a single IPBlock with 253 IP addresses. As all IP addresses are in the same IPBlock, the difference between them is zero for both snapshots. However, there is some (limited) difference *between* the snapshots, as the cluster labels changed over time. Precisely, while the IPBlock after six days contained the

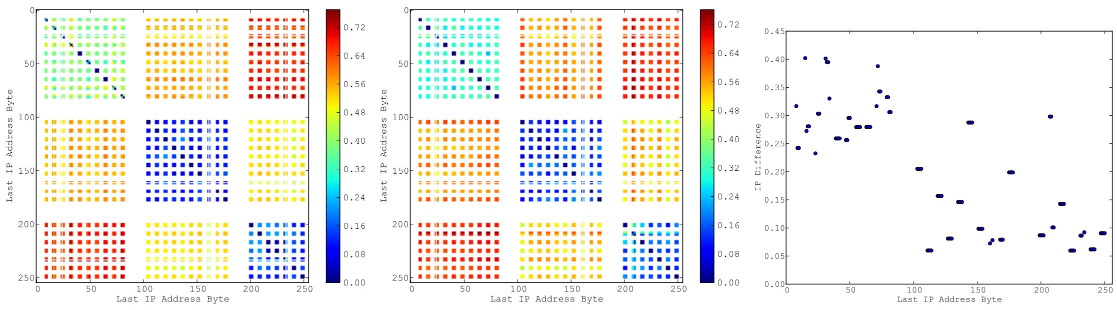

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.14: The figures (a) and (b) represent the status of the DNSMap representation at 6 days (resp. 14 days) into the data set. Shown are the differences between the observed IP addresses of the /24-network 92.122.212.0 of Akamai for which we saw the most IP addresses (102 out of 255) being used. We show the differences of all pairs of IP addresses in this range, w.r.t. to the FQDNs they host. Figure (c) shows the difference over time.

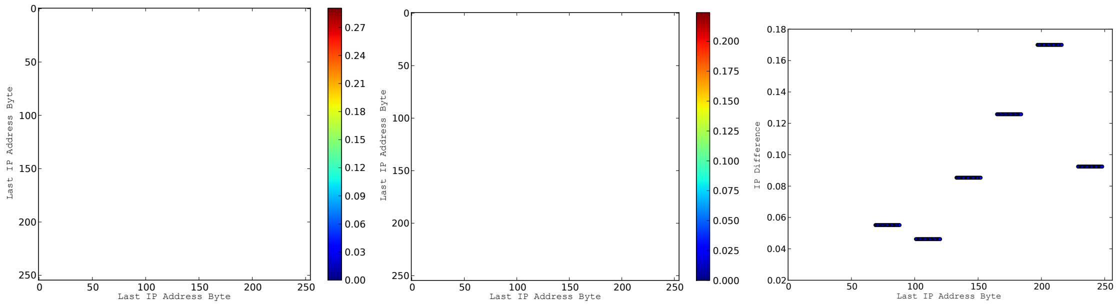

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.15: Google, 120 IP addresses (173.194.18.0/24).

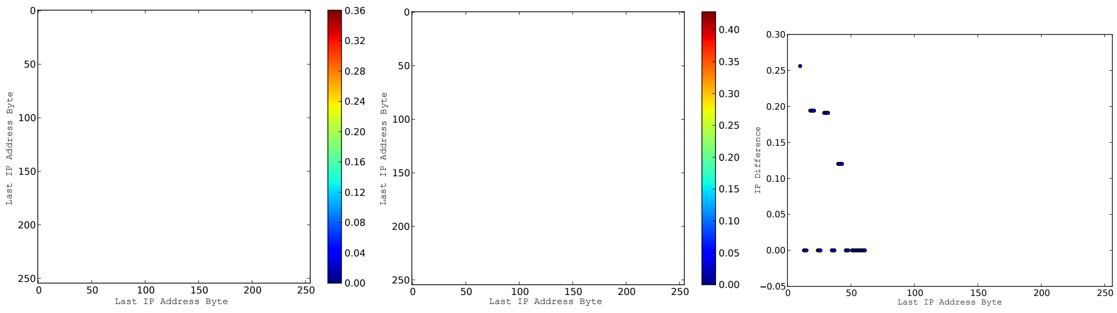

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.16: Facebook, 36 IP addresses (66.220.147.0/24).

label `cs1254.vkontakte.ru`, the IPBlock representing the same range after 14 days carried the labels `cs12465.vkontakte.ru` and `cs12465.vk.com`. DNSMap reacted to new FQDNs which mapped to this range, and adjusted the labels to reflect this new activity. Similarly, also Amazon (Fig. II.19) contains only one IPBlock. The differences between the cluster labels of the two snapshots again originates from newly observed mappings, to which DNSMap reacted accordingly. We show the cluster labels of both snapshots in Table 5. For this IP range, we observe both the problem of DNS wildcards (i.e., `*.profile.iad2.cloudfront.net`) as well as limited visibility (i.e., small sites

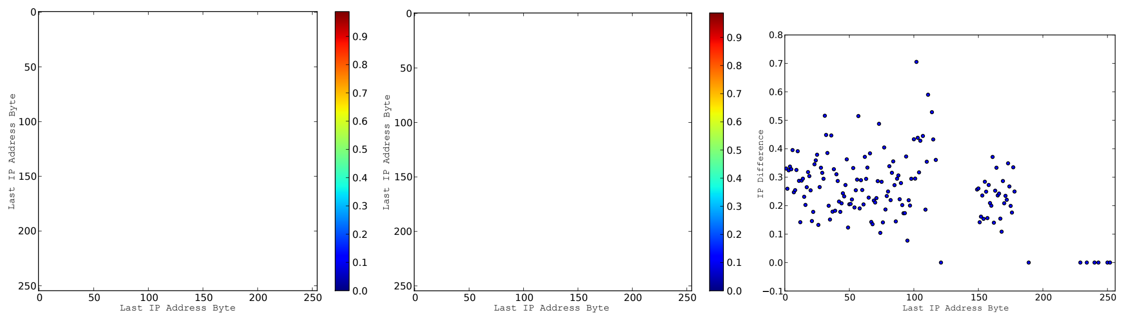

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.17: Hetzner, 151 IP addresses (213.133.104.0/24).

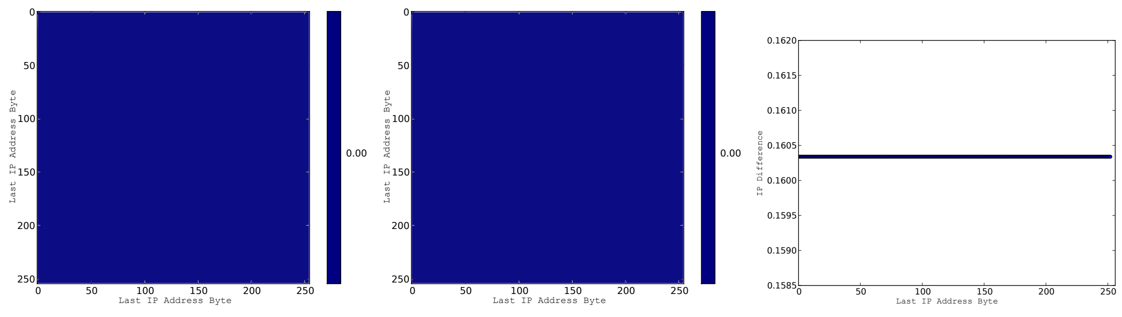

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.18: VKontakte, 253 IP addresses (95.142.201.0/24).

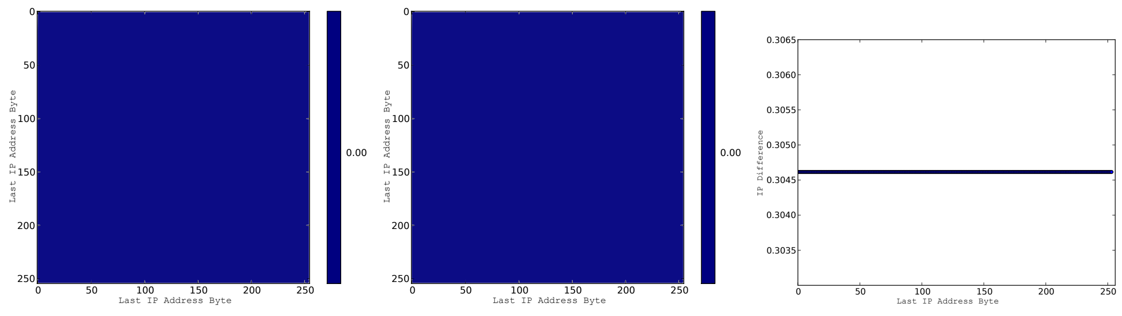

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.19: Amazon, 254 IP addresses (216.137.33.0/24).

like `static.udmserve.net`), which introduce some agility. Still, DNSMap is able to keep up with these changes and reports only significant changes.

Finally, even the DNS activity in the IP ranges of the two CDNs Level 3 (Fig. II.20a) and Peer 1 (Fig. II.20a) is highly stable over time. Despite their overall hosting diversity (see Fig. II.13), these particular ranges host a highly stable set of services.

Summary DNSMap enables us to "zoom-in" on large networks, and reveals substructures that are used for similar services. The plots represent DNSMap's status after analyzing two weeks of DNS data, and show that even large CDNs use DNS mappings which are stable to a certain extent. We can use the DNSMap representation to exactly pinpoint the IP ranges for which an FQDN is considered normal, and assign anomaly scores that depend on *both* the FQDN and the IP address, respectively. In contrast to other approaches [27][9], we

6 days	14 days
a06cd02a1d5fb04c951b7.profile.iad2.cloudfront.net	aec2738be30f2d7a350c1.profile.iad2.cloudfront.net
a9b16142569117d98020b.profile.iad2.cloudfront.net	af0680d79e50b165a5b9c31a3.profile.iad2.cloudfront.net
a5711c0f3e7e15a73825da.profile.iad2.cloudfront.net	a134a6660bff6915189f4228949cbffcd.profile.iad2.cloudfront.net
ace4c235d3341e43144932af71cea8.profile.iad2.cloudfront.net	a00cbbcd4cce13b66800a108a62ae759e.profile.iad2.cloudfront.net
a485e7be7f03e0fd99f98a4aa886ede90.profile.iad2.cloudfront.net	a655139e3b106b71572156d18415b3626.profile.iad2.cloudfront.net
a7c349373de5c55c8b43140aaeea88e76.profile.iad2.cloudfront.net	a035643db0906600760226041031156345.profile.iad2.cloudfront.net
a43a2597623514040028bc6804d5b72220c.profile.iad2.cloudfront.net	**d2313l3iac0f56.cloudfront.net**
static.socialvi.be	static.socialvi.be
static.sp-ask.com	static.sp-ask.com
adweb.hornymatches.com	adweb.hornymatches.com
static01.flirtfair.de	**d6vb69kil3yqp.cloudfront.net**
universal-ma.sundayskylb1.com	**cdn.udmserve.net**
pxml1.4publishers.com	**widgets.sodahead.com**
	static.udmserve.net

Table 5: Cluster labels for Amazon's 216.137.33.0/24 network. For convenience, similar cluster labels are arranged side-by-side. Significant changes after 14 days are highlighted in bold print.

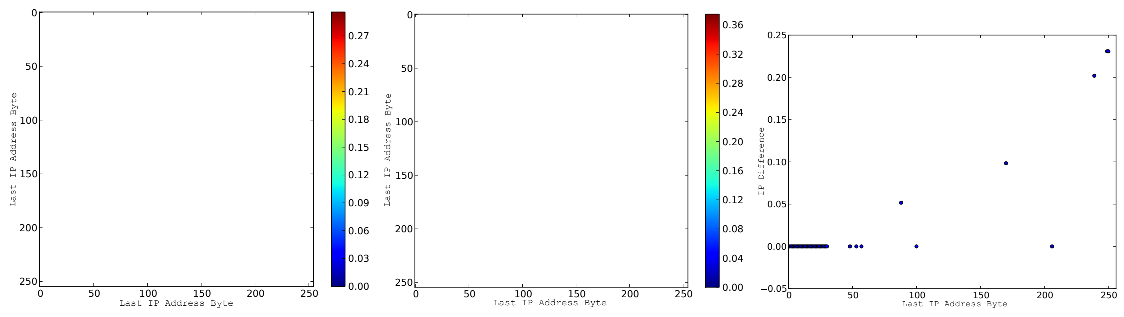

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.20: Level 3, 62 IP addresses (195.122.131.0/24).

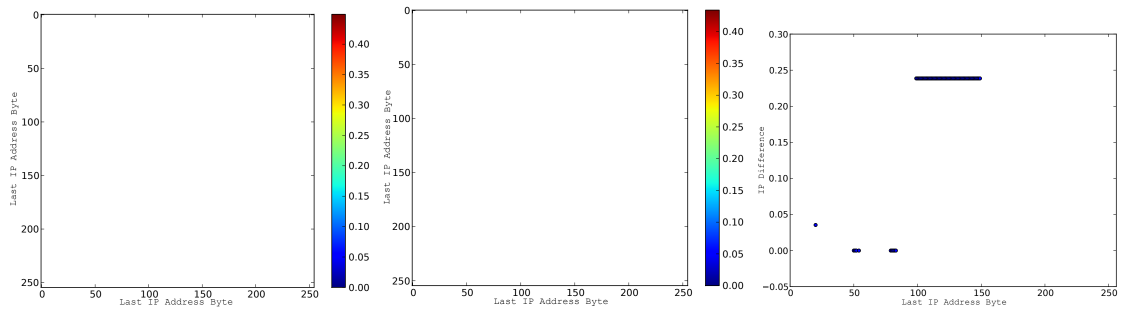

(a) 6 days (b) 14 days (c) Difference 6 days vs. 14 days

Figure II.21: Peer 1, 61 IP addresses (65.39.176.0/24).

would therefore be able to detect that, e.g., `www.google.com` mapping to a Facebook IP is anomalous. DNSMap produces change event messages in these cases, which we further investigate in the following.

CHAPTER 9

Malware Detection

After investigating DNS mapping stability in §6 we developed in §7 an efficient modeling apparatus and evaluated the proposed system in §8. Finally, we can now address the original problem of DNS-based malware detection.

In contrast to many other systems [27, 10, 130, 129], our approach is based solely on the *mappings* between FQDNs and IP addresses. We use DNSMap for tracking which FQDNs map to which IP addresses, and therefore establish an understanding of what represents "normal" DNS activity w.r.t. a specific FQDN and IP. The resulting profiles are the basis of our detection approach. Any DNS mapping which involves an FQDN that does not "fit" to the profiles is considered suspicious, and enters the second analysis stage. There, we represent all suspicious mappings in a certain epoch as a bipartite graph, where nodes are FQDNs and IP addresses respectively, and edges indicate the existence of a suspicious mapping between them. The structural properties of these graphs relate to the agility of the underlying DNS activity. For example, high-degree IP address nodes host many different FQDNs, while high-degree FQDN nodes imply that the respective FQDN maps to many different IP addresses. This allows us to apply standard graph analysis techniques to quickly identify likely malicious sites. E.g., graph components which involve just one FQDN and a single IP are probably not malicious. Conversely, a component which contains mappings between a single FQDN and hundreds of IP addresses, all of which are considered suspicious, represents well the targeted malware model, and is therefore more likely malicious. Most importantly, graph analysis also finds *transitive* connections between FQDNs and IP addresses, and can therefore find groups of graph nodes which by themselves may seem of little interest, but *jointly* appear suspicious.

The final analysis system comprises four main stages, shown in Fig. II.22. The *Parser* reads DNS traffic data (from the wire/from a dump file), and further processes DNS NOERROR request responses. Any queries for an FQDN which were answered with one or more IP address records are extracted, and are forwarded to the *Duplicates Filter*. Recall that our approach is based solely on DNS mappings, therefore there is no added value in multiple evaluations of the same mapping within short time. The filter consequently reports each mapping only once, and ignores any further occurrences within a time interval `filtWindow`. The remaining mappings are then processed by *DNSMap*, which in turn produces a series of change events. All events within a certain evaluation epoch ε are then processed by the *Graph Analysis* module which produces the final output. We discuss the latter component in the following.

9.1 Graph Analysis

DNSMap provides a stream of change events, most of which represent natural, non-malicious DNS activity. In the following, we apply graph analysis techniques for mining these events,

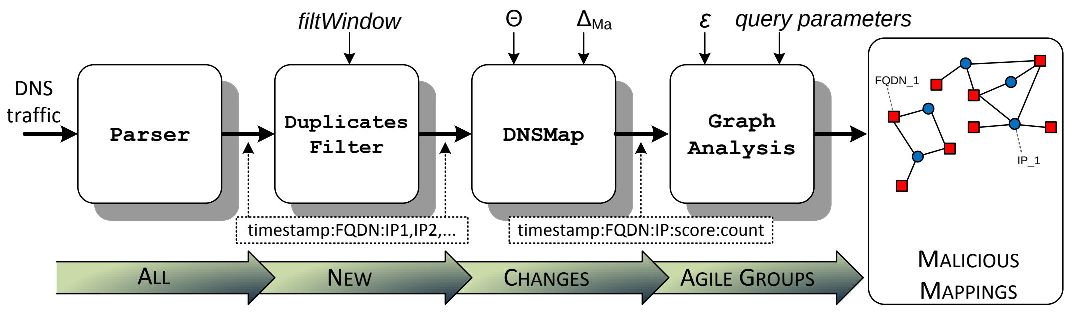


Figure II.22: System overview.

and find the ones which match the malware model. Our approach is based on the observation that malicious DNS agility maps to *group activity patterns* in a graph.

We consider the sets of FQDNs $\mathcal{F}$ and IP addresses $\mathcal{I}$ for which we received a change event within an evaluation epoch ε (e.g., 1 day). These events are represented as a bipartite graph $\mathcal{G}(V, E)$ with vertices V and edges E, where $V = \mathcal{F} \cup \mathcal{I}$ and each $e \in E$ indicates the existence of a mapping between an FQDN and an IP. DNS changes are reported almost instantaneously when the corresponding DNS traffic is observed, and the graph analysis procedure we propose in the following is lightweight. This allows us to implement ε as a sliding window, and create a new graph, using an updated list of change events, every few minutes. This ensures that newly collected evidence is quickly integrated in the graph, which potentially allows for almost immediate detection.

As a first analysis step, we partition the graph and find the set of *connected components*, i.e., subgraphs which are not connected to each other [44]. This reveals groups of FQDNs and IP addresses which are (directly or indirectly) related to each other over time, *and* which were found to represent a significant change. Note that this partitioning is enabled in the first place by DNSMap. A direct representation of DNS mappings to graphs would yield giant connected components, where the majority of services is connected to many others, due to the widespread use of CDNs. DNSMap accomplishes the omission of graph edges which represent activity that is sufficiently similar to previously seen one, and therefore makes the graph separable.

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

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

9.1.1 Agile Group Features

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

ϕ_1 The number of FQDNs per agile group. Malware reuses IP addresses for multiple different FQDNs, in which case this number is high.

ϕ_2 The number of IP addresses per agile group. Malware hosts FQDNs on multiple IP addresses over time, in which case this number is high.

ϕ_3 The number of different Autonomous Systems (ASes) per agile group. We find the AS for an IP using MaxMind's freely available database[1].

ϕ_4 Although a benign FQDN might be known to map to N IP addresses, the graph analysis module may receive events indicating suspicious activity of the same FQDN on M IP addresses, often with $N >> M$. We therefore remove all events that involve FQDNs for which we have significantly more reason to believe that they are benign, than malicious. The maximum $\langle \texttt{count} \rangle$ value C_{max} for a reported FQDN indicates on how many IPBlocks this FQDN was active during ε in total. We compute the ratio $\phi_4 = M/C_{max}$, and consider FQDNs with $\phi_4 < 0.3$ as *likely benign*. For our experiments, we do not consider such FQDNs for further analysis. Note, however, that this does not necessarily imply that the entire agile group is ignored.

ϕ_5 Hosting providers often allocate a set of IP addresses in a sub-network to a large set of (very different) FQDNs. These IP addresses often form dense groups, i.e., they differ only by a small (integer) value. In contrast, malicious hosting is less organized, and uses IP addresses that are scattered over multiple networks. This has been recognized also by other authors, and was often addressed by computing entropy-based measures from the set of used IP addresses [130]. Entropy does however not consider the distance between IP addresses, and cannot capture the difference between *dense* and *sparse* groups of IP addresses. In the following, we therefore derive a score describing this "scatteredness". For each /24-network per agile group, we find the lowest (x_1) and the highest (x_n) of the Integer representation of n IP addresses, and compute the average distance $\mu = (x_n - x_1)/n$ between them. We weight μ logarithmically and derive $\rho \in [0, 1]$. Additionally, ν conveys on how many addresses μ is based, and therefore expresses the confidence level of ρ.

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

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

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

Fig. II.23 shows the values of σ_i for different μ and n. In order to reach $\phi_5 \leq 0.7$, a malicious component must *on average* either use ≥ 21 IP addresses per /24-network, or use at least six IP addresses (per /24-network) which are direct neighbors, or implement a trade-off between these two extremes. This is hard to achieve for malware using agile DNS, which aims at always being "on the move" to evade countermeasures. In our experiments, we remove all agile groups with $\phi_5 \leq 0.7$ Note that this is a conservative setting, as we typically observed $\phi_5 > 0.9$ for malicious components.

[1] `http://dev.maxmind.com/geoip/geolite`

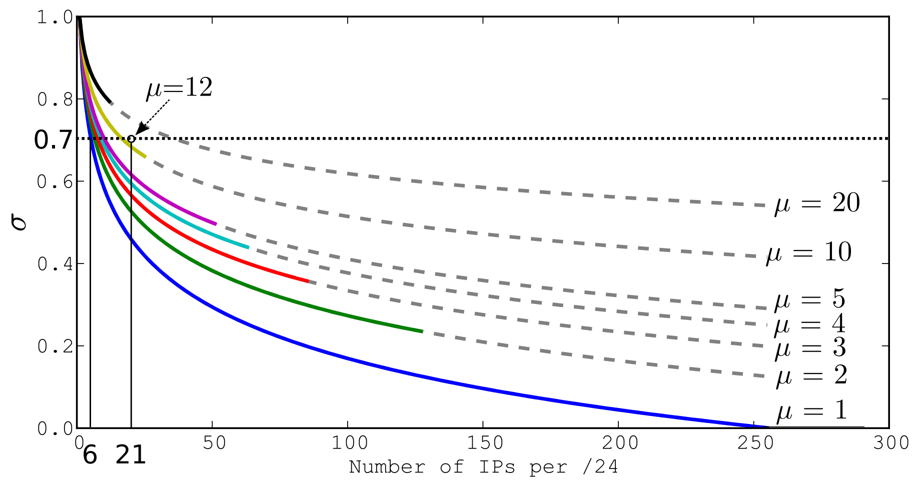


Figure II.23: IP distance scores σ for different numbers of IP addresses per /24-network seen, and different mean distances μ between them. The dashed lines indicate theoretical values which cannot be reached (e.g., there cannot be 100 IP addresses with a mean distance of 20 in a single /24-network).

