Botnet Detection Using Passive DNS

Master Thesis

Pedro Marques da Luz
pa.vasconcelosalvamarquesdaluz@student.ru.nl

Supervisors:
Erik Poll (RU)
Harald Vranken (RU & OU)
Sicco Verwer (TUDelft)
Barry Weymes (Fox-IT)

Department of Computing Science
Radboud University Nijmegen

2013/2014

Abstract

The Domain Name System (DNS) is a distributed naming system fundamental for the normal operation of the Internet. It provides a mapping between user-friendly domain names and IP addresses. Cyber criminals use the flexibility provided by the DNS to deploy certain techniques that allow them to hide the Command and Control (CnC) servers used to manage their botnets and frustrate the detection efforts. Passive DNS (pDNS) data allows us to analyse the DNS history of a given domain name. Such is achieved by passively collecting DNS queries and the respective answers that can then be stored and easily queried. By analyzing pDNS data, one can try to follow the traces left by such techniques and be able to identify the real addresses of the botnet Command and Control servers. For instance, we expect malware-related domain names to have lower Time-to-Live (TTL) values than legitimate and benign domains.

The aim of this research is the development of a proof-of-concept able to automatically analyze and identify botnet activity using pDNS data. We propose the use of machine learning techniques and devise a set of 36 different features to be used in the classification process. With two weeks of pDNS data we were able to set up, create and test different classifiers, namely k-Nearest Neighbours (kNN), Decision Trees and Random Forests. Using all-purpose blacklists we were able to achieve an accuracy of 97%, having a False Positive Rate (FPR) of 3%. However, with only two weeks of data it is not possible to find sufficient domain names used for botnet CnC servers such that we are able to extract statistically significant results. Furthermore, some of our initial assumptions hold when analysing botnet-related domain names but do not for malware-related domain names. For instance, the average TTL value for legitimate domain names is twice lower than for malware-related domain names. We believe this is due to the fact that only a small portion of our blacklist is composed of botnet-related domain names that have small TTL values. In addition, many legitimate domain names make use small values of TTL possibly to increase the availability of the services provided.

Related work such as Notos [2], EXPOSURE [10] and Kopis [3] reported similar accuracy levels, however with lower FPRs. This might be due to the fact that while our feature set is extracted solely from pDNS data, such systems include also WHOIS and Autonomous System (AS) data. This data is also useful for detection of malware-related domain names and it contributes to build more accurate and precise systems.

[Keywords: Botnet, Malware, Detection, DNS, Passive DNS, Machine Learning, Classification]

Acknowledgments

- First, and foremost, I would like to thank my parents that made my studies abroad possible and supported me throughout my life.

- Secondly, I would like express my gratitude to Fox-IT for conceding me an internship and to allow me to work with their data. I would also like to thank all members of the Cybercrime team for their suggestions and advices, in particular to Barry Weymes who supervised me during this period.

- In addition, I would also like to thank my other supervisors, namely Erik Poll, Harald Vranken and Sicco Verwer for guiding me throughout the development of this thesis.

- Last, but not the least, to my girlfriend Clara and my dearest friend Roman for putting up with me in the past few months, for listening to my endless speeches on the topic and proofread my work.

Contents

Acronyms

ANS Authoritative Name Server.

AS Autonomous System.

BGP Border Gateway Protocol.

CDN Content Delivery Network.

CnC Command and Control.

DDoS Distributed Denial-of-Service.

DGA Domain Generation Algorithm.

DNS Domain Name System.

DoS Denial-of-Service.

FF Fast-Flux.

FPR False Positive Rate.

HTTP Hypertext Transfer Protocol.

IDS Intrusion Detection System.

IRC Internet Relay Chat.

ISP Internet Service Provider.

kNN k-Nearest Neighbours.

MCC Matthews Correlation Coefficient.

NS Name Server.

P2P Peer-to-Peer.

pDNS Passive DNS.

RFC Request For Comments.

ROC Receiver Operating Characteristic.

RR Resource Record.

TPR True Positive Rate.

TTL Time-To-Live.