We stress that one could easily define more features (e.g., based on [27, 10, 130, 129]), with the additional advantage of being able to exploit the graph's structural connection information. However, we did not consider this necessary during our experiments, as the detection results were excellent, and a lower number of features can be tuned more intuitively.

9.1.2 Graph Analysis Parameters

The final analysis stage requires as input a set of six parameters. The settings of ϕ_4 and ϕ_5 were already discussed in §9.1.1. The features ϕ_1,ϕ_2, and ϕ_3 intuitively quantify the FQDN/IP/AS-diversity per graph component over an evaluation period ε. In order to be able to detect even moderately agile malware [95], we aim at setting ϵ as large as possible, i.e., ε=1 week for *DNS–DS1* and ε=4 days for *DNS–DS2* (see Table 4). Together, the settings for ε and ϕ_1, ϕ_2, and ϕ_3 define the lower bound for the rate of agility we are able to detect.

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

9.1.3 Analysis Workflow

As noted already in §4.3, we consider it essential to integrate a human expert in the analysis procedure, in particular for the evaluation of the results. As the Internet as a whole is constantly evolving, we cannot assume a stable system configuration at which a malicious activity is always detected accurately without many false positives. The FQDNs of new, large, benign services do resemble agile, malicious patterns, and should undergo a manual inspection. Therefore, the system's results should be continuously evaluated, and should result in configuration updates. This basic workflow is illustrated in Fig. II.24.

Updating the configuration implies either changes to the parameters discussed in §9.1.2, or providing additional information. We found that providing information about only few

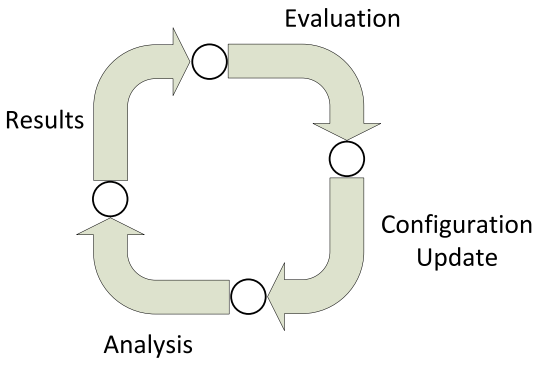


Figure II.24: Basic workflow of our system. A human analyst regularly evaluates the detection results and updates the system's configuration if needed, which in turn leads to improved analysis results in the next iteration.

FQDNs which are actually benign had a dramatic impact on the detection performance. We therefore integrate a whitelisting procedure in our system, which we further discuss in §9.2.2. Note that the derived whitelist is not static and therefore evolves over time to consider new benign activity which resembles malicious one.

9.2 Experimental Evaluation

In the following, we discuss the configuration of the graph analysis stage and the detected malware activity. Again, all experiments were done on a standard PC (Intel i5@3.1 GHz) with 16 GB of main memory, of which we used at most 9 GB during our tests. The data sets (see Tab. 4) were prefiltered with `filtWindow`=1800 seconds before processing them, so to not let disk I/O and the DNS parser dominate the processing time.

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

As noted already in §4.3, the evaluation of our results is a challenging task, as there exists no reliable ground truth. Similar to [27, 10, 130], we use publicly available blacklists for assessing the quality of our results. Note, however, that we never found *all* detected malicious FQDNs in any single blacklist we used. This is not only because of limitations of the underlying analysis systems, but also due to the fact that blacklists are based on different data sets (i.e., Spam emails) from other networks. Typically we do not know what specific techniques were used to flag a domain name as malicious, nor can we say with absolute certainty if the blacklisted domains are actually malicious.

As an example, we consider the blacklist from `exposure.iseclab.org`. The underlying analysis is based on [27], which is conceptually similar to our approach. We downloaded the list on Nov. 22, 2011, i.e., one day after the end of DNS–DS2[2]. Out of $\sim$60,000 domains in the blacklist, only 478 were also present in DNS–DS2. Only 16 of them were

[2]The blacklist service was not available for the time frame of DNS–DS1.

	Blacklist URL	**Note**
BL1	`https://zeustracker.abuse.ch`	
BL2	`https://spyeyetracker.abuse.ch`	
BL3	`https://palevotracker.abuse.ch`	
BL4	`https://amada.abuse.ch`	Discontinued
BL5	`https://www.malwaredomainlist.com`	
BL6	`http://exposure.iseclab.org`	Last update on 2012/07/09
BL7	`https://www.dshield.org`	
BL8	`http://joewein.net/bl-log/bl-log.htm`	

Table 6: Public blacklists used for evaluating our results.

reported as changes by DNSMap, and entered the second analysis stage, which resulted in zero final alerts. We manually checked all 16 domains, and could not find any evidence that they were related to malware activity. In fact, most of the sites are still online (1.5 years after the data set was recorded), which indicates that they are actually benign. Out of the remaining 462 domains, 426 were first observed *during* the initial two-days training phase of our system, and were therefore correctly classified as representing no DNS change later on. The remaining 36 domains were seen the first time *afterwards*. Nevertheless, for 304 out of the 426 we found the *same* group of eight IP addresses hosting 103 *other* domains during our experiments, which were not found by [27]. Viewing this activity as an agile group enabled us later to understand that this was a single cluster of botnet sinkhole IP addresses. We checked both the 122 other domains and the remaining 36 domains manually, using other blacklists (see Table 6) and online services (see below), and found no evidence for malicious activity. The automated comparison with existing blacklists can therefore be grossly misleading, in particular for evaluating the number of false negatives. In our example, we detected all malware-hosting IP addresses reported by [27] without any false positives, while a simple comparison would have yielded 478 false negative FQDN reports.

Consequently, we employ a semi-automatic approach for evaluating our results. We rely on a set of eight publicly available blacklists, shown in Table 6, and on online services[3] for assessing the quality of our detection results in terms of true/false positives. For several domains, we were only able to finally verify them as being malicious by running manual Google searches, which typically led us to a variety of malware analysis sites. Still, for some domains we were not able to find any information online, despite the fact that their activity would definitely deserve a closer look. Due to lack of ground truth, we cannot reliably quantify the number of false negatives. We compensate for that by choosing highly sensitive configurations, and show that even these settings yield extremely few false positives (see discussion in §4.3).

First, we demonstrate that our system can provide valuable results even with minimal training. Then, we discuss a selective whitelisting procedure, which targets only a small set of highly specific FQDN patterns. Finally, we show a variety of different groups of malware activity we discovered.

9.2.1 Results with Limited Training

In contrast to state-of-the-art approaches [9, 27, 130], our system is able to provide reliable results even with minimal training. To demonstrate that, we analyze the change events of the third day (ε=1 day) in our data set DNS–DS1, i.e., after two days of initial training. We set $\Theta = 0.35$ and received 332,606 change events, out of which 181,822 *singles* and 17 *likely benign* FQDNs could immediately be discarded.

[3]`phishtank.com`, `projecthoneypot.org`, `virustotal.com`.

First, we targeted groups of agile FQDNs sharing a set of IP addresses (Malicious Hosting) and discarded all components with $\phi_1 < 10$, $\phi_2 < 2$, and $\phi_3 < 2$. The remaining 199 FQDNs were split in 11 agile groups: the largest one represented four sinkhole IP addresses for botnet mitigation, which hosted 51 random-looking FQDNs. 28 FQDNs hosted on five IP addresses in three ASes clearly represented malware, which we call the *jailcoach* cluster (see §9.2.3.2). Another group of 10 FQDNs related to online pharmacy Spam (e.g., `bargaincapsules.ru`). The remaining eight groups with 110 FQDNs in total represented benign sites using a small set of different hosting providers, which were previously requested not often enough to be known to the system.

Second, we targeted sites which quickly jump from one IP to the next (Fast-Flux) and required that $\phi_1 \geq 1$, $\phi_2 \geq 20$, and $\phi_3 \geq 5$. After filtering the original graph, only eight FQDNs in three groups were left. The first one contained the malicious Fast-Flux domain `ttcuhdwk.biz` (mapping to 38 IP addresses in 28 ASes). The FQDN `36c9f6794ed2d07d.outpostwastes.com` and five similar others mapped to 39 IP addresses in 32 ASes. While we could not find any information on these domains, we suspect that this represented a DNS cache poisoning attack [2]. The remaining FQDN `irc.efnet.org` on 21 IP addresses in 20 ASes was a false alarm. A manual investigation revealed that this FQDN was not requested *at all* during the first two days. Although this represented no malicious activity, it therefore still represents the kind of agile DNS which we aim to detect.

9.2.2 Targeted Whitelisting

The case of `irc.efnet.org` showed that certain benign DNS activity is indistinguishable from malware. This is an intrinsic limitation of DNS analysis, which also affects other approaches. We explicitly acknowledge this, and integrate a fine-grained procedure for targeted whitelisting in our system. That is, in contrast to other approaches which remove large numbers of domains *a priori* [27, 10, 130, 129], we identify those which are truly indistinguishable from malware, and remove only them.

We take advantage from the representation of DNS mappings as graph: some agile groups may be large and contain, e.g., thousands of FQDNs and IP addresses, and therefore it is not immediately clear which of these FQDNs should be checked for potentially being whitelisted. For every agile group, we find the FQDN node with the maximum normalized *node betweenness* [44]. This indicates that it lies on many shortest paths between two other nodes, and is therefore important for the component's connectivity. After checking manually that the FQDN is actually benign, we create a new whitelist entry and rebuild the graph without considering this FQDN. This often causes the group to fall apart, and the individual parts are then removed by the filtering rules. Fig. II.25 shows a simple example.

In other cases, there is no single FQDN responsible for a group of sites being reported. One such service is NTP, which involves a large number of different IP addresses around the globe, which are used for many FQDN aliases (see also [130]). Another example is CoralCDN, which uses FQDNs like `PREFIX.nyud.net`, where *PREFIX* is an FQDN of an existing website. A cached copy of *PREFIX* is located in a large peer-to-peer network, and the IP of the allocated server is returned to the requesting host. In other words, a large number of IP addresses serves an ever changing set of FQDNs, which resembles malware activity. In this case, there exists an FQDN pattern which we should whitelist. Therefore, we find the popularity of FQDN suffixes for each group, and can quickly identify the dominant one, which we then whitelist eventually. Table 7 shows the *entire* set of highly specific whitelist patterns we derived, and which we use for all following experiments.

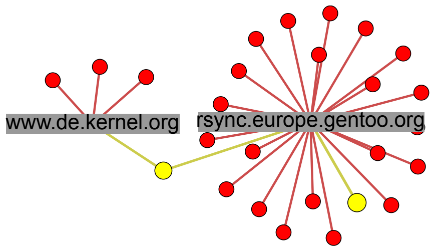


Figure II.25: Whitelisting example: the relative node betweenness of `rsync.europe.gentoo.org` is 92%, and it is therefore by far the most important node in this component. Removing it after a manual investigation causes this component to fall apart and disappear from the results. Note that we collapse several IP addresses from the same AS for better visualization (yellow nodes).

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

Table 7: Whitelist used for experiments.

		Θ=0.25	Θ=0.30	Θ=0.35	Θ=0.40	Θ=0.45
DNS–DS1	FF	4/14	4/13	4/13	3/10	2/9
	MH	9/42	9/33	9/29	7/27	6/26
DNS–DS2	FF	9/21	9/21	9/20	9/19	9/18
	MH	4/19	4/16	4/14	4/10	3/8

Table 8: Results for different selections of Θ and two detection scenarios (FF, MH). Shown is the relation between confirmed malicious agile groups vs. the total number of reported agile groups.

9.2.3 Malware Detection Scenarios

As discussed in §9.1.2, the agile group parameters allow us to encode queries for targeting specific malicious hosting strategies. First, we address *Fast-Flux* (FF) activity by requiring that $\phi_1 \geq 1$, $\phi_2 \geq 20$, and $\phi_3 \geq 5$. This configuration resembles the goals of [130], but is more sensitive and allows for much quicker detection. Second, we find *Malicious Hosting* (MH) patterns by setting $\phi_1 \geq 50$, $\phi_2 \geq 4$, and $\phi_3 \geq 2$. For our experiments, we consider the last week of DNS–DS1 (i.e., ε=1 week), and the last four days of DNS–DS2 (i.e., ε=4 days). For each data set and each detection scenario, we investigate the agile groups which are reported for $\Theta = [0.25, 0.30, 0.35, 0.4, 0.45]$. Table 8 shows the overall results, which demonstrate that even for low settings of Θ the number of false positives is still small. In the following, we discuss the results for Θ=0.35 in detail. As shown in Table 8, this setting consistently revealed all malicious agile groups that were found using lower Θ, and returned less false positives.

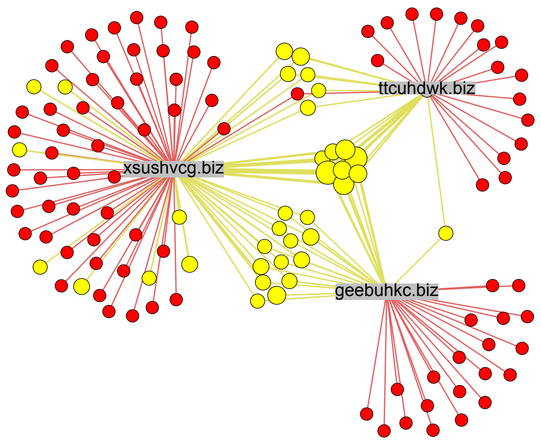


Figure II.26: Group of Fast-Flux sites. Again, multiple IP addresses (red nodes) from the same AS are collapsed to yellow nodes for better visualization.

9.2.3.1 Fast-Flux

For DNS–DS1, we retrieve 291 FQDNs in 13 agile groups, which we sort according to the number of IP addresses per group. The top four groups include three confirmed malicious ones, and none of them contained less than 50 IP addresses. The largest group contains the FQDNs `xsushvcg.biz`, `geebuhkc.biz`, and `ttcuhdwk.biz` (which we found already in §9.2.1), and 365 IP addresses in 155 ASes (see Fig. II.26). The three other malicious groups contained 16/86/152 FQDNs and 136/50/23 IP addresses, respectively. The remaining 9 groups with 34 FQDNs in total were misclassified benign, yet highly agile, sites (e.g., `connect.facebook.net`). Note that this corresponds to a false positive rate of less than 0.0001% of all unique FQDNs seen during one week, even when we assume that the number of false negatives is 10,000 times higher than the number of false positives. We maintain this false positive rate for *all following experiments*.

For DNS–DS2, we retrieve 321 FQDNs in 20 agile groups. A group with two FQDNs on 288 IP addresses in 125 ASes represented malicious Fast-Flux. Less obviously, a group of 12 FQDNs (34 IP addresses/12 ASes) was used for hosting malware (e.g., the Cycbot trojan on `fdg45e.nl.ai`). Interestingly, seven other groups contained only one FQDN and exactly 20 IP addresses in 15-20 different ASes each. All these FQDNs were subdomains of `syringemexican.com`, and used domain prefixes which appeared to be randomly generated. While we did not find any reference to these domains online, we consider this activity to most probably relate to malware. The remaining ten agile groups represented benign activity. Note, however, that the initial training period for DNS–DS2 was only two days and ε=4 days, which was the reason for misclassifying some popular services (e.g., YouTube).

9.2.3.2 Malicious Hosting

For DNS–DS1, we find 29 agile groups with 3,409 FQDNs, of which 9 groups with 892 domains in total represented malicious hosting activity. One particularly interesting group is shown in Fig. II.27 (the *jailcoach* cluster). It contains 147 FQDNs (8 IP addresses, 5 ASes), out of which we found 76 FQDNs in blacklists which were derived from Email Spam. Note that neither the number of IP addresses per FQDN nor the number of ASes is excessively high. Rather, the overall agility of the FQDNs and IP addresses as a group stands out, which made them detectable. The fact that a subset of these FQDNs were found also by the analysis of Spam emails underlines the complementarity of these approaches.

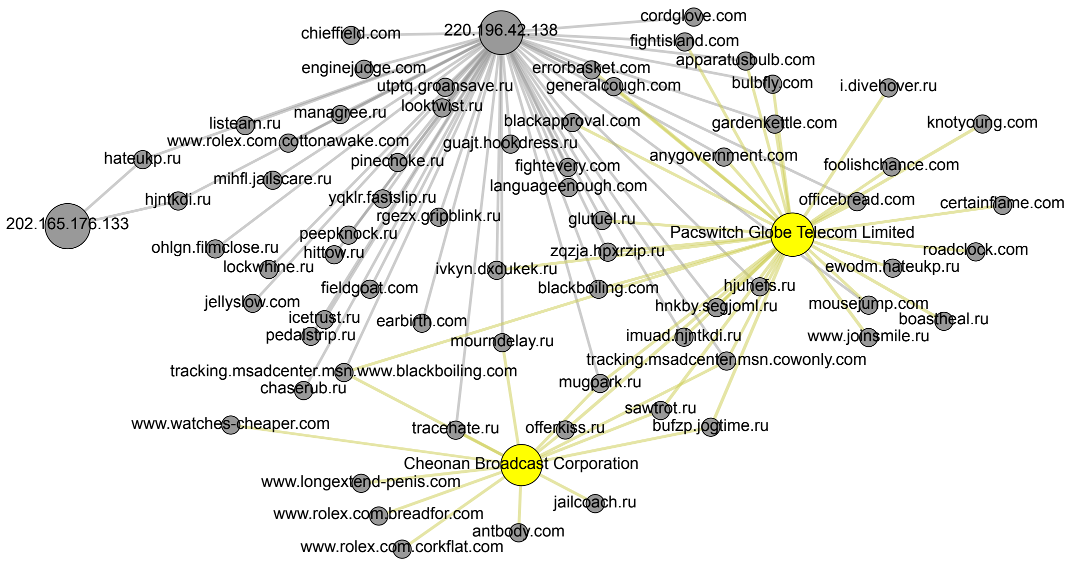


Figure II.27: The *jailcoach* cluster: a group of malware sites, most of which seem to be generated by concatenating two English words (e.g., jailcoach.ru). Few sites relate to selling of fake watches (e.g., www.rolex.com.SUFFIX); others follow a different naming pattern (e.g., hjuhefs.ru). We removed $\sim 2/3$ of the original graph for better visualization.

Spam-based blacklists are typically highly accurate, but are limited in scope. On the other hand, DNS-based analysis is less precise, but is able to provide the big picture for malware activity, which goes beyond the detection of network resources hosting Spam content.

We identified a malicious group of 123 FQDNs (7 IP addresses, 5 ASes) (see Fig. II.28a). Most of these FQDNs seemed to relate to pharmacy Spam and 12 FQDNs were found by the Spam blacklist BL8. However, other FQDNs using the same IP addresses followed a completely different naming scheme (e.g., `igfh.ru`) and were *not* listed by BL8.

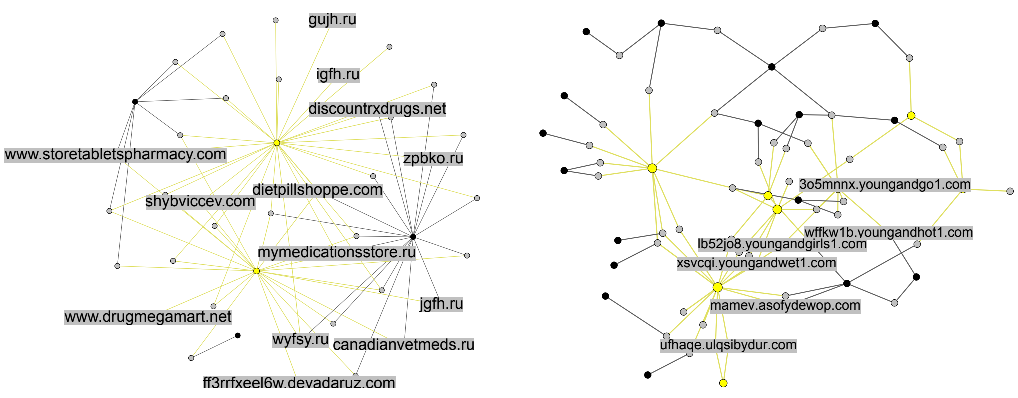
Another particularly interesting example is a group of 86 FQDNs (50 IP addresses, 28 ASes) (see Fig. II.28b). Most of the FQDNs have the format `<PREFIX>.youngand<WORD>1.com` where ⟨PREFIX⟩ was a random string and ⟨WORD⟩ an English word (e.g., `girls`). 37 of these FQDNs were found by blacklist BL8. Other sites hosted on the same IP addresses do not appear in any blacklists though (e.g., `ulqsibydur.com`). We suspect that these domains were used for a different purpose, and were therefore not seen by the Spam analyzers.

For DNS–DS2 we find 14 groups with 1399 FQDNs, of which 408 domains in four groups were indeed malicious. For example, Fig. II.29 shows a group of 85 malicious FQDNs which are hosted on 12 IP addresses in 8 ASes. Again, many sites map to only one IP address, and are still being detected. Note that the vast majority of non-malicious groups we found related to adult content offerings, which use a highly agile set of domain name aliases to attract more customers.

9.2.3.3 Other Findings

As the graph analysis module is decoupled from DNSMap, one can quickly query the system for a variety of different malware activity patterns, and thereby enable explorative, curiosity-driven network data analysis. In the following, we exemplify several interesting findings.

Some malware domains neither use an excessive number of IP addresses, nor do they share these addresses with other domains. As these sites often use redundant hosting, our



(a) “Pharma” group (b) “YoungAnd” group

Figure II.28: Two groups of malicious sites found in DNS-DS1. We removed parts of the revealed groups for better visualization and highlighted several illustrative examples of FQDNs. Black nodes represent IP addresses, yellow ones ASes, and gray ones FQDNs.

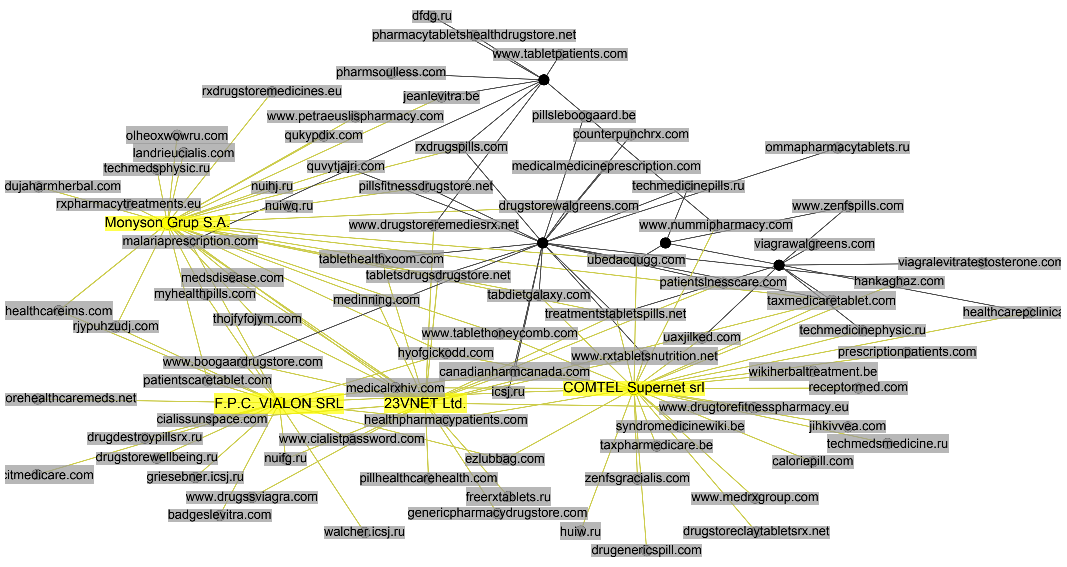


Figure II.29: A group of malicious sites found in DNS–DS2.

system can be tuned to detect them. We set ε=1 second and required that $\phi_1 \geq 1$, $\phi_2 \geq 5$, and $\phi_3 \geq 5$. This allowed us to find, e.g., the Phishing site `flickr.loginformm.org` immediately when it was requested the first time, 3.3 days into DNS–DS1[4]. Running this analysis for an entire day of DNS–DS1 yielded only 10 distinct FQDNs, of which three related to malware. Note that the strength of the system is here to *not* report a vast number of benign sites (each mapping to many IP addresses), as the system understood that these represent no significant changes. Therefore, we can use a highly sensitive configuration, without introducing many false positives.

Another group of Phishing sites was found in DNS–DS2. All domains followed the format `PREFIX.com.net`, where PREFIX was identical or similar to a well-known service name (e.g., `facebook`, `googele`). This underlines the importance of keeping a human analyst in the loop, who can differentiate this particular activity from a benign one (e.g., NTP, see §9.2.2).

[4] `http://www.phishtank.com/phish_detail.php?phish_id=1084355`

Our analysis of DNS–DS1 revealed the two domains `ssh.bl4ze.info` and `sw.maximum-irc.info`, which mapped to five IP addresses in five ASes. We found a malware binary analysis report from Feb. 2011 which stated that these domains hosted C&C services for a ZeuS botnet at that time[5]. Our analysis would have found this activity much earlier (Dec. 2010), and would therefore have enabled early countermeasures. We found several other domains used for botnet C&C, both in DNS–DS1 and DNS–DS2. For example, `ncbcyety2j4h5g.com` related to the Torpig malware and was detected despite the fact that it used only two IP addresses in two ASes[6].

As a final example, we report the detection of `ltbowyndzv.ce.ms` in DNS–DS2. We found an entry on Google Safebrowsing[7] in May 2013, stating that this site hosted malware within the last 90 days, i.e., 1.5 years after the data set was recorded. Our graph analysis module also reported `ckrmrku.ce.ms` in the same agile group, which was however not detected by Google.

9.3 Discussion

In our experiments we found various kinds of malicious activity, ranging from Fast-Flux (probably used for Botnet Command-and-Control), over Phishing and exploit sites, to domains used in Email spam. Even moderate agility patterns are detected, and many of them relate to malware activity. Our parameter settings force malware to reduce its agility patterns significantly for going undetected. However, this highly sensitive configuration comes at the cost of misclassifying certain benign activity as malicious. We argue that such activity *should* be reported though, as DNS analysis alone is not able to differentiate it from malicious services, when both show similar levels of agility. A targeted, in-depth analysis is *enabled* by the low number of misclassified domains, and *supported* by the structural representation of our approach (see §9.2.2).

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

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

Our approach bases on the high stability in DNS mappings of benign services (see §8.2). Popular sites require more hosting resources (i.e., IP addresses), and appear therefore more agile, and require more training data for being properly modeled. An interesting property of this essential modeling step results from the fact that sites which are requested often are better modeled than less popular ones. We emphasize that DNSMap by design reports *all* DNS mappings which involve a new IP address, and such activity is therefore *guaranteed*

[5] `http://www.exposedbotnets.com/2011/02/swmaximum-ircinfobotnet-hosted-in.html`
[6] `http://www.threatexpert.com/report.aspx?md5=6412ba5583756b80557020197981c401`
[7] `http://google.com/safebrowsing/diagnostic?site=ltbowyndzv.ce.ms`

to be detected if sufficiently many addresses (in our trials: 4-20/week) are being used. We can detect such moderate fast flux activity due to the thorough understanding of the activity of benign sites. Large services like Google may use many more IP addresses, but they use the same ones over time, and they do not change the FQDN patterns.

9.3.1 Limitations

The proposed system is heavily based on the concept of guilt by association, i.e., if *one* FQDN or IP in a graph component is found to be malicious, then *all* are considered malicious. The advantage of being able to reveal also malware activity which does not stand out by itself, comes with the drawback that also legitimate sites may wrongly be classified as malicious. In fact, in our experiments we occasionally found sites which appeared benign, but happened to map to IP addresses used by malware. We consider this tradeoff to be inherent to DNS-based analysis though, which is unable to observe the actual data transfer between IP addresses.

DNSMap stores information about every single IP address it observes, and therefore benefits to a certain extent from the relatively small size of the IPv4 address space. With the migration to IPv6, and the omission of the requirement to reuse IP addresses for multiple services, the number of observed addresses will likely increase vastly, and our system may have trouble in keeping track of all these IPBlocks. However, not all IP addresses are equally often seen, therefore a caching solution with a database backend might solve this problem.

Also note that the DNSMap representation of DNS activity is valid for a single vantage point only, and cannot be trivially transferred to other networks. This is because the mappings between FQDNs and IP addresses often depend on the geographical location of the requesting host, so to optimize the data transfer performance between server and client.

9.3.2 Evasion Strategies

In general, an adversary has two main options for evading our system: either escaping DNSMap's change detection, or going unnoticed w.r.t. the corresponding graph features, i.e., use only few FQDNs, IP addresses, and ASes per agile group. This comes at a cost, as fewer IP addresses being used per FQDN impacts the reliability of the malware service. Knysz et al. discuss fast-flux evasion strategies in [95], and derive models describing the relation between the number of online malware IP addresses and the availability of the corresponding malware sites. They consider a minimum number of 100 unique IP addresses per week and FQDN, which results (according to their model) in an average of 2.89 online IP addresses and a connection loss probability of 71.1%. Although this would already result in poor malware connectivity, this kind of activity is still far beyond the sensitivity limits of our system. In fact, we would have revealed any Fast-Flux activity which involves $\geq$ 20 IP addresses per week and *agile group* (as opposed to a single FQDN). Therefore, we can easily detect malware activity even when all the proposed evasion techniques are implemented, and the overall malware utility is already considered poor.

Therefore, the adversary might try the second option, i.e., avoid that malware activity results in DNS change events in the first place. All new IP addresses being used are reported always, therefore the challenge lies in using only IP addresses which are known to DNSMap, and use FQDNs which are similar (i.e., $DD < \Theta$) to the corresponding IPBlocks' cluster labels. Less than ϕ_1 new FQDN "families" on $< \phi_2$ IP addresses in $< \phi_3$ ASes can be introduced per epoch ε for going undetected. As shown in §7.2.1.1, our selection for Θ leaves only limited degrees of freedom, and forces the adversary to use stable patterns for constructing FQDNs which appear similar (e.g., by using a common suffix). This strategy can not be changed for at least ε, i.e., one week in our case.

In other words, the adversary is forced to use a less agile mapping procedure, which is the opposite of what was originally intended. This leaves significant time for other detection approaches (e.g., malware binary analysis) for revealing this pattern, and derive a corresponding blacklist entry (e.g., `*⟨SUFFIX⟩`). Furthermore, these restrictions have a severe impact on malware activity which requires flexibility in the choice of FQDNs for various reasons, e.g., Phishing FQDNs which should mimic the structure of the target site (e.g., `www.example-bank`**`k`**`.com`), or sites which aim at appearing benign (see, e.g., Fig. II.27).

9.4 Summary

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

In particular, we made the following main contributions:

1. We discussed the design of an analysis system which processes large amounts of DNS traffic data in real time. It produces valuable output already after two days of initial training, and continuously adapts over time without requiring a retraining phase. In contrast to most other approaches, the system does not require any prelabeled traffic data and no a prior whitelisting. As FQDN whitelists often contain many, unspecific entries as, e.g., `*.SUFFIX`, our system therefore allows more control about the actually ignored FQDNs. The set of analysis parameters is small, and can be intuitively tuned for new deployments.

2. The basis of our approach is a new methodology for assessing the level of agility of DNS FQDN-to-IP mappings. Our approach, called *DNSMap*, tracks the mappings between FQDNs and IP addresses, and stores them efficiently. In contrast to existing work [27, 10], we always consider the *entire* FQDN instead of only a suffix, and are therefore able to provide more specific results.

3. On top of DNSMap, we employ graph analysis for analyzing sets of suspicious DNS mappings, in order to address the distributed (CDN-like) nature of malicious networks. We proposed a set of graph analysis features, and demonstrated that even conservative settings reveal malware activity reliably.

4. Conceptually, DNS traffic analysis cannot provide an ultimate verdict of the maliciousness of domain names and IP addresses, without considering any actual payload exchanged with a site. Our approach enables almost instantaneous detection, so to enable such an analysis at the time when a malicious site is still active. Furthermore, it provides an intuitive way to assess the overall suspiciousness of groups of FQDNs and IP addresses, which allows the analyst to quickly focus on highly suspicious activity.

5. We use significantly less data than other approaches and our system is more sensitive to malware activity. It can be easily tuned to reveal different kinds of malware activity, and is therefore not restricted to Fast-Flux detection. Therefore, it is more versatile than previous work [130, 129].

Part III
Connection Analysis

Network-based malware detection is heavily complicated by the fact that malware developers introduce obfuscation and encryption techniques [126]. The analysis of such communication is difficult, as, in general, many traffic features do not contain information which exclusively relates to malicious activity. Malware mimics benign traffic by using the same protocols, and infected hosts typically engage in (malware-initiated) criminal communication, as well as in (user-initiated) benign one. Telling those types of communication apart is challenging, as the criminals behind the malware are free to introduce new measures which prevent successful analysis, or at least delay it long enough to make revenue.

Among the few traffic features which cannot be easily obfuscated are the identifiers of the communication endpoints, i.e., the IP addresses of source and destination of a packet. The analysis of connection graphs built from such information has been explored previously (see §4.2.3), and it was found that malicious sub-structures in such graphs can be revealed. However, these graphs are typically extremely large, and existing analysis procedures cannot cope with them in real time (see §4.2.2). Furthermore, Internet hosts which fell victim to criminal activity engage in *both* benign and malicious network communication. Connection graph analysis which considers these connections only, cannot differentiate these two types, and is prone to consider benign sites as part of a criminal plot, just because they were contacted by hosts which *also* contacted malicious sites.

The work presented in the following is driven by the idea of identifying the set of network locations which are normally contacted by a *specific* host. Each anomalous connection of each monitored host is further analyzed, and patterns of collaboration between monitored hosts are revealed. We evaluate the prototype implementation using traffic data from a network operator, and show that the size of the network graphs to be analyzed can be significantly reduced, and malware activity can be revealed reliably.

This work has appeared in the following publication:

- Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Oliver Jung. Locality matters: Reducing internet traffic graphs using location analysis. In *Proceedings of the Performance and Dependability Symposium (PDS) at the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, Budapest, Hungary, June 2013

CHAPTER **10**

Problem Definition

The premise of graph-based malware detection approaches is that Internet hosts which fell victim to the same Internet crime would contact the same hosts. For example, victims of a Phishing attack would contact the same servers, and therefore be *indirectly connected* to each other, via these servers. In case of, e.g., botnet C&C communication based on P2P protocols, the infected hosts would even connect directly. In general, the precise type of connections required by a specific malicious activity is unknown a priori. Nevertheless, there is often *some* degree of connectivity among and between victim hosts and malicious servers. We summarize all these different levels of connectivity as patterns of *collaboration* among a set of Internet hosts, which represent sub-structures in a graph representation of Internet traffic.

In accordance with the existing literature [88], we adopt the term "Traffic Dispersion Graph" (TDG) for describing connections between Internet hosts as graphs. Specifically, we consider *static* TDGs (as defined by [88]), which represent the aggregated connections over a certain time interval (i.e., an *epoch*). As discussed already in §4.2.3, the main practical problem of such approaches are the vast sizes of TDGs, which hinder the analysis of the underlying traffic information. As one does not know in advance which hosts are involved in malicious activities, it is necessary to consider as *many hosts* as possible in the graph, so to increase the chances of detection of collaboration patterns of a subset thereof. Furthermore, a single graph should represent a sufficiently *long period of time*, as malicious activity is, in general, not synchronous. I.e., although two hosts may be infected with the same malware instance, they may contact a malicious server at different moments in time, because of, e.g., simply the fact that the infected machines were not switched on simultaneously. Therefore, no collaboration patterns would appear in the TDG if the chosen time interval is chosen too short to contain the communication of both hosts. Finally, one cannot know in advance which protocols are being used by a specific type of malicious activity. Therefore, it is, in general, not possible to preprocess the TDG such that only a specific type of communication is included, as, e.g., only packets with payloads which contain a specific keyword. Consequently, the graph representation should be *independent* of the actual *payload*, but rather allow the detection of *any* specific type of collaboration, regardless of the communication protocol in use.

These three main requirements are illustrated in Fig. III.1. Any two requirements can be fulfilled rather trivially. For example, an analysis system can easily consider all packets of all monitored hosts, if the time window is being kept small. Likewise, it is possible to build graphs representing the communication of many hosts over long periods of time, if only highly specific packets are considered (e.g., only HTTP packets containing a certain keyword). Meeting all three requirements is significantly harder, and is addressed in the following.

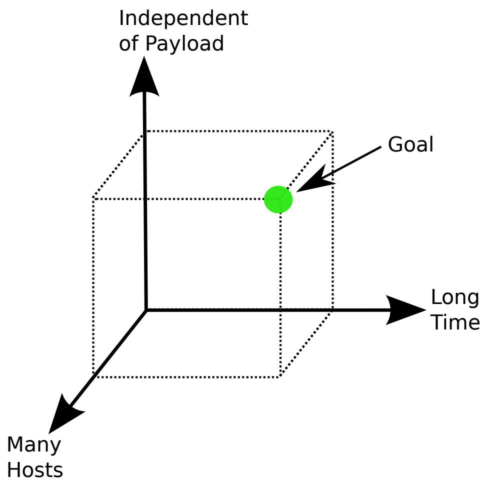


Figure III.1: Illustrating the problem of graph analysis of Internet connections.

Filtering is a standard data reduction technique. In the context of network anomaly detection, connections to a set of services, which are considered out of scope for the particular detection task, as, e.g., `www.google.com`, would often fall in this category. This concept is commonly called *whitelisting* and usually causes a significant reduction of the graph complexity, due to the high share of Internet traffic that these sites attract. This simple approach has significant drawbacks though: (i) The increasing usage of CDNs and cloud computing platforms make it hard to decide which IP address represents which service at which time. Any IP address can be used for a variety of services. Should therefore all connections to/from an IP address in, e.g., Google's AS be removed? Similarly, should we whitelist all connections to CDNs (e.g., Akamai) and cloud providers (e.g., Amazon EC2), which are also commonly used by large services? The more broad the filtering for a specific service is being configured, the more other services are unintentionally being removed together with it. (ii) This concept of service-based whitelisting is intrinsically a technique that assumes a uniform level of confidence in the decision to filter a certain site, independent from the individual host contacting this site. It is therefore quite possible that a connection to a popular IP address *is* in fact *anomalous* for a *particular* host.

In summary, this problem poses a number of challenges: (i) Highly popular "good" sites should be removed early, to avoid false positives and unnecessarily high load on the graph analysis algorithms (many are NP-complete); (ii) The packets' payload cannot be considered for preprocessing; (iii) The graph reduction technique needs to scale to thousands of monitored hosts, as usually no evident structure can be identified from the analysis of only few hosts; And (iv), the graph should represent connections during a time window wide enough to catch asynchronous, but still related, communication of multiple hosts.

Basic Concept

We are interested in deriving *reduced* TDGs while retaining the *suspicious* traffic patterns of individual monitored hosts. That is, instead of focusing only, e.g., on connections which involve specific packet payloads, we aim at deriving a system which is able to understand which connections are anomalous for each of the monitored hosts, based on these hosts' previous activity. Any such preprocessing must be scalable, and therefore cannot employ any information that is expensive to extract, as, e.g., using deep packet inspection. Similar to the presented DNS analysis approach (see §II), we aim at exploiting the inherent *agility* of malicious Internet activity. In contrast to benign services, malicious services are hosted on a *changing* set of server resources, and the set of hosts which are contacted by a victim of

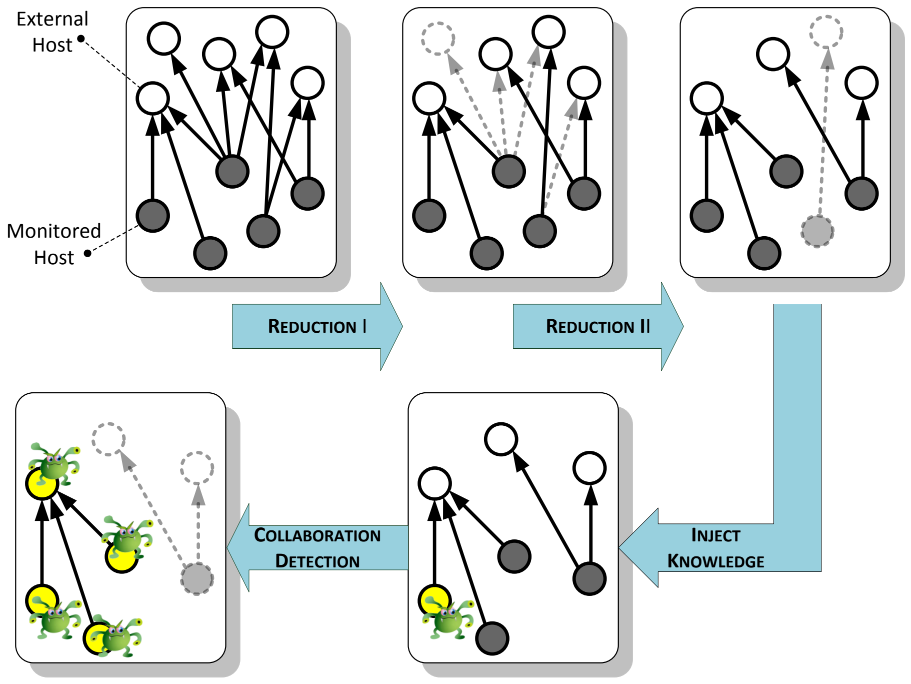


Figure III.2: The four main steps of the connection analysis approach. Starting from a complete TDG which contains all traffic data, we derive a reduced TDG which can be analyzed efficiently.

Internet crime is therefore expected to vary over time. Hence, we consider only such agile connections for inclusion in the TDG to be analyzed. The envisioned approach involves a series of four main steps which are illustrated in Fig. III.2. We discuss these steps in the following.

Reduction I: We observe that there exist specific distributions of the (external) contacts for each monitored host, since users tend to consume the same Internet services over time. However, each host has a specific distribution of contacts, which does not allow to use a one-fits-all pattern to distinguish normal from anomalous or suspicious connections. Therefore, we focus on a per-host traffic description, which is compact in terms of memory consumption and adaptive to the observed traffic data per host, while still being able to represent typical traffic patterns with high accuracy. Any deviations to a particular host's previous activity are retained, while connections which are known to be typical for this host are not further considered, and the corresponding edges are deleted from the TDG. Note that this may disconnect single nodes, which are then also removed from the graph.

Reduction II: Following the removal of individual nodes and edges, we consider the topological structure of the remaining graph for further reduction. This is based on the observation that certain sub-structures are not relevant for finding collaborating malicious hosts. For example, nodes representing external hosts which have a node degree of one, are destinations of only one suspicious connection, and are therefore unlikely to represent a popular malicious server.

Inject Knowledge: We rely on external information for identifying a set of malicious hosts which serve as a *seed* for the detection of malicious collaboration. Such information can be retrieved, e.g., from malware binary analysis (see §4), from public blacklists, or from other detection approaches like the DNS analysis presented in §II.

Collaboration Detection: Finally, we use graph-based community detection approaches to identify the hosts which collaborate with the malicious seed hosts, and which therefore are likely malicious themselves.

CHAPTER 11

Analysis Approach

In the context of malware detection, a standard approach for reducing the complexity of the Internet traffic data to be analyzed is the omission of any data which refers to well-known, benign services like `www.google.com`. However, as discussed in §8.2, due to the widespread use of CDNs and cloud computing platforms, there exists in general no bijective mapping between service names and IP addresses. In other words, a single IP address usually hosts a multitude of different services, and the mapping between a service's name and an IP address rotates quickly depending on the time of day and the current load. As TDGs represent communications between hosts which are identified by IP addresses, service-based whitelisting requires us to understand precisely which IP address corresponds to which service at which time. Alternatively, one could whitelist entire ASes. This is however a rather coarse measure, and potentially removes many "interesting" connections together with the popular ones.

Instead of completely ignoring large IP ranges or entire ASes, just because a single popular site is occasionally being hosted there, we envision a technique that is able to remove those edges from a connection graph that represent activity that is considered normal *for a particular host*. We propose to use *Points of Presence* (PoPs) as a trade-off between IP address and AS information. PoPs are physical locations where a set of network hosts accumulate, e.g., the location of a particular data center or the city in which an Internet customer lives. Considering a PoP instead of an IP address has the advantage that PoPs absorb a certain amount of the fluctuation caused by CDNs. Instead of trying to untangle the mappings between service names and IP addresses, we rather present a methodology that directly infers the PoPs which a host contacts regularly, and which are therefore not considered anomalous *for this particular host*.

Fig. III.3 shows the considered scenario: a network operator observes connections between monitored hosts and groups of external hosts at a number of different PoPs. For being able to apply "Reduction I" (as introduced in the previous chapter), we aim at deriving an automatic procedure which finds those PoPs that are regularly contacted by a particular host. Consequently, we would then disregard connections of this host to hosts at these PoPs, and thereby achieve a reduction of the TDG.