1 Introduction

The Domain Name System (DNS) is of paramount importance in the operation of the Internet. It is responsible for translating human-friendly domain names into IP addresses. The concept of mapping intelligible domain names to IP addresses dates back to ARPANET time. In those days, a global table containing all existing mappings was used. Each time a host had to connect to the network it would have to download the table to ensure it had the most recent version. As the Internet growed, such a non-scalable solution became impracticable. As such, in 1983, Mockapetris submitted the Request For Comments (RFC) 882 and 883 containing the first proposal of a decentralised DNS. Nowadays, DNS is used by a wide range of services, such as mail transfer applications or implementations of blacklists. Consequently, the value of DNS goes beyond the mapping of IP addresses to domain names and, the Internet as we know it today, highly depends on it. Even though domain names are typically used for benign purposes, they can equally serve malicious ones. In the past few years, domain names and DNS have been used by cyber criminals to manage botnets, host malicious content or facilitate the management of phishing websites to steal users' information [24]. The fact that DNS is a distributed and global system makes it perfect for cyber criminals which wish to carry on attacks on a global scale.

A botnet is a collection of hosts (bots) under control of a common Command and Control (CnC) server or centre. The CnC server is used by a remote controller to issue commands to the different bots. Whenever an attacker is able to compromise an end-user machine, this machine stilly becomes a bot that can be controlled by the botmaster or botherder [1]. Botnets may be used in a wide range of applications, including malicious and benign ones. For instance, one of the original botnets, Eggdrop (1993), was developed to facilitate the Internet Relay Chat (IRC) management [18]. However, typical applications of botnets include Distributed Denial-of-Service (DDoS) attacks, identity theft, proxy, spreading of malware or spamming. Over the last few years, botmasters have been making use of domain names to manage their distributed network of bots. In addition, botmasters devised techniques that take advantage of DNS agility (ability to quickly change the IP address associated with a domain name). By continuously changing the IP address associated with a domain name, botmasters are able to avoid capture and frustrate the detection of their CnC servers. Such techniques are often difficult to detect and make botnets very difficult to investigate.

The DNS only stores and provides current information over a domain name. Hence, as soon as a DNS entry expires, it is discarded and new information is fetched. Furthermore, botmasters have become experts in hiding their tracks. Hence, it is of paramount importance to be able to follow and analyse their trails over time. Passive DNS (pDNS) is a technique devised by Florian Weimer [46] that can be used to passively collect and aggregate DNS data. By collecting DNS information over time, one is able to analyse the DNS history and evolution of the domain names. With such information, we are then able to study and keep track of the botnet behaviour, with respect to DNS information and behaviour.

1.1 Motivation

DNS agility is not a property desired or claimed only by botmasters and their botnets. Different techniques have been developed that make use of DNS agility to improve the performance and

[1] The one in charge of the botnet.

availability of legitimate services. Two of such techniques include Round-Robin DNS and Content Delivery Network (CDN) [25]. Round-Robin DNS allows content to be evenly distributed over a set of machines. To achieve this, each time the same domain name is queried a single IP address is selected from a pool of possible IP addresses in a round-robin fashion. Such technique increases the resilience against DoS attacks by load-balancing the traffic directed towards the servers. CDNs try to perform load-balancing not only across different machines in the same location, but with different machines spread across different geographical locations. When used in cooperation with DNS information, CDNs become able to move the content closer to the end-user.

Botnets became the centre of cybercrime and are used to perform all sorts of attacks. By using domain names to manage their infrastructure, botmasters are able to migrate their CnC servers with ease. Furthermore, botmasters can also use Round-Robin DNS, providing a straightforward way to cycle the IP addresses associated with a domain name, and CDN-like techniques on their own botnets, such that each time a different server is used to process the requests. Such techniques allow botnets to achieve a high level of agility and increase their availability. Botnets are also known for using several communication protocols and having a heterogeneous structure. The fact that botnets can be so diverse, that botmasters may hide their CnC centre behind a proxy server and/or use IP cycling to frustrate detection efforts makes them difficult to detect and control.

Identification and categorization of botnets that requires little human effort may have a great impact in the uphill struggle against botnet propagation. It is not only useful to identify potential victims in the local network but also to enhance domain name reputation based on the DNS history of the domains. Even though these measures might not be sufficient to control the botnet propagation on their own, they contribute to decrease the spreading rate.