Certainly, this is still a rather coarse measure, as we exclusively use IP address information for deciding about the unusualness of a connection. Therefore, we do not claim that we would find each and every suspicious connection, but rather hope to find *enough* ones to identify the malicious activity. To illustrate this, consider the following (simplified) example: assume that a new malware infects a large number of monitored hosts, and starts contacting IP addresses at a number of different PoPs, e.g., to download additional data or report back to a control server. Some monitored hosts would probably have contacted some of these PoPs before, and therefore this activity would seem normal for them. However, some other hosts would have never contacted a subset of these PoPs, and therefore

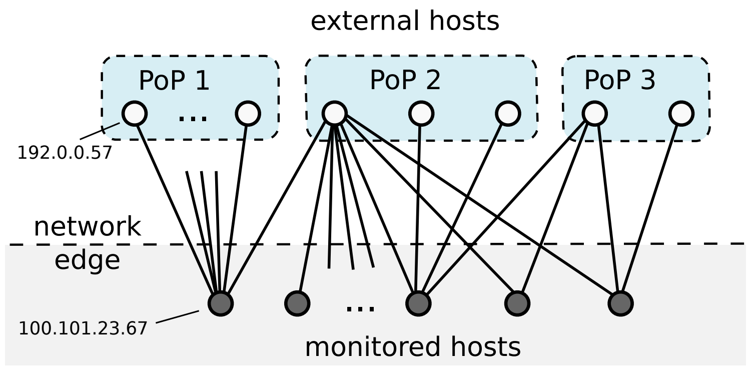


Figure III.3: The basic scenario we consider: a network operator monitors the connections between a set of internal hosts and a (much larger) set of external hosts at one or more edge routers.

these connections would stand out and are consequently considered in the reduced TDG. By knowing that at least one of the involved hosts is malicious, we can find the community of hosts which are involved in this malicious activity. Upon this first alert, a network operator can then conduct an analysis that focuses on the newly found contacted IP addresses, and eventually find the specific type of malicious activity.

Limitations This approach is obviously not perfect though. An immediate argument that speaks against the usage of PoPs is that whitelisting a PoP (for a particular monitored host) is equivalent to ignoring *all* services hosted at the same PoP. This is obviously true, but we argue that we cannot do much better than that. In general, we cannot learn from connection information only, if a new IP address at a known PoP represents a known, “good” service, or a new, “bad” one. At the same time, we cannot use significantly more data and still capture the activity of all hosts. However, we combine per-host profiles and graph analysis. By that, we do observe connections of hosts which never consumed *any* service at this PoP, which makes the PoP appear in the graph, and makes it accessible to the subsequent structural analysis of connection patterns.

There are other constraints of more practical nature. The mapping between an IP address and its PoP can be efficiently found using a variety of (commercial) databases. For our system, the *granularity* of these databases is highly important, as it defines the achievable resolution of the envisioned per-host traffic description. Note that the *accuracy* of the location information for a particular IP is less crucial for our approach. As we aim at detecting *changes* in the set of PoPs being contacted by a particular host, the true location of a PoP is of secondary importance.

We preliminary evaluate the granularity of such databases in as follows. For this work, we use MaxMind’s freely available GeoCityLite database[1]. It contains 144,335 PoP locations with registered IP addresses, which we show in Fig. III.4. Note that no map is underlying this plot. Apparently the coverage spans almost all populated areas, and the granularity of the data is sufficiently high even to recognize coastlines.

Next, we evaluate the advantage of using PoPs instead of AS information. Fig. III.5 shows the cumulative distribution of the number of PoPs per AS. Out of 33,406 ASes, 40% use more than one PoP location, around 20% use three or more, and some reach thousands of locations. The ASes with most locations are mostly ISPs (e.g., one AS belonging to AT&T maps to 6,455 PoPs). Hence, PoPs consistently differentiate Internet hosts better than AS information. Furthermore, the quality of our representation further improves with better location resolution in the database.

[1]`http://dev.maxmind.com/geoip/`

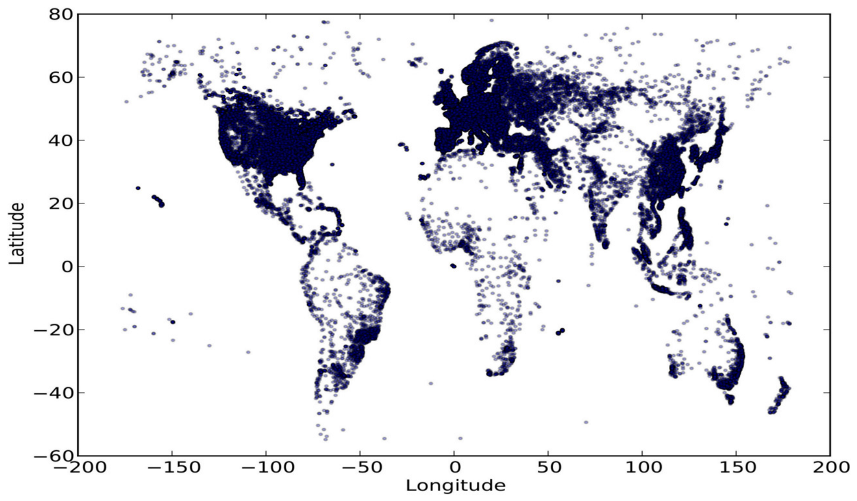


Figure III.4: Locations with registered IP addresses, based on Maxmind's GeoLite City database from Dec. 2011.

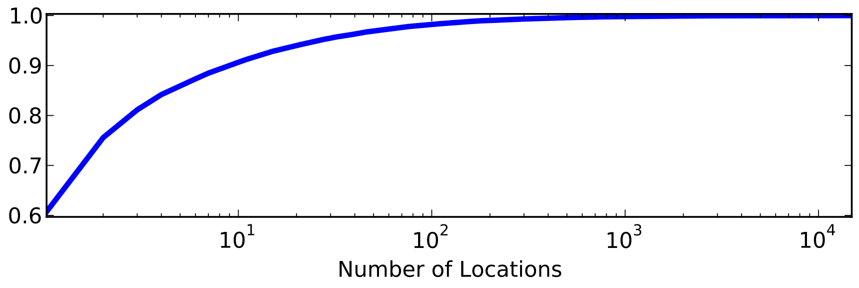


Figure III.5: CDF of number of PoP locations per AS.

11.1 Methodology

The main challenge consists in characterizing the normal activity of each monitored host. Fig. III.6 illustrates this basic idea: PoPs that a monitored host contacted often in the past and which are therefore in "high density areas" are "normal", PoPs in "low density areas" are suspicious. The figure also illustrates a fundamental problem: using a fixed grid for defining the areas is imprecise. Rather than describing each possible area with the same resolution, we want to concentrate it on high density regions, and thereby provide a more accurate approximation of the hosts' activity.

Our goal is the efficient modeling of the distribution of PoPs which were contacted by a specific monitored host in a certain epoch ε. We consider in the following two-dimensional distributions of PoP locations (i.e., longitude and latitude). We propose a technique that compresses these data, by approximating these distributions. The approach is based on *order statistics* and leverages their ability to capture the approximate shape of a distribution without requiring any a priori assumptions. We found that this is a fundamental feature in our context, as individual distributions vary heavily depending on the services a monitored host consumes, as well as with the time of day. In the remainder of this section, we first revisit order statistics of unidimensional samples and sketch the basic idea. We map this then to the two-dimensional case and show how we can derive scores for individual connections.

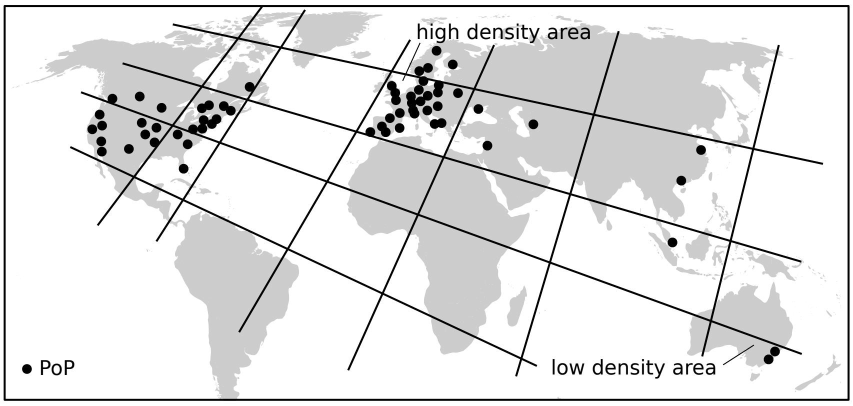


Figure III.6: Examples of PoPs contacted by a monitored host and a possible (though suboptimal) partitioning in geographical regions. For the sake of better visualization we assume here that all locations have been contacted equally often.

11.1.1 Preliminaries: Order Statistics

Order statistics are an established, parameter-free, and simple technique to describe any empiric unidimensional distribution $\mathcal{D}$. Given an ordered set $\mathcal{P}$ of N samples $\mathcal{P} := \{p_1 \leq p_2 \leq \cdots \leq p_N\}$, then p_ρ is the ρ-th order statistic and ρ is called the *rank* of p_ρ. We call $k = \rho/N \in [0,1]$ the *relative rank* and η_k the (sample) *k-quantile*. Order statistics provide a non-parametric way to compute estimate quantiles η_k of the distribution $\mathcal{D}$ underlying the set of samples, without making any assumptions about the nature of $\mathcal{D}$. Note that there are several different ways to define a quantile [86]. We are particularly interested in the non-interpolating definition, where the estimate is an element of $\mathcal{P}$. These quantiles can be derived from a data sample in $\mathcal{O}(N \cdot \log N)$ time for sorting, plus $\mathcal{O}(K \cdot \log N)$ for deriving K quantiles.

The relation of the distances between two relative ranks and their corresponding quantiles relates to the *relative density* of $\mathcal{D}$ in this range. Therefore, K quantiles implicitly provide $K-1$ *estimates for the Probability Density Function* (PDF) of $\mathcal{D}$. This enables basic comparisons of currently observed samples with a previously observed set of quantiles: if the value falls in a dense region of the PDF, similar values were observed when computing the quantiles. However, simply choosing a fixed, large set of relative ranks for computing quantiles for whatever set of samples, with the expectation that this would provide a highly precise approximation of the PDF, is misleading. For few samples, the estimation of the PDF can be only a coarse one, with a high degree of uncertainty in the accuracy of the computed quantiles. Therefore, the set of relative ranks needs to be data-adaptive, i.e., it must depend on the available number of samples.

To address this issue, we propose to compute for each set of relative ranks the minimum number of samples such that the *confidence bounds* of the corresponding quantiles do not overlap. Hence, the less samples are available, the less quantiles are being estimated, making the concept adaptive to N. The Clopper-Pearson confidence interval [40] requires as input the number of samples N, a confidence level $\gamma \in (0,1)$, and k, to compute the interval $\left[\rho_k^-, \rho_k^+\right]$, where ρ_k^- and ρ_k^+ are the ranks of the lower and the upper bound for η_k. Starting from a maximal set $\mathcal{K}_{\text{MAX}}$, we derive different combinations $\mathcal{K}_m \subset \mathcal{K}_{\text{MAX}}$. By sorting the elements of $\mathcal{K}_m$ such that $\{k_1 < k_2 < \cdots < k_i < \dots\}$ we find those $\mathcal{K}_m$ for which every $k_i \in \mathcal{K}_m : \rho_{k_i}^- > \rho_{k_{(i-1)}}^+$ and $\rho_{k_i}^+ < \rho_{k_{(i+1)}}^-$. I.e., we find combinations $\mathcal{K}_m$ for which the confidence bounds of the quantiles to be computed do not overlap.

The Clopper-Pearson interval for a quantile estimate can be efficiently computed as

$$
\begin{aligned}
\rho_k^+ &= \left(1 - \text{BetaInv}\left(\frac{1-\gamma}{2}, N-\rho, \rho+1\right)\right) \cdot N \\
\rho_k^- &= \left(1 - \text{BetaInv}\left(1 - \frac{1-\gamma}{2}, N-\rho+1, \rho\right)\right) \cdot N
\end{aligned}
\tag{11.1}
$$

where BetaInv is the inverse Beta distribution [34]. The (extended) algorithm for selecting relative ranks based on the number of samples is shown in Table 11.1, and will be further discussed in §11.1.4.

11.1.2 ϕ-α Quantiles

Given the two-dimensional nature of our problem, i.e., distributions of locations defined by longitude and latitude of the contacted PoPs, we present in the following a solution that applies the previously introduced ideas. Computing exact two-dimensional quantiles is computationally expensive [42] and does not scale to the envisioned scenario with thousands of Internet hosts and millions of connections. Therefore we rely on an approximate solution, which has the advantage of having similar computational complexity as determining unidimensional quantiles. The idea is based on the concept of *dominance*.

Definition 3 *A point p dominates a point q if p has higher rank in both dimensions x and y: $p \succ q := \rho_x(p) > \rho_x(q) \wedge \rho_y(p) > \rho_y(q)$.*

Definition 4 *Given a set $\mathcal{P}$ of N points in the plane. The dominance set of a point $p \in \mathcal{P}$ is defined by $dom(p) := \{q \in \mathcal{P} \mid p \succ q\}$. Its weight is defined as $w(dom(p)) := \sum_{q \in dom(p)} |q|$, where $|q|$ is the number of occurrences of q.*

Note that these definitions are analogous to unidimensional quantiles. Cormode et al. [42] suggest a fast approach to compute so-called *ϕ-α quantiles*, which are sets of point estimates for multidimensional quantiles. We replicate in the following the basic formulation, and refer the reader to their paper for further details. We developed an efficient algorithm for computing ϕ-α quantiles which we discuss (including a proof of its correctness) in the appendix (see §A). The computational complexity of our solution scales like $\mathcal{O}(N \cdot \log N)$ where N is the number of unique points.

Definition 5 *The ϕ-Dominance $\phi(p)$ of p is $w(dom(p))/N$. The set $\mathcal{P}_\phi$ contains all points $p \in \mathcal{P}$ where $\phi(p) \leq \phi$.*

Definition 6 *The α-Dominance α_p of p is $\rho_y(p)/(\rho_x(p) + \rho_y(p))$. The set $\mathcal{P}_\alpha$ contains all points $p \in \mathcal{P}$ where $\alpha_p \leq \alpha$.*

Definition 7 *The **ϕ-α quantile** $\eta_{\phi,\alpha}$ is the point $p \in \mathcal{P}_\phi \cap \mathcal{P}_\alpha$ with the maximum ρ_y. In case many points share the same maximum ρ_y, the one with the maximum ρ_x is selected.*

Example: In Fig. III.7 we plot the $\phi - \alpha$ quantiles for N=100 synthetic samples from a uniform-uniform distribution, using $\phi \in \Phi := \{0.1, 0.5, 0.9\}$ and $\alpha \in \mathcal{A} := \{0.4, 0.5, 0.6\}$. The solid lines connect point estimators with the same ϕ, and the dashed lines the ones with the same α, respectively. On the lower left, we highlight the region that the quantile $\eta_{\phi=0.1,\alpha=0.5}$ dominates. The points in that region comprise the set $dom(\eta_{(0.1,0.5)})$. Including $\eta_{(0.1,0.5)}$ itself there are nine points in this region, therefore $w(dom(\eta_{(0.1,0.5)})) = 9$. The quantile's ϕ-value is therefore $9/100 = 0.09$, which is the closest approximation to the requested $\phi = 0.1$ that the algorithm was able to find using $\alpha = 0.5$.

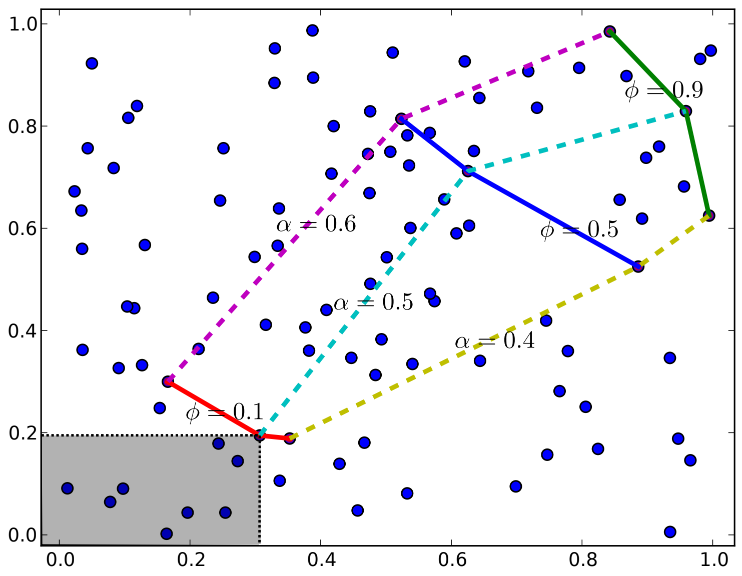


Figure III.7: $\phi - \alpha$ quantiles example for 100 random samples from a uniform-uniform distribution.

11.1.3 2D Density

For characterizing the density of the two-dimensional distribution we adopt the ideas introduced in §11.1.1. Given the sorted sets $\Phi := \{\phi_1 \leq \phi_2 \cdots\}$ and $\mathcal{A} := \{\alpha_1 \leq \alpha_2 \cdots\}$, we can construct polygons defined by the points $\{\eta_{\phi_i,\alpha_j}, \eta_{\phi_i,\alpha_{j+1}}, \eta_{\phi_{i+1},\alpha_{j+1}}, \eta_{\phi_{i+1},\alpha_j}\}$. We refer to such polygons in the following as $poly_{\phi_i,\alpha_j}$. The area covered by these polygons is equivalent to the distances between unidimensional quantiles, and therefore relates to the PDF of the two-dimensional distribution. However, the interpretation is not as straightforward as before: due to the construction of $\phi - \alpha$ quantiles, the individually covered areas differ even for the uniform-uniform distribution, as $\mathcal{P}_\phi$ grows exponentially with linearly increasing ϕ.

We solve this by relating the derived PDF to a *null model*, in which each point is equally probable. Specifically, we compare the areas of the observed $\phi - \alpha$ polygons to the corresponding areas of the uniform-uniform distribution. Consider a uniform-uniform distribution of an infinite number of samples. We can compute the theoretical values (x, y) for every $\eta_{\phi,\alpha} = (x, y)$ by solving the following equations, derived from the definition of $\phi - \alpha$ quantiles in [42]:

$$x = \sqrt{\phi/\alpha - \phi}; \quad y = \phi/x$$

This allows for computing the area covered by each polygon $\overline{poly}_{\phi_i,\alpha_j}$. Let δ be the result of subtracting the area of $\overline{poly}_{\phi_i,\alpha_j}$ of the uniform-uniform distribution from the actual result of the corresponding $poly_{\phi_i,\alpha_j}$, extracted from a set of samples from an unknown distribution which we normalized to [0,1]. We derive the final score $\sigma \in [0, 1]$ associated with this polygon as follows.

$$\sigma := \begin{cases} (1 + \sqrt{-\delta^2 + 2 \cdot \delta})/2 & \text{for } \delta > 0 \\ (1 - \sqrt{-\delta^2 - 2 \cdot \delta})/2 & \text{for } \delta < 0 \\ 0.5 & \text{for } \delta = 0 \end{cases} \tag{11.2}$$

By definition, a score of 0.5 represents an activity that is "neutral", i.e., neither normal nor anomalous, as it is completely in line with the uniform-uniform distribution. Scores

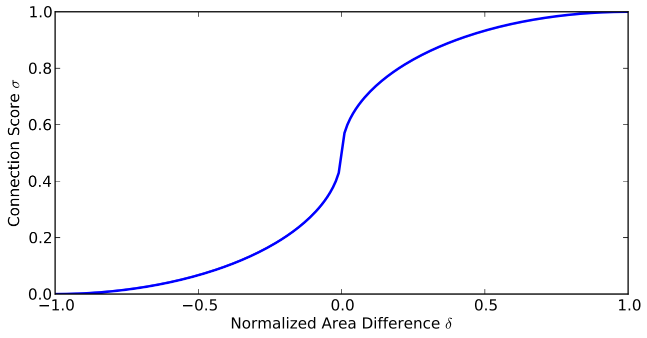


Figure III.8: Scoring function.

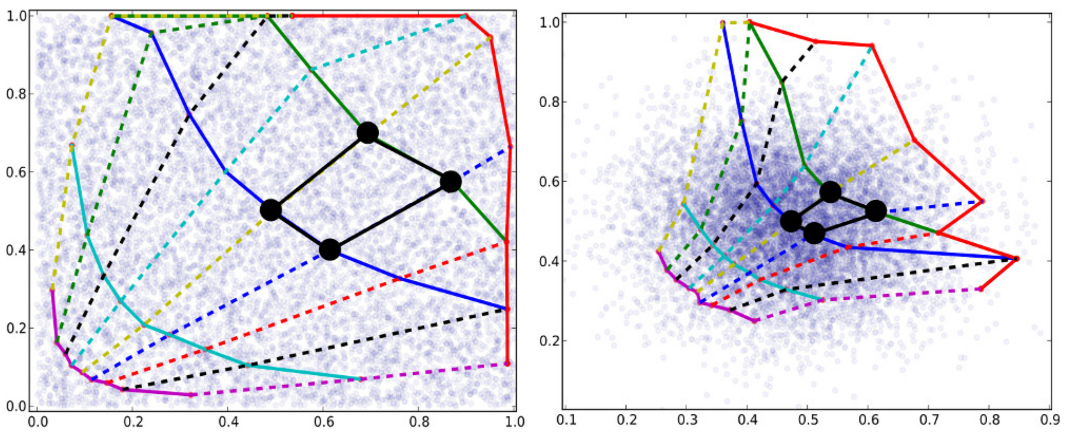


Figure III.9: Comparison of the $\phi - \alpha$ quantiles of 10,000 uniform-uniform (left) and 10,000 normal-normal samples (right).

lower than 0.5 represent high-density regions of the two-dimensional distribution, i.e., the lower a score the more "normal" the observed activity. Conversely, scores higher than 0.5 represent more anomalous observations. σ is defined such to magnify deviations of δ from zero, so that the minimum and maximum scores (0.0, and 1.0) are approached quickly (see Fig. III.8).

Example: Fig. III.9 shows the $\phi - \alpha$ quantiles of two synthetic sets of samples drawn from a uniform-uniform (*u*-*u*), and from a normal-normal (*n*-*n*) distribution (with $\text{MEAN} = \text{STDDEV} = 1$), respectively, normalized to [0,1]. For both sets we compute the identical $\phi - \alpha$ quantiles, where $\Phi := \{1, 5, 25, 50, 90\}\,\%$ and $\mathcal{A} := \{10, 20, \cdots, 90\}\,\%$. The areas covered by the polygons of *u*-*u* indicate, by definition, uniform probability, i.e., in a set of samples that produces these quantiles, no event is more likely than another. For the *n*-*n* distribution, this does of course not hold true, and consequently the polygons representing the *identical* combinations of ϕ and α are differently shaped and have a different size. We highlight one example for such a polygon in the figure. Notice the much smaller size of the right-hand polygon, which indicates the much higher density of the *n*-*n* distribution in this area, leading to a small σ (< 0.5).

11.1.4 Confidence Bounds

Next, we derive a two-dimensional equivalent to the confidence intervals introduced in §11.1.1. Recall that we aim at deriving sets $\Phi_m \subset \Phi_{\text{MAX}}$ such that the confidence intervals of all $\phi_k \in \Phi_m$ do not overlap given a certain confidence level α and the number

Algorithm 11.1: MinNumSamples

```
input  : $\Phi$, $\gamma$
output: $N$
1  $N \leftarrow 1$;
2  while True do
3      bounds $\leftarrow$ List();
4      foreach $\phi$ in $\Phi$ do
5          $\rho^-$,$\rho^+ \leftarrow$ ConfidenceBounds ($\phi$, $\gamma$, $N$);
6          bounds.append(($\rho^-$,$\rho^+$));
7      end
8      if not Overlap (bounds) then
9          return $N$;
10     end
11     $N \leftarrow N$ +1;
12 end
```

$\Phi_m[\%]$	90%	95%	99%
1,50,90	38	45	71
1,**30**,50,90	126	166	282
1,30,50,**70**,90	135	176	287
1,**20**,30,50,70,90	355	486	802
1,20,30,50,70,**80**,90	399	552	905
1,20,30,50,**60**,70,80,90	491	675	1142

Table 9: Minimum number of samples for various Φ_m, with $\gamma = \{90\%, 95\%, 99\%\}$. The ϕ_k printed in bold highlight the changes to the preceding Φ_m.

of samples N. Note, however, that for a two-dimensional distribution there are, in general, infinitely many different selections for $\rho_x(p)$ and $\rho_y(p)$ such that $\phi(p) < \phi_k$. Therefore, we select a number of choices for $\phi_{k,x}$ from the interval $[\phi_k, 1]$, and compute the corresponding confidence bounds $\rho^+_{k,x}$ and $\rho^-_{k,x}$ for each, using Eq. (11.1). In other words, we sweep over the range of possible combinations for ρ_x and ρ_y, following the same concept as implemented by using different selections for α, to derive confidence bounds for different quantile estimates with the same ϕ. The more intermediate steps in $[\kappa, 1]$ we take, the more accurate is the representation. In our experiments, we used 10 equidistant choices for $\phi_{k,x}$. Then, we compute the means of all $\rho^+_{k,x}$ and all $\rho^-_{k,x}$ and accept these two values as the average confidence bound for ϕ_k.

For finding the sets Φ_m to be used by our system, we run the simple iterative algorithm MinNumSamples (see Algorithm 11.1) for a number of candidate sets Φ_c. Specifically, we select all combinations $\Phi_c \subset \Phi_{\text{MAX}}$ for which $|\Phi_c| \geq 3$ and use MinNumSamples to retrieve the minimum number of samples required so that the confidence bounds do not overlap. For each set cardinality in $\{3, 4, \ldots, |\Phi_{\text{MAX}}|\}$ we select the corresponding Φ_c which yielded the minimum number $M(\Phi_c)$ of required samples. The selected candidates Φ_m are then stored in a lookup table which is further used for finding the set of $\phi - \alpha$ quantiles to be computed given a certain number of available samples (i.e., locations contacted by a single monitored host in a certain epoch). Note that this entire procedure needs to performed only once, as it does not depend on the particular samples themselves, but only on their number. Table 9 shows some examples for various Φ_m and different confidence levels γ.

CHAPTER **12**

System Design and Architecture

Putting together the puzzle pieces we developed so far, we can now design a system which enables us to derive reduced TDGs for further analysis. The system builds on the following main steps:

1. Processing of traffic data and extraction of the set of PoPs P_s contacted per each monitored host s. We aggregate this information over an evaluation epoch ε and track the number of times each PoP $\in P_s$ has been contacted by s (i.e., the weight of each PoP). The sum of all weights gives the number of location samples $N^{\varepsilon}(s)$ of s in ε. We then repeat the following steps for each host s.

2. We identify the sets Φ_m where $M(\Phi_m) \leq N^{\varepsilon}(s)$ and select the set Φ_X with maximum $M(\Phi_m)$ (see §11.1.4).

3. Computation of the $\phi - \alpha$ quantiles of P_s using their weights and Φ_X (see §11.1.2).

4. Construction of polygons $\text{poly}_{\phi_i,\alpha_j}$ and computation of a score σ for each, according to Eq. (11.2).

5. The set of polygons and associated scores represent the *activity profile* of a host s in ε. We store these profiles for future reference of the activity of a particular host in this epoch of time.

Let $\mathcal{SRC}$ be the set of all monitored hosts (sources) and $\mathcal{DST}$ be the set of all external hosts (destinations) in a given traffic data set. The activity profile of a monitored source host $s \in \mathcal{SRC}$ in an epoch ε_i relates to the destination addresses $d \in \mathcal{DST}$ which were contacted by s, or more precisely, to their geographical locations. Each observed packet of s gives one sample of the distribution of locations contacted by s. Note that we require *stable identifiers* for the monitored source hosts, in order to be able to relate their network activity to the stored profiles. We address this practical problem in §13.

Each profile relates to a particular epoch and comprises a set of polygons, which are defined by $\phi - \alpha$ quantiles, and where each polygon has an associated score σ. Locations which were contacted often receive a higher weight, and thereby influence the ϕ-α quantiles. Therefore, popular areas are automatically framed tightly by the corresponding polygons, and receive low scores. Note that for each host there exist many such profiles, namely one for each epoch ε in which the host was active.

In the following, we show how we employ these profiles to build an anomaly detection system that adapts to diverse, dynamically changing host activity, and continuously updates each host's set of profiles. First, we show how to use these profiles to compute scores for individual end-to-end connections, to derive an intermediate reduced TDG by removing connections with score lower than the detection threshold Θ. Second, we discuss

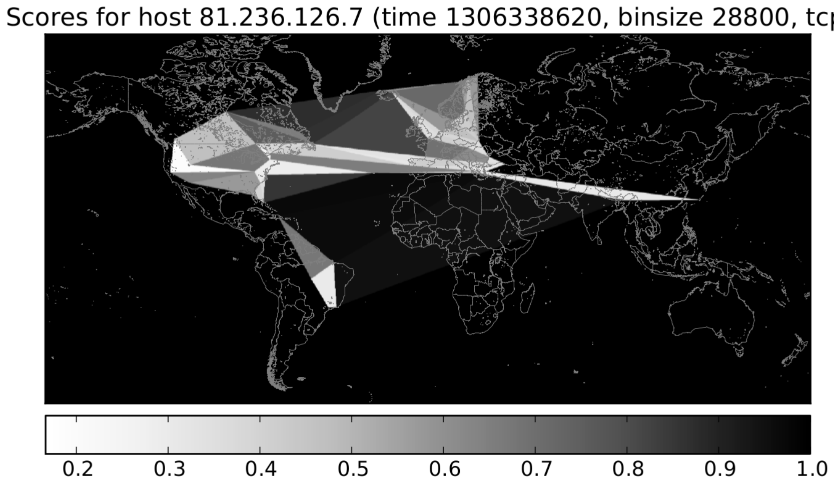

Scores for host 81.236.126.7 (time 1306338620, binsize 28800, tcp)
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Figure III.10: Example for the regions found by computing ϕ-α quantiles for the TCP connections of one host in a time window of eight hours.

a set of simple graph analysis techniques to find groups of suspicious hosts, for deriving a further reduced TDG. And finally, we describe a complete anomaly detection system that implements these approaches

12.1 Reduction I: Connection Scoring

Scoring individual connections between a monitored and an external host enables us to apply the first reduction (i.e., "Reduction I" in Fig. III.2) to the original TDG. Consider a connection from source s to destination d with location (d_{lon}, d_{lat}) (i.e., longitude and latitude). In order to find the score for this connection w.r.t. to a *single* activity profile, we need to find out in which polygon of this profile (d_{lon}, d_{lat}) falls. The problem of deciding whether a polygon contains a certain point is known as "Point in Polygon". Multiple solutions with a computational complexity of $\mathcal{O}(n)$ exist, where n is the number of polygon edges [79]. Note that in our approach always $n \leq 4$.

Once the polygon is found, we immediately learn the corresponding score (see §11.1.3). In case no polygon is found, we assign the maximum score 1.0. For polygons that are collapsed to a single point, and therefore represent a maximum density region with only one location, we assign a score of zero. Note that in this case the representation is equivalent to the exact one, i.e., it is certain that exactly this location was highly popular for the considered source s, and hence represents normal activity.

Example: Figure III.10 visualizes the profile extracted for a host from a real network, for one epoch ε_i. The $\phi - \alpha$ polygons are clearly visible, and are colored according to the score that a connection of that host to a destination inside a certain polygon would cause. E.g., all black regions in the figure represent highly unusual locations for this host. Note that for the sake of demonstration we picked a "scattered" profile here. We observed that typically the covered regions are significantly smaller, but are less suitable for illustrating the concept. In fact, many polygons typically collapse to single points, or describe small, high-density regions. We show this analytically in §13.

The connection score w.r.t. to *all* activity profiles of s is then found by computing the mean of the ψ smallest scores. This is motivated by the findings of other anomaly detection approaches (see, e.g., [48]): the network activity of a host depends on the time of day and varies on weekday, weekends, and festivity days. Furthermore, it is influenced by special events like sports broadcasts. Instead of trying to take all that into account,

General configuration	
ε	- Length of epoch (seconds)
Φ_{MAX}	- Maximal set of ϕ_κ, i.e., $\{1, 5, 10, \ldots, 90, 95, 99\}$ %
$\mathcal{A}$	- Fixed set of α_i, i.e., $\{10, 20, \cdots, 90\}$ %
γ	- Confidence level for MINNUMSAMPLES (0,1)
Reduction I: Connection Scoring	
Θ	- Scoring threshold $[0, 1]$
ψ	- Number of profiles to consider [1,∞)
L	- Maximum number of profiles per user to keep [1,∞)
Reduction II: Graph Analysis	
θ_1	- Minimum destination degree threshold [0,∞)
θ_2	- Community threshold [0,∞)

Table 10: System Parameters

e.g., by comparing the activity on festivity days only to weekends, we rather search for epochs in which the host activity was most similar to the current one. Any activity that is *still* significantly different, is considered an anomaly. Note that this search can be aborted as soon as the running mean is lower than the detection threshold Θ, which significantly reduces the computation time. Also note that this has to be done only *once* per epoch ε, i.e., independent of the number of packets between s and d.

All connections with a score lower than the detection threshold Θ are considered normal for the particular monitored host, and are therefore ignored. The remaining connections define the (intermediate) reduced TDG.

12.2 Reduction II: Graph Analysis

The reduced TDG is a weighted digraph $\mathcal{G} := \{V, E\}$, where $V \subset (\mathcal{SRC} \cup \mathcal{DST})$ and E is the set of (suspicious) connections between them. The ultimate goal of the system is to derive groups of connected hosts, as, e.g., botnets. Therefore, we aim to reveal popular destinations that receive a significant number of suspicious, i.e., highly scored, connections. For this reason, we apply a second reduction step (i.e., “Reduction II” in Fig. III.2), based on TDG’s structure.

We process the graph as follows: first, we remove all $d \in \mathcal{DST}$ with an in-degree less than θ_1. Then we remove all sources that are disconnected now, i.e., all s with a degree of zero. Finally we use the Louvain method [28] to find the best partition of $\mathcal{G}$, i.e., the set of communities that maximizes the modularity metric (see §4). We remove all those communities from $\mathcal{G}$ which do not contain a single destination node with an in-degree $\geq \theta_2$. In our experiments, these two steps proved to be highly efficient even for low threshold settings (e.g., $\theta_1 = \theta_2 = 3$), as they removed many connections that appeared individually suspicious for only single hosts, which is not representing the type of large-scale anomalies we are hunting for. Note that this last step is similar to other approaches [116, 43]. However, instead of sampling the graph, or whitelisting a certain set of connections, we take advantage of the reduced representation, that takes individual host activity into account.

12.3 System Design

The system requires as input tuples of aggregated connection information in the form

$\langle$`timestamp`$\rangle$, $\langle$`sourceIP`$\rangle$, $\langle$`destinationIP`$\rangle$, $\langle$`protocol`$\rangle$, $\langle$`packetCount`$\rangle$

This information is typically exported by standard network equipment like routers, e.g., as NetFlow or IPFIX messages.

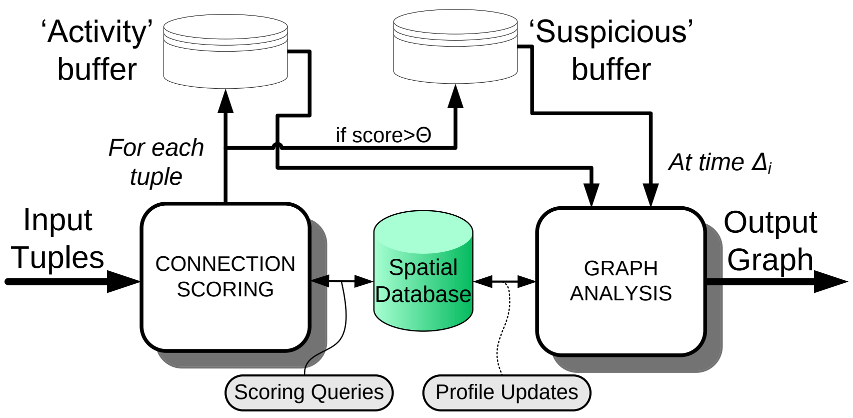


Figure III.11: System Overview.

The overall concept is illustrated in Fig. III.11 and the system's parameters are summarized in Table 10. We use a spatial database [143] for storing and querying the hosts' activity profiles (i.e., the polygons and the associated scores). The "activity" and "suspicious" buffers are initially empty. Upon receipt of a new tuple, the system resolves the geographical location (i.e., longitude and latitude) of *destinationIP*, and computes a score σ according to the procedure described in §12.1. Connections with a score $> \Theta$ are additionally kept in the "suspicious" buffer. Then it stores the location together with rest of the tuple in the "activity" buffer, and continues with the next tuple. At the end of each epoch ε_i, the system applies the simple graph analysis procedure "Reduction II" on the connections in the "suspicious" buffer, to find groups of individually suspicious hosts (see §12.2).

Finally, the system computes for each observed *sourceIP* s the $\phi - \alpha$ quantiles of the contacted locations in the "activity" buffer. Specifically, it requests the quantiles with $\alpha \in \mathcal{A}$ (which is a system parameter) and $\phi \in \Phi_m$, such that $|\Phi_m|$ is maximal while still guaranteeing that the corresponding confidence bounds do not overlap (as given by the procedure described in §11.1.4, and depending on the number of packets per source host). The newly derived profiles are then stored in the database. At the same time the oldest set of profiles is removed, to maintain the invariant that the length of the sliding window holding the reference profiles is equal to L. Then both buffers are emptied, and the system continues with processing the next epoch. For each epoch ε_i, the system outputs the *reduced TDG* of *individually* suspicious connections.

CHAPTER **13**

Experimental Evaluation

For the following experiments we use a data set that is based on a network trace captured at a router that connects around 700 residential broadband customers to the Internet. The trace includes the RADIUS accounting information [139], which allows us to relate the dynamically changing customer IP addresses to their fixed customer identities. For each IP packet in the trace, the customer IP address was replaced with a 32-bit (pseudonymous) value, computed from hashing the customer identity with a secret key, which was not available to us and therefore protected the customers' true identities. The total length of the trace is three weeks, of which we extracted tuples (⟨`timestamp`⟩, ⟨`srcIP`⟩, ⟨`dstIP`⟩, [`tcp`, `udp`], ⟨`packetCount`⟩). We used only the uplink traffic, i.e., the packets sent by the customers, and compute separate profiles for TCP and UDP connections. In the following, we refer to the first two weeks of these data as data set CONN–DS1 and to the last week as CONN–DS2.

For all following experiments we set $\mathcal{A} := \{10, 20, \cdots, 90\}\,\%$, $\gamma = 95\%$, and $\Phi_{\text{MAX}} := \{1, 5, 10, \ldots, 90, 95, 99\}\,\%$. Note that this implicitly eliminates outliers in the distribution of locations per monitored host s (<1%, >99%). We resolved the geographical locations for IP addresses using MaxMind's database introduced in §11. All analysis tools are implemented in Python.

13.1 Validation Tests

Accuracy As a first validation of the proposed system, we evaluate how "tightly" the location profiles describe the monitored hosts' activity. We set $\varepsilon = 28,800$ seconds and extracted all host profiles for CONN–DS1. For each monitored host s, we then randomly selected 10,000 locations from the MaxMind database, and computed TCP and UDP scores for them. As the locations were selected randomly, the system was expected to detect many deviations from the previously derived profiles, and the scores should be high. In case the derived profiles were too "loose", the scores would be expected to be low. The distribution of the mean scores per host s are shown in Fig. III.12. The scores are high in general, with more UDP than TCP connections being reported as "normal". Note that this implicitly shows also that location does matter, i.e., that the hosts' network activity is not completely scattered across the world. Our approach makes no assumptions about *which* locations these are though, but rather derives them automatically.

Next, we evaluate the opposite: how well do the extracted profiles represent the actual network activity? We compute scores for each connection in CONN–DS1, using the activity profiles as references, and thereby test CONN–DS1 against itself. If the extracted profiles are accurate, the scores should therefore be low. Again we compute the mean score per host s and show the distribution of all mean scores in Fig. III.13. We find that 97% of all mean TCP scores are lower than 0.003. For UDP, 90% are lower than 0.03. This indicates

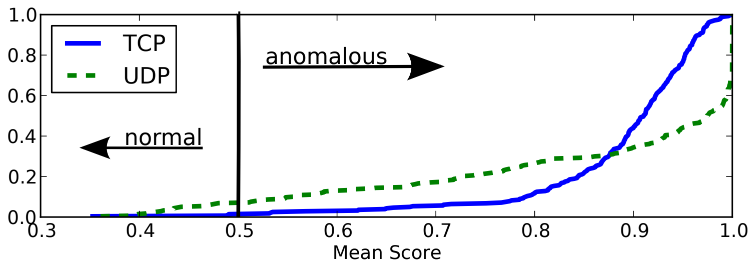


Figure III.12: CDF of mean scores for 10,000 random locations per monitored host.

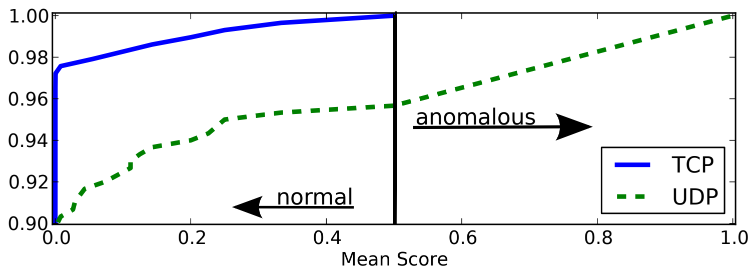


Figure III.13: CDF of mean scores for testing CONN–DS1 against itself.

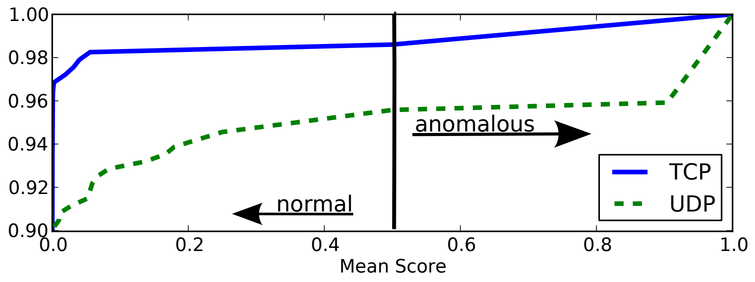


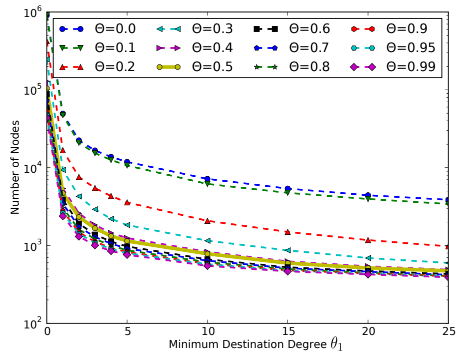
Figure III.14: CDF of mean scores for testing CONN–DS2 against CONN–DS1.

that many scores are in fact zero, i.e., the system automatically found single, highly popular locations for a host, that caused the quantiles defining the polygon to collapse to a single point. We conclude that our system is able to extract typical per-host activity.

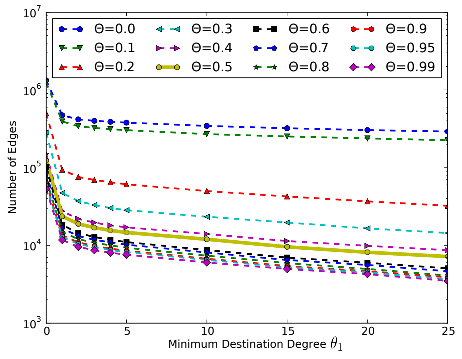
As a final accuracy evaluation, we test the persistence of the activity profiles, i.e., we evaluate if previous activity is indeed indicative for future one. We again set ε=28,800 seconds, load the profiles extracted from CONN–DS1, and run the system with CONN–DS2 as input. Fig. III.14 shows the results, which are similar to the previous test, and indicate that there is indeed stability in the locations contacted by a host. That is, monitored hosts do contact the same PoPs over time, despite the widespread use of CDNs.

Graph Complexity Reduction Recall that our main objective is to reduce the overall complexity of the connection graph (i.e., the number of nodes and edges), while keeping the connections to anomalous locations. Therefore, we evaluate the influence of applying different settings for Θ (on the edge weights), and θ_1 (on the node in-degree) on the resulting graph.

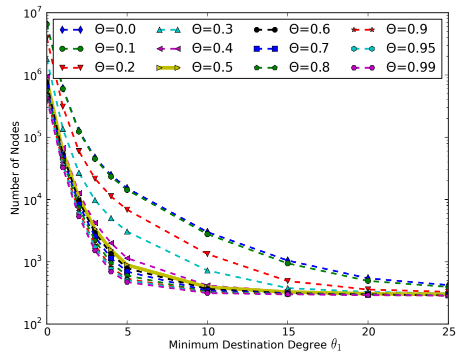
First, we evaluate the impact that our method has on complexity of the graph when using the same data set as input that was used for generating the activity profiles. Note that this is similar to the results shown in Fig. III.13, but considers also the impact of the structural analysis using θ_1, as well as the effects of the changing set of available activity profiles as the analysis goes on. We set $\varepsilon = 28,800$ and extract all host profiles for CONN–DS1. Then, we apply the scoring procedure to the same CONN–DS1, using the activity profiles



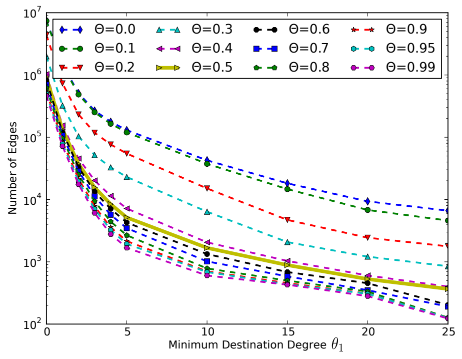
(a) TCP Nodes



(b) TCP Edges



(c) UDP Nodes

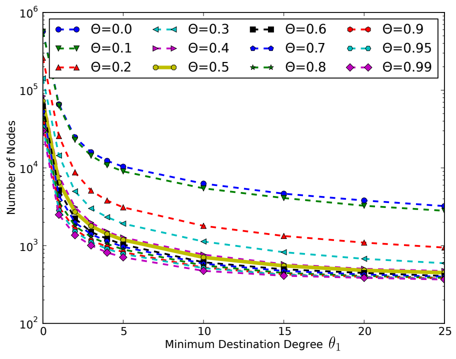


(d) UDP Edges

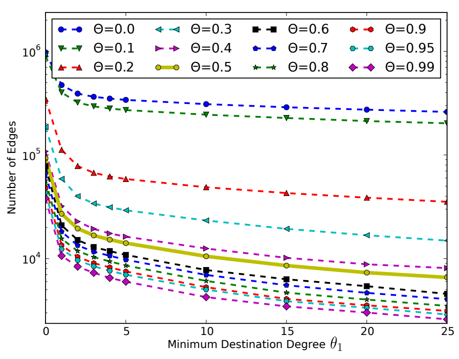
Figure III.15: Evaluating CONN–DS1 against itself; $\varepsilon = 28,800$.

as references. We expect that the output graph is significantly smaller than the original connection graph, as we use the same data for training and testing. Fig. III.13 shows the resulting graph complexity for different choices of Θ and θ_1, both for TCP and UDP. Note that we plot the complexity of the *entire* connection graph for these two weeks of traffic. Also note that we remove disconnected nodes from the resulting reduced TDG. The data points for $\Theta = 0, \theta_1 = 0$ show the original size of the graph, i.e., around 1 million nodes and edges for TCP, and around 10 million for UDP, respectively. For TCP, our technique is able to reduce the graph's complexity in terms of number of nodes by up to three orders of magnitude, even for very moderate setting for Θ and θ_1 (e.g., $\Theta = 0.5$, $\theta_1 = 10$). The results for UDP are even better, with a complexity reduction of four orders of magnitude, for even more conservative parameter settings (e.g., $\Theta = 0.5$, $\theta_1 = 5$).

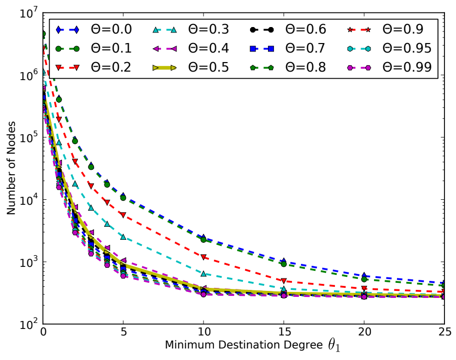
Next, we repeat the experiment for data set CONN–DS2, using the activity profiles extracted from CONN–DS1. Fig. III.16 shows the resulting graph complexity in terms of number of nodes and edges, for both TCP and UDP. Again, we observe a reduction of three to four orders of magnitude in terms of the number of nodes, for similar parameters. The number of edges is being reduced by two orders of magnitude for TCP, and by three orders of magnitude for UDP, respectively. Even settings of $\Theta \leq 0.5$, i.e., below the designed demarcation between normal and anomalous (see Eq. (11.2)) have a significant effect. Note that setting θ_1 to very high values has little impact on the graph complexity. This indicates that low values should be preferred, which enable the detection of suspicious destinations d even when they are considered suspicious for only few monitored hosts s.



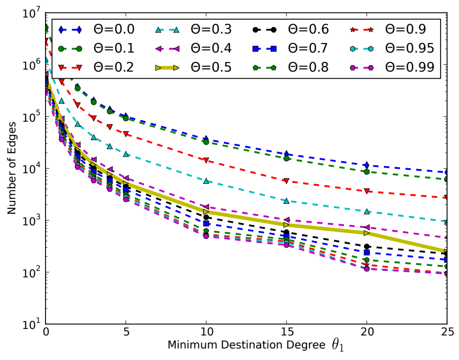
(a) TCP Nodes



(b) TCP Edges



(c) UDP Nodes



(d) UDP Edges

Figure III.16: Evaluating CONN–DS2 against CONN–DS1; $\varepsilon = 28,800$; $L = 3 \cdot 14$ (i.e., 14 days with three profiles per day).

13.2 Malware Detection

We conducted two experiments for evaluating the system's ability to detect malicious activity. First, we show that we are able to heavily reduce the TDG's complexity while still retaining malicious traffic patterns. Then, we discuss the system's detection performance after injecting partial knowledge about one malicious host and running an algorithm for collaboration detection. These two steps complete the system description as shown in Fig. III.2 (i.e., "Inject Knowledge" and "Collaboration Detection").

13.2.1 Botnet Emulation

We used the CORE network emulator[1] to run 15 instances of the popular *Zeus* malware[2]. The bots connect to their Command-and-Control infrastructure at regular time intervals, to download new commands and report back their latest activity. We selected 15 out of around 300 monitored hosts that were active during a time period of one hour, and overlaid the malware traffic to the first hour of CONN–DS2. Furthermore, we assumed the following scenario: *Fast-Flux* botnets usually contact several different proxies (which are themselves infected hosts) to reach their hidden C&C server. We assume three different locations for these proxies (namely Moscow, Rome, and Vienna), and let the bots pick one of these con-

[1] http://cs.itd.nrl.navy.mil/work/core/

[2] http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99

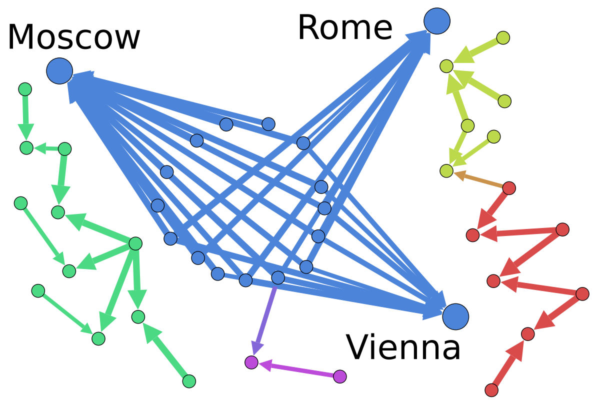


Figure III.17: Botnet emulation results: the reduced TDG as output by the system.

nections randomly whenever they want to send something. Note that given our trace from a European operator, these locations are tough choices for our algorithm, as also legitimate traffic is being sent there. We loaded the profiles extracted from CONN–DS1, and set $\Theta = 0.5$ and $\theta_1 = 2$. Note that these settings are very conservative. Fig. III.17 shows the system's output: the three C&C servers as well as all 15 "bots" are visible in the graph, and the structure of the botnet emerges clearly. In addition, a few benign connections are considered suspicious, and therefore appear in the graph. Note, however, that all of them would, e.g., be removed when setting $\theta_1 = 4$, without any impact on the botnet subgraph, and without increasing Θ.

13.2.2 Dye-Pumping

For our second experiment we revisit the ideas presented by Coskun et al. (see [43] and our discussion in §4.2.3) and relate them to the so-called "confessions of a botnet operator", as recently reported in the news[3]. A botmaster decided to publish some details about his daily "business", and answered questions online. Apparently he employs the Tor network[4] for exchanging C&C messages. We decided to evaluate the performance of our system in a scenario based on this assumption. We retrieved the publicly available list of Tor nodes, and randomly picked 100 out of ∼3,000. Similarly as in the previous experiment, we then picked 30 random monitored hosts in our trace. We assume that each bot would on average send only 1 packet per minute, each to a random out of the 100 Tor nodes, so to operate in a stealthy manner. We overlaid this communication to the network trace, and ran our detection system for one hour to produce a graph of suspicious connections. We selected a random bot node as the seed node for our implementation of the dye-pumping algorithm.

We repeated the experiment with different thresholds Θ, trying to find the mutual contacts graph of bots, and evaluated the detection accuracy in terms of precision and recall. Fig. III.18 clearly shows that both precision and recall are significantly better for the reduced TDG, than for the not reduced one (i.e., when $\Theta = 0$), at all settings for Θ. The precision is lower for higher thresholds as such aggressive removal of graph edges also removes malicious connections. Note that we achieve perfect precision for Θ around 0.5 (with a recall of 90%), and that even very aggressive filtering (with $\Theta \geq 0.8$) does still leave enough malicious connections in the graph to detect almost all bots with few false positives.

[3] `http://h-online.com/-1574453`
[4] `https://www.torproject.org`

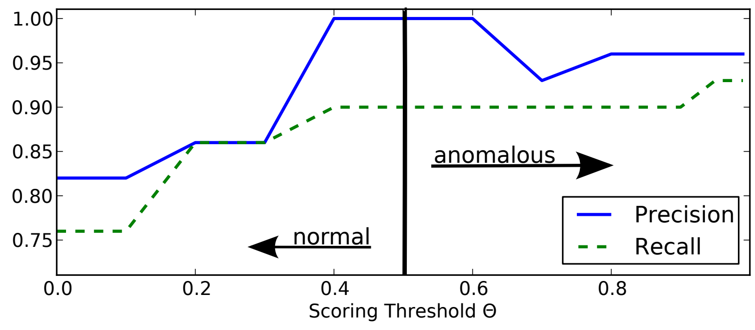


Figure III.18: Dye-Pumping results.

13.3 Discussion and Limitations

The system can reliably detect large-scale deviations from the reference profiles. However, due the limited traffic information we employ, certain types of anomalies can never be as accurately identified as with, e.g., signature-based systems. A flash crowd for example, would most likely be indistinguishable from a massive botnet. Also, we emphasize that we do not claim to detect *all* individual suspicious connections. In particular, our system is unable to detect suspicious connections of monitored hosts which *always* contact a vast number of different locations (i.e., powerusers). The value of the presented approach is therefore clearly in *enabling* structural analysis of network connections and raising a first alert when a change in the *collective* traffic patterns is observed, that should then be subject to further investigation. We are convinced that thereby the topological information our system provides in the form of a graph of suspicious links, each with its individual score, is invaluable.

However, large networks with many thousand monitored hosts may require more aggressive graph reduction than we demonstrated here. This can be achieved by parameter tuning. Obviously, smaller ε cause less data to be represented per TDG. Likewise, higher Θ result in more individual connections to be removed. For detecting large-scale malware outbreaks, tuning of the graph analysis parameters θ_1 and θ_2 should be considered. Larger values of these setting cause the system to remove connections which are not considered individually suspicious *and* are directed to a popular destination. Furthermore, the analysis of such graph communities using more advanced techniques like [116, 43, 68] is an interesting direction of future research.

Note that the system's ψ parameter represents the sensitivity to connection agility. The higher it is set, the more profiles are considered for computing a connection score, and the longer it takes until new, persistent activity patterns are finally considered "normal" for a particular host. Conversely, lower settings of ψ cause this new activity to be reported initially, but accept it as new, "normal" pattern rather quickly, and therefore exclude it in the reduced TDG. Ultimately, the parameter settings depend on the specifics of the activity which should be detected. In this thesis, we focus on the detection of highly agile connection patterns and therefore prefer low settings for ψ, which we further discuss in §IV.

Our system strictly requires stable identifiers for the monitored hosts. We realize that this information is often difficult to retrieve, but we do not consider it a fundamental handicap. In networks where client IP addresses are frequently reassigned, one could, as an alternative to RADIUS information, use the IMSI in mobile networks [48], or the MAC address in LANs.

13.4 Summary

We presented a method to assess the degree of anomaly of Internet connections solely based on the analysis of the geographical locations of IP addresses that a particular host contacts over time. We developed a profiling tool that is robust to changes in traffic volume, and which does not assume any predefined malware communication model. This is in contrast to existing systems (see §4.2.3), which are, e.g., limited to detecting malicious communication which uses specific protocols or focus on revealing particular, predefined C&C topologies. Our system is designed for monitoring large (ISP-size) networks, and the main processing routine for extracting reference profiles scales like $\mathcal{O}(n \log n)$ where n is the number of unique locations contacted per host and time bin. The evaluation using a trace from a live network showed that large-scale anomalies can be detected already using very conservative threshold settings. In two experiments we verified that injected botnet traffic can be reliably detected, while keeping the output graph of suspicious connections extremely compact. In particular, we achieved the automatic omission of connections to highly popular, benign services without having to whitelist certain IP ranges manually. In our experiments we observed excellent detection results for the injected botnet traffic with perfect precision (1.0) and high recall (0.9), for adequate threshold settings.

Our specific contributions are the following:

1. We studied the activity of Internet users w.r.t. the geographical locations (i.e., PoPs) they contact. We discussed the usage of two-dimensional quantiles ($\phi - \alpha$ quantiles) for representing the per-host profiles and showed their ability to reliably characterize "normal" networking activity.

2. We showed how to adaptively generate $\phi - \alpha$ quantiles in order to cope with the variance in the available data. This makes the approach scalable to potentially thousands of monitored hosts and adaptive to natural Internet traffic variations.

3. We defined a scoring framework for quantifying the degree of anomaly for end-to-end Internet connections based on per-host profiles. Being based on the previously derived $\phi - \alpha$ quantiles, the scoring procedure is highly efficient and scales like $K \cdot \mathcal{O}(N)$ where N is the number of points per derived polygon (in our case: $N \leq 4$) and K is the number of polygons per monitored host and time interval.

4. We integrated the approach into a malware detection system, and evaluated its performance using a real-world network trace.

Part IV
Joint Analysis and Final Remarks

In this thesis, two complementary network-based malware detection approaches have been proposed. The DNS analysis presented in §II yields highly accurate results, and is able to identify hosts which serve malicious content in real-time. The analysis of Traffic Dispersion Graphs (TDGs) presented in §III considers *all* types of communication between Internet hosts and is therefore not restricted to specific protocols.

In the following, we discuss the *joint* analysis of network traffic data, using both approaches at the same time. In particular, we demonstrate how the DNS results can be used to seed the analysis of the reduced TDG derived in §III, in order to find groups of hosts which relate to a particular type of Internet crime. For this purpose, we use a data set which contains both TCP and DNS information of 1.7 million monitored hosts over a period of 15 days.

Chapter 14

Joint Analysis

The analysis of DNS traffic provides agile groups of FQDNs and IP addresses which are involved in malware activity. As soon as a monitored host establishes a connection to one such IP address, this connection appears in the TCP data set. Therefore, we can use these IP addresses as *seeds* for the TCP connection analysis to find additional malicious communication in the reduced TDG. In the following, we directly apply the previously developed techniques and discuss our experimental results. First, we provide an overview of the experimental setup. Second, we briefly discuss the individual results of the DNS analysis (§14.1.1) and the TCP connections analysis (§14.1.2). Finally, we present an example for joint analysis in detail (§14.1.3) and discuss the merits and the limitations of our approach in §14.2.

14.1 Experimental Evaluation

For our experiments, we have two separate data sets (DNS–DS3 and CONN–DS3) from the same monitored network available, which connects ∼1.7 million monitored hosts to the Internet. Both data sets represent the same period of 15 days, from 2013/04/15 to 2013/04/30. Data set DNS–DS3 contains DNS mappings (see §7.2), while data set CONN–DS3 contains TCP connection information (see §13). The decision about which part of which data set to use for training and evaluating our system is non-trivial, and depends on the systems' parametrizations. Our configuration is illustrated in Fig. IV.1 and is further discussed in the following.

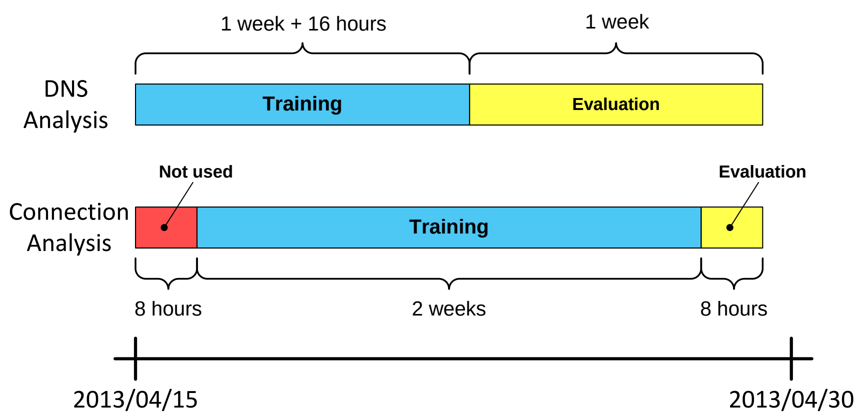


Figure IV.1: Usage of data sets for joint analysis.

We begin with the parametrization of the connection analysis approach and refer the reader to Table 10 on page 105 for an overview of the parameters. Note that we use in the following ε_{DNS} and $\varepsilon_{\text{CONN}}$ instead of ε, to distinguish the corresponding evaluation epochs. Likewise, we use Θ_{DNS} and Θ_{CONN} instead of Θ, to distinguish the corresponding detection thresholds.

For the connection analysis, we again set $\varepsilon_{\text{CONN}} = 28,800$ seconds (i.e., eight hours). As in §13, we set $L = 14 \cdot 3$ to maintain a sliding training window of 14 days (i.e., three ε_{CONN} per day). Therefore, the first reduced TDG is produced after 14 days and eight hours into the data set. However, these last eight hours fall into night (i.e., 00:00h – 08:00h) and contain only low traffic activity. We therefore consider the following eight-hour-epoch for our evaluation, as shown in Fig. IV.1.

For the DNS analysis approach, we consider all activity in the last week of our data set DNS–DS3, i.e., ε_{DNS}=7 days. Recall that the detection of malicious, agile DNS activity benefits from longer evaluation periods, as this allows us to collect more evidence of malicious activity. In particular, this enables the detection of IP addresses which host many different malicious services over time, and which therefore probably belong to a malware platform. As in §9, we set $\Theta_{\text{DNS}} = 0.35$ and all other parameters as shown in Table 2 on page 65.

Implementation Improvements The large volume of traffic information contained in our data sets required further improvements to the prototype implementation. While the existing DNS analysis prototype could be used without modification, the connection analysis implementation had to be improved. The main challenge was the storage of the vast number of activity profiles which are computed by our approach. Assuming that each of the 1.7 million monitored hosts would be active in each considered epoch $\varepsilon_{\text{CONN}}$ (i.e., eight hours) over the entire training length (i.e., two weeks), an upper bound for the number for the number of activity profiles is $1.7 \cdot 10^6 \cdot 3 \cdot 14 = 71.4$ million. Recall that each of these profiles contains multiple polygons. These profiles would not fit into the main memory of the machine we used for our experiments, and therefore we employed the PostGIS spatial database[1] for storing the profiles to disk. The database schema is shown in Appendix B. These improvements enabled us to process our data set in acceptable time (see §14.1.2).

Note that the lookup time for finding the polygon which contains a particular point (see §12.1) can be further improved by using an adequate data structure for storing the set of polygons. In particular, *Quadtrees* [61] may be considered. Instead of testing each polygon (of a single activity profile of a particular host) for point containment, one can quickly find the subset of candidate polygons using such tree structures, and test only them. However, these improvements are out of scope for this thesis.

14.1.1 DNS Analysis Results

In order to capture both “Fast-Flux” and “Malicious Hosting” activity we consider all agile groups (AGs) with ($\phi_1 > 1$ **and** $\phi_2 > 50$ **and** $\phi_3 > 5$) **or** ($\phi_1 > 50$ **and** $\phi_2 > 7$ **and** $\phi_3 > 5$). Using a slightly extended whitelist (created using the procedure discussed in §9.2.2) with 31 entries, we retrieve 16 agile groups. By manual inspection, we found that four out of these 16 groups do indeed represent malicious activity and show them in detail in Table 11. Note again that each of these malicious agile groups would have been detected using a much lower analysis sensitivity (i.e., higher limits for ϕ_1, ϕ_2, ϕ_3, e.g., $\phi_3 > 12$). Despite the sensitive configuration, the absolute number of false positives is remarkably low and amounted to only 1,203 false positive FQDNs (in 12 groups) out of 6.6 Million FQDNs in ε_{DNS}, while 99 malicious FQDNs were found in total.

[1]`http://postgis.net`

	Number of IP addresses	Number of ASes	FQDNs
AG1	526	177	serv4.cloudstoreservice.ru, bano4eva.com, tguniverse.com, techinformationgate.com, hecked-by-brain-krebs.biz, 32v235235n645645435.org
AG2	68	13	com-businesstimesblog.net, losebellyfat-now1.com, www.askmenow.com, com-the-financial-news.net, topwaystoloseweightquick.com, com-standartdaily.net, howicloseweightfastwithoutexercise.com, apps.facfbook.com, finance-reports.abc15news.net, ... (66 more)
AG3	66	44	kbitdsdt.com, wuhttkdi.biz
AG4	60	40	www.datinglalibta.ru, datingtisuvle.ru, ns1.ns01fonofni.ru, www.datinglafomvu.ru, www.datingsabipgu.ru, www.datingdugiksi.ru, www.datingtalugpi.ru, www.datingmerikdo.ru, aortnaa.ru, ... (7 more)

Table 11: Malicious agile groups found in DNS–DS3.

For evaluating our results, we again used the blacklists from Table 7 on page 82, various online services (e.g., `www.virustotal.com`), as well as manual Google searches. Note again that, due to our whitelisting procedure described in §9.2.2, we were *not* required to manually check all 1,203 false positive FQDNs. Rather, after the evaluation of a few tens of them, the corresponding agile groups fell apart, and the leftovers were removed by the graph analysis procedure, based on the configured limits for ϕ_1, ϕ_2, ϕ_3.

The evidence of malicious activity we found differed for each agile group, and underlines the importance of letting a human expert assess the results. The FQDNs of AG1 were partially listed in blacklist BL1 (cf. Table 7). Several FQDNs of AG2 were found by online services[2]. The FQDN `kbitdsdt.com` of AG3 was found to have been registered with 24 other FQDNs following the same naming scheme (i.e., eight random letters) on the same day, which were not present in our data set, but were found by others to partially represent malicious activity[3]. Finally, the FQDNs of AG4 were used in Spam emails and were partially listed in blacklist BL8 (cf. Table 7).

Recall that our analysis approach works independently of the number of queries and the number of unique monitored hosts querying the FQDNs. While this is an advantage of our system in general, our results do not immediately reveal the impact of a particular malicious activity on a monitored host population. Therefore, for the detected malicious groups we manually checked how many monitored hosts queried at least one FQDN in a group during the evaluation period of the connection analysis approach (i.e., eight hours). For AG1, we found two monitored hosts, for AG2 we found 18 monitored hosts, for AG3 we found zero monitored hosts, and for AG4 we found three monitored hosts.

Overall, the number of monitored hosts querying the identified malicious FQDNs was low in the considered evaluation period of eight hours. From the viewpoint of a traffic analyst, this is unfortunate as it is hard to identify a dominant group activity behavior given that few samples. Therefore, the detection of further malicious activity therefore poses a significant challenge. In the following, we focus on the malicious agile group which attracted the most monitored hosts, i.e., AG2, and use the IP addresses in this group as seeds for the connection analysis.

[2]E.g., `https://www.virustotal.com/en/domain/howicloseweightfastwithoutexercise.com/information/`

[3]`https://www.virustotal.com/en/ip-address/96.9.160.68/information/`

14.1.2 Connection Analysis Results

For the connection analysis, we first processed the entire training period of our data set CONN–DS3. Despite the large volume of our data set and the requirement to store the computed activity profiles to the PostGIS database (i.e., to disk), our prototype implementation required less than two weeks of time for processing the two weeks of training data. The total volume of activity profiles of 1.7 million monitored hosts amounted to only ~ 60 GB of disk space. This demonstrates that the system is practically usable for monitoring even large networks.

Subsequently, we configured the system for processing the eight-hours evaluation period, i.e., we set the parameters for the two graph reduction steps (see §10 and Table 10). Given the vast size of our data set, we set $\Theta_{\text{CONN}} = 0.9$ in order to focus on highly suspicious connections only. Furthermore, we set $\psi = 3$, i.e., we considered the activity profiles which yielded the lowest three scores for computing the final score for a connection of a particular host. Finally, due to the small number of monitored hosts querying the FQDNs in AG2, we set $\theta_1 = 1$ and $\theta_2 = 2$. In other words, we remove only graph communities which contain nodes representing external hosts which *all* have an in-degree of one. Given the low number of querying hosts, high in-degrees are not to be expected, and higher settings for θ_1, θ_2 would likely remove the corresponding connections from the reduced TDG. Note that it would certainly be much easier to find large communities of hosts involved in malicious activity, as (slightly) higher settings for θ_1, θ_2 are expected to have a significant impact on the complexity of the reduced TDG and would remove many small communities (see, e.g., Fig. III.16 on page 110). However, given the DNS analysis results of our particular data set, the assumption that such activity existed for AG2 would be unsubstantiated.

Using these settings, we processed the selected evaluation period $\varepsilon_{\text{CONN}}$ of our data set CONN–DS3, using the previously extracted activity profiles as a reference. The scoring of individual connections was significantly slower than the extraction of activity profiles, and required about one day for processing one hour of data. As we used a standard desktop machine for our experiments, the large number of disk accesses for retrieving the activity profiles for scoring the individual connection, severely slowed down our system. However, as mentioned previously, the total volume of all activity profiles amounted to only ~ 60 GB of disk space. Besides the suggested improvements to the data representation (see §14.1), one can the improve the system's performance by using more main memory resources. Professional server systems can easily be configured with 60 GB of main memory, in which case the performance is expected to improve vastly. Therefore, we do not consider this as a significant limitation of our system.

The original (i.e., not reduced) TDG for the evaluation period had 2,8 million nodes and 36 million edges. In comparison, the reduced TDG produced by our system had 302,540 (i.e., 11%) nodes and 1,226,450 (i.e., 3.3%) edges. We proceed with the analysis of the reduced graph by using the previously collected DNS analysis results.

14.1.3 Joint Analysis Results

In the following, we use the DNS analysis results as a seed for further analyzing the reduced TDG. In particular, we consider all IP addresses in AG2 as malicious and thereby inject partial knowledge (as defined in §10) of malicious activity. Our goal consists of finding the graph community to which these IP addresses belong, i.e., the group of hosts with which the malicious hosts collaborate. From this structural connection information, we hope to be able to reveal further malicious activity which the DNS analysis was unable to detect.

Primarily, this is therefore a community detection problem as defined in §4.2, which we solve as follows. First, as a preprocessing step, we find the connected components in the TDG and identify the component $\mathcal{C}$ to which the seeds belong. Recall that connected components provide only a coarse partition of the original graph, but can be found very efficiently. Therefore, we further process $\mathcal{C}$ using the Louvain method, to find the communities of $\mathcal{C}$. However, as discussed in §4.2, the Louvain method suffers from a resolution limit which causes small communities in large graphs to be merged. Therefore, we continue with finding the (sub-) communities of the community to which the seeds belong, until any two seeds end up in two different communities. As we know that any two of our seeds are related to each other (i.e., are member of the same agile group of malicious sites), their assignment to different communities would be contradicting, and would indicate that the resulting resolution is too high. Note the simplicity of this technique, which is enabled only by the previously conducted data reduction procedure.

As a preparatory step, we had to identify the seeds for this particular analysis which were actually contained in the reduced TDG. Out of the 68 IP addresses in AG2, only 27 addresses appeared in CONN–DS3 during the evaluation period. This is not surprising, as the DNS analysis considered a significantly longer epoch of time (i.e., one week). Furthermore, one DNS NOERROR response may contain multiple IP addresses for the queried FQDN, and monitored hosts usually contact only one of these addresses. We found that 16 of the 27 IP addresses were contacted only by a single monitored host, and were therefore removed by the connection analysis' "Reduction II" procedure. Out of the remaining 11 IP addresses, seven addresses were removed by our approach's "Reduction I" procedure. After a manual investigation, we found that the main reason for their removal was that these IP addresses belonged to address ranges of popular hosting providers. In particular, the most often contacted IP address belonged to "Hetzner", a large German hosting provider. As expected, connections to these IP address ranges are considered normal for many of the monitored hosts, and were thus removed from the TDG. The remaining three IP addresses were present in the reduced TDG, and were used as seeds for our analysis.

Our results for AG2 were the following: the first analysis step yielded 274 connected components. One of them was a giant component which contained 99.7% of all nodes of the entire reduced TDG. All seeds belonged to this giant component. We applied the Louvain method and retrieved 73 communities. All seeds belonged to the same community with 13,151 nodes and 22,578 edges. We applied the Louvain method again, and retrieved 133 (sub-)communities. Once again, all seeds belonged to the same community with only 6 nodes and 5 edges. A further iteration of the Louvain method would have assigned the seeds to different communities. Therefore, we stopped at this point and considered this community the final result.

The identified graph community is shown in Fig. IV.2. It contains all three seeds, plus the additional IP address *195.3.145.94*, which was not contained in DNS–DS3 and could therefore not be found by our DNS analysis system. We found evidence that this IP address relates to malicious services[4]. Besides the actual detection of this additional malicious host, it is interesting to note which connections in CONN–DS3 were *not* represented in the reduced TDG. During the evaluation period, the two remaining monitored hosts in Fig. IV.2 contacted 155 and 54 external hosts, respectively. If all these connections would have been represented in the reduced TDG, it would have been significantly harder to isolate the malicious community and find the additional malicious host. Instead, the connection analysis approach enabled us to focus on the connections which are atypical for the corresponding monitored hosts, and allowed us to successfully identify the malicious community.

14.2 Discussion

DNS analysis provides reliable results independent of the number of queries and the number of requesting monitored hosts. This enables us to detect even malicious activity that

[4]`https://www.virustotal.com/en/ip-address/195.3.145.94/information/`

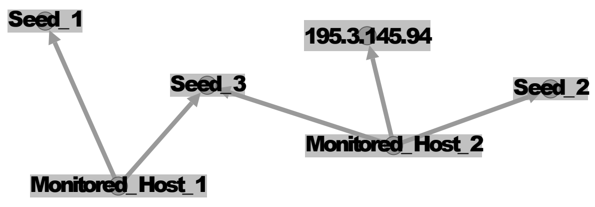


Figure IV.2: Community of hosts involved in malicious activities.

affects only a small number of monitored hosts. This is complemented by the connection analysis approach which provides heavily reduced connection graphs (TDGs) for further analysis of "who-contacted-whom". As we showed in our experiments, this allows us to find additional hosts which are involved in malicious activities. A clear merit of our system is thereby the use of minimal traffic information. On the one hand, this allows us to consider large numbers of monitored hosts without limiting the analysis to specific traffic features (e.g., payloads with a particular length or content) a priori, which may be altered or concealed by Internet criminals. Any (third-party) analysis approach, based on arbitrary traffic features, can therefore potentially provide seed information which can be used for better understanding the connections of individual, malicious hosts. This is particularly useful for the approaches discussed in §4.2.3.

The proposed joint analysis approach identifies both malicious FQDNs and the IP addresses hosting them, as well as the victim hosts. However, the system was not built for revealing any single malicious connection, but rather provides the big picture of the activity of platforms used for Internet crime. Due to the lack of application layer information (e.g., HTTP payloads), we cannot immediately learn from our analysis the details of actual activity, as, e.g., which particular attack is carried out. Instead, we provide topological connection information which enable the detection of sub-communities of hosts which are involved in malicious activity, irrespective of the specifics (e.g., protocols) of the malicious communication (cf. §2). Once this information is known to the network operator, additional analyses of the communication of these hosts can be initiated. A rich variety of analysis systems exists, which usually suffer from the problem that the vast amount of traffic data prevents a targeted application of these tools. Our approach addresses this problem of "finding the needle in the haystack" and complements existing systems.

We demonstrated this concept in §14.1.3 and showed that even a very simple analysis procedure (i.e., repeated runs of the Louvain method) is able to provide additional insight into malicious traffic activity. Due to the availability of reduced TDGs, each run of the Louvain method completes in under one minute, which makes our system practically usable for real-world monitoring of large networks. For the entire analysis procedure, we do not assume that any particular IP address (or range thereof) is benign a priori, as it is often done by other approaches, e.g., by using whitelists. With the increasing usage of CDNs and cloud services, IP addresses are more and more being used for multiple services. It is therefore becoming increasingly difficult to label a particular IP address as benign, as it potentially may be used by a malicious service in the next second. Our approach addresses this problem by considering all those connections as suspicious which involve PoPs that have not been contacted by the *specific* monitored host in question. Therefore, it becomes irrelevant which service used an IP address in the past. Rather, we analyze if a particular host is well-known to contact IP addresses at this PoP, and we derive an *individual* score. This enables the detection of communities of hosts which collaboratively engage in agile network activity.

Chapter 15

Conclusion and Outlook

In this thesis, two complementary approaches for network-based malware detection have been proposed. We focused on the detection of malicious service platforms which are commonly used for a variety of different "services" in the ecosystem of Internet crime. Both the DNS analysis and the connection analysis approach are driven by the idea of accurately modeling *normal* traffic activity patterns, and are able to detect *deviations* from these models. We explicitly take the inherent dynamics of (benign) Internet traffic into account, and continuously update the models automatically and efficiently over time. Individual suspicious relations among IP addresses and between IP addresses and FQDNs are represented as graphs. These graphs reveal the platform-like character of many different types of malicious activity, as we discussed in §2, §5, and §10. Our experiments with each separate analysis component (see §9 and §13) were conducted using traffic data from ISP networks and demonstrated the practical utility of the developed approaches. As a final evaluation, we discussed the joint usage of both approaches in §IV and showed their complementarity.

This work focused on the problem of collecting evidence for the presence of malicious service infrastructures in large networks. We consider the main challenge in such scenarios to consist of "finding the needle in the haystack". Massive amounts of network monitoring data are available for collection, but it is highly difficult to decide where to look first and which data should be considered at all for any analysis. Malicious Internet activity tries to blend in benign traffic and adapts its network footprint such that it is indistinguishable from benign traffic, w.r.t. higher layer traffic data. In the extreme case, encrypted criminal network communication exposes no application layer information which could be accessed by network-based analysis approaches, and would therefore be indistinguishable from benign traffic. However, given sufficient time, other analysis approaches as, e.g., dynamic analysis of malware binaries (see §4.1), are able to reveal that particular Internet hosts are involved in malicious activities. In view of that, malicious Internet services migrate from one Internet host to the next over time, in order to evade their detection and, consequently, their blocking. As we showed in this thesis, these evasive actions appear as *agile* relations in graphs extracted from traffic data, and are thus detectable.

Therefore, we consider our contributions as puzzle pieces in a larger framework of malware detection approaches. Together with these approaches, we are able to "corner" criminal activity and, ideally, limit its degrees of freedom to a point where it is not profitable anymore (see, e.g., §9.3.2). For this purpose, an essential design goal of our approach is the *timely* detection of malicious activity. Extensive processing delays would often result in analysis reports when it is already to late, and when a malicious service is not active anymore or moved on to a new host. The approaches proposed in this thesis require only minimal traffic data and are often able to report malicious activity only minutes after the corresponding traffic data was observed. The more agile the malicious activity patterns are, the faster we are able to detect them. Less agile activity requires more time for detection, but results in less reliable malicious service operation.

As our experiments showed, some benign activity is indistinguishable from malicious one, and any fully automatic alerting system would be prone to misclassify such events. Therefore, the detection framework should integrate a human analyst who is able to assess the detection results, and who can initiate further analyses when needed. Our proposed system is designed with the human analyst in mind, and provides *structured* detection results (graphs) which are better accessible than long lists of individual alerts and therefore enable the identification of entities which are central for a group of related events (see, e.g., §9.2.2).

Internet crime cannot be defeated by technology alone. Even the best detection systems are useless if, e.g., the process of issuing takedown notices for the detected malicious services is inefficient and comes into effect only long after the malicious activity was detected. The individual Internet stakeholders and, in particular, large organizations and ISPs need to share more information in order to detect malicious activity earlier and counteract more efficiently. Technological advances are required as supporting instruments, and should mainly address the privacy-preserving exchange of traffic analysis results (see, e.g., our own proposal [19]) as well as the exchange of incident reports (see, e.g., IETF working group on Managed Incident Lightweight Exchange (MILE)[1]). Above all, political actions are required for fostering the collaboration of victims, defenders, and legal authorities. The foundation of institutions like the European Network and Information Security Agency (ENISA) as well as the recently published Internet security strategy of the European Union [5] are important steps in this direction.

[1]`http://tools.ietf.org/wg/mile/`

APPENDIX **A**

ϕ-α Quantiles: Algorithm Details and Analysis

The complexity of computing profiles is dominated by calculating the ϕ-dominance values for N locations. A naïve implementation needs $\mathcal{O}(N^2)$ operations, which makes this approach unusable for large data sets with many monitored hosts. Güting et al. [78] gave an optimal algorithm for the 2D dominance counting problem matching the lower bound of $\Omega(N \log N)$ by a divide-and-conquer approach. We present here an easy-to-implement algorithm for the weighted case with the same asymptotic complexity, which scales with the number $n \leq N$ of *unique* points, where the number of occurrences of a point corresponds to its weight.

The main algorithm is shown in Algorithm A.3, and the pseudo-code for the encountered sub-problems is presented in Algorithms A.1 and A.2. A complete complexity analysis is provided in the following, using the notation in Table 12.

Theorem 1 *Given a set of points sorted according to* $\max\{\rho_x(\cdot), \rho_y(\cdot)\}$*, the* DOMINANCE *algorithm in Algorithm A.2 correctly determines the weighted dominance* $w(dom(p))$ *of each point* p*.*

Proof: For two points p_i and p_j sorted in increasing order according to $\max\{\rho_x(\cdot), \rho_y(\cdot)\}$

Input data and parameters:	
$\mathcal{P}$	- ordered set of unique points
$\mathcal{A}$	- set of α-values
Φ	- set of ϕ-values
Lookup tables and data structures:	
$\rho_x(\cdot), \rho_y(\cdot)$	- ranks in both dimensions
$w(\cdot)$	- weight of points
$\phi(\cdot)$	- *phi*-dominance values for all points
T_x, T_y	- binary indexed trees (BIT) for both dimensions with operations INSERT and GETCUMUL
$Q(\cdot,\cdot)$	- returns a ϕ-α-value
Variables:	
p	- currently processed point
c	- cumulative weight of points dominated by p
α	- α-value (skewness) of the current point
$\overline{w}$	- sum of weights of all points processed so far
W	- sum of all weights

Table 12: Parameters and variables.

Algorithm A.1: Algorithm PREPROCESS.

```
input  : $\mathcal{P}$, $w$
output: $\mathcal{P}$, $\rho_x$, $\rho_y$, $W$
1  $W \leftarrow 0$;
2  sort $\mathcal{P}$ w.r.t. x-coordinate;
3  for $i \leftarrow 1$ to $|\mathcal{P}|$ do
4      $\rho_x$ ($\mathcal{P}$ [i]) $\leftarrow i$;
5      $W \leftarrow W + w$ ($\mathcal{P}$ [i]);
6  end
7  sort $\mathcal{P}$ w.r.t. y-coordinate;
8  for $i \leftarrow 1$ to $|\mathcal{P}|$ do
9      $\rho_y$ ($\mathcal{P}$ [i]) $\leftarrow i$;
10 end
11 sort $\mathcal{P}$ w.r.t. max ($\rho_x$, $\rho_y$);
12 return $\mathcal{P}$, $\rho_x$, $\rho_y$, $W$
```

Algorithm A.2: Algorithm DOMINANCE.

```
input  : $\mathcal{P}$, $\rho_x$, $\rho_y$, $w$, $W$
output: $\Phi$
1  $\overline{w} \leftarrow 0$;
2  foreach $p \in \mathcal{P}$ do
3      $\overline{w} \leftarrow \overline{w} + w$ ($p$);
4      if $\rho_x$ (p) > $\rho_y$ (p) then
5          $C \leftarrow w$ ($p$) + GETCUMUL ($T_y$,$\rho_x$ (p));
6      end
7      else if $\rho_x$ (p) < $\rho_y$ (p) then
8          $C \leftarrow w$ ($p$) + GETCUMUL ($T_x$,$\rho_x$ (p));
9      end
10     else
11         $C \leftarrow \overline{w}$;
12     end
13     INSERT ($T_x$,$p$,$w$ (p));
14     INSERT ($T_y$,$p$,$w$ (p));
15     $\Phi$ (p) $\leftarrow C\ /W$;
16 end
17 return $\Phi$
```

with indices i and j holds: if $i > j$ then $\max\{\rho_x(p_i), \rho_y(p_i)\} > \max\{\rho_x(p_j), \rho_y(p_j)\}$. This implies that, given a fixed p_i, for all previous elements p_j the following holds:

$$\forall p_j \text{ with } j < i : \rho_x(p_i) \geq \max\{\rho_x(p_j), \rho_y(p_j)\}$$
$$\vee\ \forall p_j \text{ with } j < i : \rho_y(p_i) \geq \max\{\rho_x(p_j), \rho_y(p_j)\}$$

In other words, p_i's rank is maximal in either dimension or in both dimensions. If $\rho_x(p_i) > \rho_y(p_i)$, we know that $\rho_x(p_i)$ is the search key and from the considerations above follows that $\rho_x(p_i)$ is the maximum x-rank among all previous elements. For all next elements p_k with $k > i$ holds that they have a higher x-rank than p_i or a higher y-rank; otherwise it would contradict the sorting order. Thus, in order to obtain the number of elements dominated by p_i, it is sufficient to count the number of elements with lower y-rank than p_i. This is done by the counting query GETCUMUL.

Algorithm A.3: Algorithm ϕ/α-QUANTILES.

```
input  : $\mathcal{P}$, $\Phi$, $\mathcal{A}$
output: $\mathcal{Q}$
1  $\mathcal{P}$, $\rho_x$, $\rho_y$, $w$, $W \leftarrow$ PREPROCESS ($\mathcal{P}$, $w$);
2  $\Phi \leftarrow$ DOMINANCE ($\mathcal{P}$, $\rho_x$, $\rho_y$, $w$, $w$, $W$);
3  foreach $p \in \mathcal{P}$ do
4      $\alpha \leftarrow \rho_x\ (p)/(\rho_x\ (p)+\rho_y\ (p))$;
5      foreach $(a, b) \in \Phi \times \mathcal{A}$ do
6          if $\Phi$ (p) $< a$ and $\alpha < b$ then
7              if $\Phi$ (p) $> \mathcal{Q}$ (a,b) then
8                  $\mathcal{Q}$ (a,b) $\leftarrow p$;
9              end
10         end
11     end
12 end
13 return $\mathcal{Q}$
```

If $\rho_x(p_i) < \rho_y(p_i)$, similar arguments apply for the other dimension. In the special case $\rho_x(p_i) = \rho_y(p_i)$, p_i has both maximum x-rank and y-rank, thus it dominates all previous points p_j and its dominance is given by the overall weight of all points up to index i, including its own weight. ■

Lemma 1 DOMINANCE *requires* $\mathcal{O}(n \log n)$ *time and* $\mathcal{O}(n)$ *space.*

Proof: Querying the cumulative weight GETCUMUL from a binary indexed tree (BIT) as well as the INSERT operation both require $\mathcal{O}(\log n)$ time [59]. Both operations are called for each point of the data set (in each step of the foreach-loop) which results in an overall complexity of $\mathcal{O}(n \log n)$ for the DOMINANCE function.

The BITs can be implemented as arrays [59], which require linear space. ■

Theorem 2 ϕ/α-QUANTILES *requires* $\mathcal{O}(n \log n)$ *time and* $\mathcal{O}(n)$ *space.*

Proof: PREPROCESS (Algorithm A.1) first sorts the data w.r.t. x-coordinate, then w.r.t. y-coordinate. This takes time $\mathcal{O}(n \log n)$ and linear space. Then the ranks ρ_x, ρ_y can be obtained in a single pass over the sorted data for each dimension (linear time). The ranks are then used for sorting the data w.r.t. $\max\{\rho_x, \rho_y\}$, i.e. in the order in which the DOMINANCE function processes the points. This sorting step takes again $\mathcal{O}(n \log n)$ time. Storing ranks in a lookup table requires n elements to be stored for each dimension.

By Lemma 1, DOMINANCE (Algorithm A.2) calculates ϕ values for each element and needs $\mathcal{O}(n \log n)$ and linear space.

In the final iteration over the data set, for each point p we perform a constant number of checks (depending on the input parameter sets $\mathcal{A}$, Φ) in order to determine the maximum for each ϕ-α-quantile. This last iteration takes $n \cdot \mathcal{A} \cdot \Phi \in \mathcal{O}(n)$ time and no additional space.

Thus the complexity is dominated by PREPROCESS and DOMINANCE and requires $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space. ■

APPENDIX **B**

Database Schema

We store the activity profiles extracted by the connection analysis approach (see § III) in a PostGIS database[1], using the following simple table structure. The “user_data” table allows us to quickly lookup the timestamps of the evaluation epochs at which a specific monitored host was active, and for which we created an activity profile, consequently. For each epoch, we create one table “geom_data_⟨timestamp⟩” which stores all profiles of all hosts which were active in this epoch.

User Table Schema

```
CREATE TABLE user_data (
user_data_id serial primary key,
user_id bigint,
timestamp integer
);
```

Profiles Table Schema

```
CREATE TABLE geom_data_<timestamp> (
gid serial primary key,
polygon geometry,
weight real,
user_id bigint
);
```

Note that at the end of each epoch, we create a lookup index for the newly created “geom_data_⟨timestamp⟩” table, and recreate the index on the “user_data” table. This greatly improves the lookup performance of the activity profiles of individual hosts, and therefore enables quicker computation of the score for a particular connection.

Indexing Commands

```
CREATE INDEX user_idx ON user_data (user_id);
CREATE INDEX geom_data_idx_<timestamp>
  ON geom_data_<timestamp> (user_id);
```

[1] http://postgis.net

Curriculum Vitae

- **Experience**

3/2007 –	Researcher Communication Networks Group Forschungszentrum Telekommunikation Wien (FTW)
2/2009 – 5/2009	Visiting Researcher Network Systems Lab, Simon Fraser University, Vancouver, Canada

- **Education**

2004 – 2007	MSc Program Telematik Graz University of Technology
2000 – 2004	BSc Program Telematik Graz University of Technology
1989 – 1997	Bundesgymnasium Werndlpark, Steyr

Index

Bibliography

[1] KDD cup 1999 data. Available from: `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[2] Multiple DNS implementations vulnerable to cache poisoning. Available from: `http://www.kb.cert.org/vuls/id/800113`.

[3] Hacker intelligence initiative, monthly trend report #12. Technical report, Imperva, September 2012. Available from: `http://www.imperva.com/download.asp?id=31`.

[4] Norton cybercrime report. Technical report, Norton, 2012.

[5] Cybersecurity strategy of the european union: An open, safe and secure cyberspace. Technical report, European Commission, 2013.

[6] Lada A. Adamic. The small world web. In *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 443–452, London, UK, 1999.

[7] Chris Anderson. The long tail: Why the future of business is selling less of more. Technical report, Hyperion, 2006.

[8] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel van Eeten, Michael Levi, Tyler Moore, and Stefan Savage. Measuring the cost of cybercrime. In *Proceedings of the 11th Workshop on the Economics of Information Security (WEIS)*, Berlin, Germany, June 2012.

[9] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for DNS. In *Proceedings of USENIX Security*, pages 273–290, Berkeley, CA, 2010.

[10] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of USENIX Security*, page 27, Washington, DC, 2011.

[11] Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *New J. Phys.*, 10, 2008.

[12] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *DNS Security Introduction and Requirements*. Request for Comments. IETF, March 2005. Available from: `http://www.ietf.org/rfc/rfc4033.txt`.

[13] Matt Asay. Study: 95 percent of all e-mail sent in 2007 was spam. Available from: `http://news.cnet.com/8301-13505_3-9831556-16.html`.

[14] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM conference on computer and communications security (CCS)*, pages 186–205, New York, NY, 1999.

[15] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[16] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, December 1972.

[17] Ulrich Bayer, Imam Habibi, David Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, page 8, Boston, MA, 2009.

[18] Richard Bejtlich. *The Tao of network security monitoring: beyond intrusion detection*. Addison-Wesley, Boston, MA, 2004.

[19] Andreas Berger, Jacopo Cesareo, and Alessandro D'Alconzo. Collaborative network defense with minimum disclosure. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, Houston, TX, 2011.

[20] Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Oliver Jung. Locality matters: Reducing internet traffic graphs using location analysis. In *Proceedings of the Performance and Dependability Symposium (PDS) at the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, Budapest, Hungary, June 2013.

[21] Andreas Berger, Alessandro D'Alconzo, Wilfried N. Gansterer, and Antonio Pescapè. Detecting malware activity from agile DNS mappings using graph analysis. 2013. Submitted to IEEE Transactions on Dependable and Secure Computing.

[22] Andreas Berger and Wilfried N. Gansterer. Modeling DNS agility with DNSMap. In *Proceedings of IEEE INFOCOM Workshop on Traffic Monitoring and Analysis (TMA)*, pages 387–392, Turin, Italy, April 2013.

[23] Andreas Berger, Ivan Gojmerac, and Oliver Jung. Internet security meets the IP multimedia subsystem: An overview. *Security and Communications Networks*, 3:185–206, 2009.

[24] Andreas Berger and Mohamed Hefeeda. Exploiting SIP for botnet communication. In *Proceedings of the 5th Workshop on Secure Network Protocols (NPSEC)*, pages 31–36, Princeton, NJ, 2009.

[25] Andreas Berger and Eduard Natale. Assessing the real-world dynamics of DNS. In *Proceedings of the 4th international workshop on Traffic Monitoring and Analysis (TMA)*, pages 1–14, Vienna, Austria, 2012.

[26] 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 (ACSAC)*, pages 129–138, New York, NY, 2012.

[27] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: finding malicious domains using passive DNS analysis. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2011.

[28] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), October 2008.

[29] Burton H Bloom. Space/Time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[30] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *physics/0608255*, August 2006.

[31] Peter Bright. How operation b107 decapitated the rustock botnet. Available from: `http://arstechnica.com/information-technology/2011/03/how-operation-b107-decapitated-the-rustock-botnet/`.

[32] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of USENIX Security*, pages 13–13, San Francisco, CA, 2011.

[33] F. Casacuberta and M.D. de Antoni. A greedy algorithm for computing approximate median strings. In *Proceedings of National Symposium on Pattern Recognition and Image Analyis*, pages 193–198, Barcelona, Spain, 1997.

[34] Djalil Chafai and Didier Concordet. Confidence regions for the multinomial parameter with small sample size. *Journal of the American Statistical Association*, 104(487):1071–1079, September 2009.

[35] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the inside: a view of botnet management from infiltration. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET)*, Washington, DC, 2010.

[36] Hyunsang Choi, Heejo Lee, and Hyogon Kim. BotGAD: detecting botnets by capturing group activities in network traffic. In *Proceedings of the fourth International ICST conference on communication system software and middleware (COMSWARE)*, page 2, New York, NY, 2009.

[37] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. Request for Comments. IETF, October 2004. Available from: `http://www.ietf.org/rfc/rfc3954.txt`.

[38] B. Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. Request for Comments. IETF, January 2008. Available from: `http://www.ietf.org/rfc/rfc5101.txt`.

[39] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70, 2004.

[40] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[41] M.P. Collins. Graph-based analysis in network security. In *Proceedings of the military communications conference (MILCOM)*, pages 1333–1337, Baltimore, MD, 2011.

[42] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Summarizing two-dimensional data with skyline-based statistical descriptors. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 42–60, Hongkong, China, 2008.

[43] Baris Coskun and Sven Dietrich. Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 131–140, Austin, TX, December 2010.

[44] Luciano da F. Costa, Francisco Rodrigues, Gonzalo Travieso, and P. R. Villas Boas. Characterization of complex networks: A survey of measurements. *Adv. Phys.*, 56:167–242, 2007.

[45] David Dagon and Guofei Gu. A taxonomy of botnet structures. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 325–339, Miami Beach, FL, 2007.

[46] David Dagon, Guofei Gu, Cliff Zou, Julian Grizzard, Sanjeev Dwivedi, Wenke Lee, and Richard Lipton. A taxonomy of botnets. *Proceedings of CAIDA DNS-OARC Workshop*, 2005.

[47] David Dagon, Cliff Zou, and Wenke Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)*, pages 2–13, San Diego, CA, 2006.

[48] Alessandro D'Alconzo, Angelo Coluccia, Fabio Ricciato, and Peter Romirer-Maierhofer. A distribution-based approach to anomaly detection for 3G mobile networks. In *Proceedings of the IEEE Global Communication Conference (GLOBECOM)*, pages 1–8, Honolulu, HI, 2009.

[49] Neil Daswani and Michael Stoppelman. The anatomy of Clickbot.A. In *Proceedings of the workshop on Hot Topics in Understanding Botnets (HotBots)*, page 11, Cambridge, CA, 2007.

[50] David Dittrich and Sven Dietrich. Command and control structures in malware: From Handler/Agent to P2P. *USENIX :login:*, 32(6), December 2007.

[51] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning (ICML)*, pages 233–240, New York, NY, 2006.

[52] D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222 – 232, February 1987.

[53] Guillaume Dewaele, Kensuke Fukuda, Pierre Borgnat, Patrice Abry, and Kenjiro Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the workshop on Large Scale Attack Defense (LSAD)*, pages 145–152, Kyoto, Japan, 2007.

[54] Christian J. Dietrich, Christian Rossow, and Norbert Pohlmann. CoCoSpot: clustering and recognizing botnet command and control channels using traffic analysis. *Computer Networks*, 57(2):475–486, February 2013.

[55] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44(2):6:1–6:42, March 2008.

[56] Martin Ester, Hans-Peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, Portland, OR, 1996.

[57] European Parliament and the Council of the European Union. Directive 95/46/EC of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Union*, L 281:31–50, 1995.

[58] Tom Fawcett. ROC graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4, HP Laboratories, 2003.

[59] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol - HTTP/1.1*. Request for Comments. IETF, June 1999. Available from: `http://www.ietf.org/rfc/rfc2616.txt`.

[61] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[62] Dinei A. F. Florêncio and Cormac Herley. Evaluating a trial deployment of password re-use for phishing prevention. In *Proceedings of the eCrime Researchers Summit*, pages 26–36, 2007.

[63] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Uncovering relations between traffic classifiers and anomaly detectors via graph theory. In *Proceedings of the 2nd international Workshop on Traffic Monitoring and Analysis (TMA)*, pages 101–114, Zurich, Switzerland, 2010.

[64] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[65] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, January 2007.

[66] Marc Fossi. Internet security threat report, trends for 2010. Technical report, Symantec, April 2011.

[67] Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*, pages 375–388, Alexandria, VA, 2007.

[68] Jérôme François, Shaonan Wang, Radu State, and Thomas Engel. BotTrack: tracking botnets using NetFlow and PageRank. In *Proceedings of the 10th international IFIP TC 6 conference on Networking*, pages 1–14, Valencia, Spain, 2011.

[69] Howard Fraser. Exploring the blackhole exploit kit. Available from: `http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/`.

[70] Yan Gao, Yao Zhao, R. Schweller, S. Venkataraman, Yan Chen, Dawn Song, and Ming-Yang Kao. Detecting stealthy spreaders using online outdegree histograms. In *Proceedings of the 15th IEEE International Workshop on Quality of Service (IWQoS)*, pages 145–153, Evanston, IL, 2007.

[71] Gavin O'Gorman and Geoff McDonald. Ransomware: A growing menace. Technical report, Symantec, November 2012.

[72] Jan Goebel and Thorsten Holz. Rishi: identify bot contaminated hosts by IRC nickname evaluation. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets (HotBots)*, page 8, Cambridge, CA, 2007.

[73] Benjamin H. Good, Yves-Alexandre de Montjoye, and Aaron Clauset. The performance of modularity maximization in practical contexts. *Phys. Rev. E*, 81, 2010.

[74] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the ACM conference on computer and communications security (CCS)*, pages 821–832, Raleigh, NC, 2012.

[75] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of USENIX Security*, pages 139–154, San Jose, CA, 2008.

[76] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: detecting botnet command and control channels in network traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2008.

[77] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. A study of the packer problem and its solutions. In *Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection (RAID)*, pages 98–115, 2008.

[78] Ralf-Hartmut Güting, Otto Nurmi, and Thomas Ottmann. Fast algorithms for direct enclosures and direct dominances. *Journal of Algorithms*, 10(2):170–186, 1989.

[79] Eric Haines. Graphics gems IV. page 24–46. Academic Press Professional, Inc., San Diego, CA, 1994.

[80] Shuang Hao, Nick Feamster, and Ramakant Pandrangi. An internet wide view into DNS lookup patterns. Technical report, VeriSign Incorporated, 2010.

[81] Shuang Hao, Nick Feamster, and Ramakant Pandrangi. Monitoring the initial DNS behavior of malicious domains. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC)*, pages 269–278, Berlin, Germany, 2011.

[82] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *Proceedngs of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2008.

[83] 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 (LEET)*, pages 1–9, San Francisco, CA, 2008.

[84] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[85] Xin Hu, M. Knysz, and K. G Shin. Measurement and analysis of global IP-usage patterns of fast-flux botnets. In *Proceedings of the Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 2633–2641, Shanghai, China, 2011.

[86] Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *American Statistician*, 50(4):361–365, 1996.

[87] Marios Iliofotou, Michalis Faloutsos, and Michael Mitzenmacher. Exploiting dynamicity in graph-based traffic analysis: techniques and applications. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT)*, pages 241–252, Rome, Italy, 2009.

[88] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 315–320, San Diego, CA, 2007.

[89] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. JACKSTRAWS: picking command and control connections from bot traffic. In *Proceedings of USENIX Security*, page 29, San Francisco, CA, 2011.

[90] Nan Jiang, Jin Cao, Yu Jin, Li Erran Li, and Zhi-Li Zhang. Identifying suspicious activities through DNS failure graph analysis. In *Proceedings of the 18th IEEE International Conference on Network Protocols (ICNP)*, pages 144–153, Kyoto, Japan, 2010.

[91] Yu Jin, Esam Sharafuddin, and Zhi-Li Zhang. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *Proceedings of the eleventh international joint conference on measurement and modeling of computer systems*, pages 49–60, Seattle, WA, 2009.

[92] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, Philadelphia, PA, 2005.

[93] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the first Workshop on Hot Topics in Understanding Botnets (HotBots)*, Cambridge, MA, 2007.

[94] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999.

[95] M. Knysz, Xin Hu, and K.G. Shin. Good guys vs. bot guise: Mimicry attacks against fast-flux detection systems. In *Proceedings of the Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 1844–1852, Shanghai, China, 2011.

[96] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of USENIX Security*, pages 351–366, Montreal, Canada, 2009.

[97] Clemens Kolbitsch, Engin Kirda, and Christopher Kruegel. The power of procrastination: detection and mitigation of execution-stalling malicious code. In *Proceedings of the 18th ACM conference on computer and communications security (CCS)*, pages 285–296, Chicago, IL, 2011.

[98] Maria Konte, Nick Feamster, and Jaeyeon Jung. Dynamics of online scam hosting infrastructure. In *Proceedings of the 10th International Conference on Passive and Active Network Measurement (PAM)*, pages 219–228, Seoul, South Korea, 2009.

[99] Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. GQ: practical containment for measuring modern malware systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC)*, pages 397–412, Berlin, Germany, 2011.

[100] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80, 2009.

[101] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical Review E*, 84, 2011.

[102] Andrea Lancichinetti and Santo Fortunato. Consensus clustering in complex networks. *Scientific Reports*, 2, March 2012.

[103] Andrea Lancichinetti, Filippo Radicchi, Jose' Javier Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PLoS One*, 6(4), 2011.

[104] Alex Lanstein. An overview of rustock. Technical report, FireEye. Available from: `http://www.fireeye.com/blog/technical/botnet-activities-research/2011/03/an-overview-of-rustock.html`.

[105] Do Quoc Le, Taeyoel Jeong, H. Eduardo Roman, and James Won-Ki Hong. Traffic dispersion graph based anomaly detection. In *Proceedings of the Second Symposium on Information and Communication Technology (SoICT)*, pages 36–41, Hanoi, Vietnam, 2011.

[106] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[107] Geoffrey Mack. The alexa blog: Why does my alexa rank jump around? a: The long tail, April 2009. Available from: `http://blog.alexa.com/2009/04/why-does-my-alexa-rank-jump-around-the.html`.

[108] MalwareIntelligence. Inside phoenix exploit's kit 2.8 mini version, October 2011. Available from: `http://malwareint.blogspot.com.es/2011/10/inside-phoenix-exploits-kit-28-mini.html`.

[109] A. Matrosov, E. Rodionov, D. Harley, and J. Malcho. Stuxnet under the microscope. Technical report, ESET, January 2011. Available from: `http://www.eset.com/resources/whitepapers/Stuxnet_Under_the_Microscope.pdf`.

[110] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.

[111] Stanley Milgram. The small world problem. *Psychology Today*, 1:61–67, May 1967.

[112] P.V. Mockapetris. *Domain names - concepts and facilities*. Request for Comments. IETF, November 1987. Available from: `http://www.ietf.org/rfc/rfc1034.txt`.

[113] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, pages 421–430, Miami, FL, 2007.

[114] Atif Mushtaq and Alex Lanstein. Srizbi control regained by original owner. Technical report, FireEye, 2008. Available from: `http://blog.fireeye.com/research/2008/11/its-srizbi-trun-now.html`.

[115] Shishir Nagaraja, Amir Houmansadr, Pratch Piyawongwisal, Vijit Singh, Pragya Agarwal, and Nikita Borisov. Stegobot: a covert social network botnet. In *Proceedings of the 13th international conference on Information hiding (IH)*, pages 299–313, Prague, Czech Republic, 2011.

[116] Shishir Nagaraja, Prateek Mittal, Chi-yao Hong, Matthew Caesar, and Nikita Borisov. BotGrep: finding P2P bots with structured graph analysis. In *Proceedings of USENIX Security*, pages 95–110, Washington, DC, 2010.

[117] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pages 24–31, Alexandria, VA, 2008.

[118] 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 (ACSAC)*, pages 21–30, Orlando, FL, 2011.

[119] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, June 2006.

[120] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2), 2004.

[121] Mark Newman. *Networks: An Introduction.* Oxford University Press, USA, 2010.

[122] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, 2010.

[123] Gunter Ollmann. The botnet vs. malware relationship. Technical report, Damballa, 2009.

[124] Gunter Ollmann. Behind today's crimeware installation lifecycle: How advanced malware morphs to remain stealthy and persistent. Technical report, Damballa, May 2011. Available from: `http://www.damballa.com/downloads/r_pubs/WP_Advanced_Malware_Install_LifeCycle.pdf`.

[125] Gunter Ollmann. Blacklists & dynamic reputation understanding why the evolving threat eludes blacklists. Technical report, Damballa, 2011.

[126] Gunter Ollmann. The evolution of network antivirus. Technical report, Damballa, October 2012.

[127] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference (WWW)*, Brisbane, Australia, 1998.

[128] Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX conference on offensive technologies (WOOT)*, page 86, Montreal, Canada, 2009.

[129] 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)*, pages 186–206, Paris, France, 2008.

[130] R. Perdisci, I. Corona, and G. Giacinto. Early detection of malicious flux networks via large-scale passive DNS traffic analysis. *IEEE Transactions on Dependable and Secure Computing*, 9(5):714–726, October 2012.

[131] Roberto Perdisci, Davide Ariu, and Giorgio Giacinto. Scalable fine-grained behavioral clustering of HTTP-based malware. *Computer Networks*, 57(2):487–500, February 2013.

[132] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2007.

[133] Dave Piscitello. Conficker summary and review. Technical report, ICANN, May 2010. Available from: `http://icann.org/en/security/conficker-summary-review-07may10-en.pdf`.

[134] David Plonka and Paul Barford. Context-aware clustering of DNS query traffic. In *Proceedings of the ACM SIGCOMM conference on Internet measurement (IMC)*, pages 217–230, Vouliagmeni, Greece, 2008.

[135] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. A foray into conficker's logic and rendezvous points. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET)*, Boston, MA, 2009.

[136] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to us. In *Proceedings of USENIX Security*, San Jose, CA, 2008.

[137] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting system emulators. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *Proceedings of the 10th international conference on Information Security (ISC)*, pages 1–18, Valparaiso, Chile, 2007.

[138] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1), 2006.

[139] C. Rigney, S. Willens, A. Rubens, and W. Simpson. *Remote Authentication Dial In User Service (RADIUS)*. Request for Comments. IETF, June 2000. Available from: `http://www.ietf.org/rfc/rfc2865.txt`.

[140] Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C. Freiling, and Norbert Pohlmann. Sandnet: network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pages 78–88, Salzburg, Austria, 2011.

[141] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, January 2008.

[142] Stefan Ruehrup, Pierfrancesco Urbano, Andreas Berger, and Alessandro D'Alconzo. Botnet detection revisited: theory and practice of finding malicious P2P networks via internet connection graphs. In *Proceedings of the INFOCOM workshop on Traffic Monitoring and Analysis (TMA)*, pages 435–440, Turin, Italy, 2013.

[143] Shashi Shekhar and Sanjay Chawla. *Spatial databases : a tour*. Prentice Hall, Upper Saddle River, NJ, 2003.

[144] Gordon Snow. Cyber security: Threats to the financial sector, September 2011. Available from: `http://financialservices.house.gov/UploadedFiles/091411snow.pdf`.

[145] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 305–316, Oakland, CA, 2010.

[146] Aditya K. Sood and Richard J. Enbody. Crimeware-as-a-service—A survey of commoditized crimeware in the underground market. *International Journal of Critical Infrastructure Protection*, 6(1):28–38, 2013.

[147] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: a botmaster's perspective of coordinating large-scale spam campaigns. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats (LEET)*, Boston, MA, 2011.

[148] W. Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 1–24. 2008.

[149] M. Tavallaee, E. Bagheri, Wei Lu, and A.A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, Ottawa, Canada, 2009.

[150] Rick van Luvender. Fraud trends in 2010: Top threats from a growing underground economy. Technical report, FirstData, April 2010.

[151] R. Villamarin-Salomon and J.C. Brustoloni. Identifying botnets using anomaly detection techniques applied to DNS traffic. In *Proceedings of the 5th IEEE Consumer Communications & Networking Conference (CCNC)*, pages 476–481, Las Vegas, NV, 2008.

[152] Tao Wang and Shun-Zheng Yu. Centralized botnet detection by traffic aggregation. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, pages 86–93, Chengdu, China, 2009.

[153] DJ Watts and SH Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):409–10, 1998.

[154] Kenneth C. Wilbur and Yi Zhu. Click fraud. *Marketing Science*, 28(2):293–308, March 2009. Available from: `http://mktsci.journal.informs.org/content/28/2/293`.

[155] Gilbert Wondracek, Thorsten Holz, Christian Platzer, Engin Kirda, and Christopher Kruegel. Is the internet for porn? an insight into the online adult industry. In *Proceedings of the 9th Workshop on the Economics of Information Security (WEIS)*, Cambridge, MA, 2010.

[156] Gilbert Wondracek, Paulo Milano, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, pages 1–14, San Diego, CA, 2008.

[157] Dallas Wood and Brent Rowe. Assessing home internet users' demand for security: Will they pay ISPs? In *Proceedings of the workshop on the Economics of Information Security (WEIS)*, Fairfax, VA, 2011.

[158] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on research in computer security (ESORICS)*, pages 232–249, Saint Malo, France, 2009.

[159] Kuai Xu, Feng Wang, and Lin Gu. Network-aware behavior clustering of internet end hosts. In *Proceedings of the Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 2078–2086, Shanghai, China, 2011.

[160] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 48–61, Melbourne, Australia, 2010.

[161] Zonghua Zhang, Ruo Ando, and Youki Kadobayashi. Hardening botnet by a rational botmaster. In *Information Security and Cryptology*, pages 348–369. 2009.