Furthermore, botnets are in constant development and new techniques to avoid detection are continuously being developed. In addition, botnets show a high level of diversity with respect to communication protocols, topologies and propagation mechanisms. Hence, techniques for botnet detection should ideally be independent of protocols, structure or payload, such that these techniques are able to bypass the constant updates and modifications issued by the botmasters [47]. As such, these techniques should rather focus on intrinsic properties and behaviour of botnets that cannot be easily modified or forged.

1.2 Contribution & Organization

In this thesis, we propose to use pDNS [19] data to detect domain names related to malicious activities. We expect domains associated with malicious activity to exhibit certain characteristics that differ from legitimate domains. As such, we make use of machine learning tools to train a classifier using supervised learning. In order to train the classifier, we first build a training set consisting of domain names that we can label as malicious or benign. Over each domain name, we extract a set of 36 features to be used by our classifier. These features attempt to characterise the domain names, such that malware-related domain names exhibit different characteristics than legitimate ones.

We are then able to use the resulting classifier to classify any domain name using its set of features. For instance, domain names related to malicious activity are often registered with small Time-To-Live (TTL) values (see Sec. 3). As such, we can use the TTL value as a feature to be used by our classifier to identify domain names associated to malicious activity. In addition, botmasters often change the IP address of the CnC server to frustrate take down efforts and attract

less attention from legal authorities. This will make the list of IP addresses associated with a domain used for CnC server grow over time. On the other hand, legitimate domains tend to keep this list without many modifications. This information can also be used as feature to measure how much the list of IP addresses of a domain name changes over time.

A classifier trained to distinguish legitimate domain names from domain names associated with malicious activity can be used also to detect domain names associated with botnet activity. We will train this classifier using a set of domain names known to be associated to malicious activity (blacklist) and, a set of legitimate and well-known domain names (whitelist). We will then measure the performance of this classifier in the detection of domains associated with malicious activity, and more particularly, with botnet activity. Consequently, we will attempt to answer the following questions:

- Can such a classifier efficiently and effectively distinguish pDNS data generated by domain names associated with malicious activity, from data generated from legitimate and benign domains?
- In particular, can such a classifier be used to detect domain names used in botnet activity?

The only established requirement for our classifier was the ability to classify domain names as the DNS queries are issued, hence be able to classify hundred queries per second (on a network of reasonable size). While the computational power required to classify each sample varies with the chosen algorithm, the biggest challenge of such requirement is to derive our feature set solely from the already available pDNS data. For instance, WHOIS data has been used by different detection techniques to enhance the feature set (see Sec. 8) but the lookup services of such data typically do not offer the required time guarantees. As such, we are not able to include such data in our analysis.

The remainder of this thesis is organized as follow: In Section 2 we provide an in-depth overview of botnets, covering different topologies, communication protocols, propagation vectors, life-cycle phases and botnet detection techniques. In Section 3, we introduce the DNS structure and organization, and elaborate on the techniques used by cyber criminals that take advantage of DNS agility, namely Domain Generation Algorithm (DGA) and Fast-Flux (Sections 3.2.1 and 3.2.2). We conclude this section with an introduction to pDNS and its properties (Section 3.3). Section 4 describes the pDNS data used in this research, how it was collected, aggregated and stored. We present background on the machine learning field in Section 5. Section 6 presents the analysis done over the DNS data and provides an in-depth description over the extracted features. In addition, we describe how we built our training sets and the different metrics used in our experiments. In Section 7 we evaluate the extracted features and analyse the results obtained. In Section 8 we elaborate on related work, presenting other research projects related with botnet detection and with the use of pDNS for detection of malware-related domain names. Finally, we conclude in Section9 presenting the future work and discussion.

2 Botnets: Engaging Hydras

Botmasters design their botnets such that these become as modular, robust and stealthy as possible. In addition, botnet controllers are able to propagate changes to the entire botnet easily or, if desirable, to individual bots. A traditional approach to take down a botnet consists of identifying

the IP address of the CnC server and try to take it down, disrupting the botmaster's ability to issue new commands to the botnet. While this approach yields good results for older botnets, more recent and advanced botnets started to create redundancy at the CnC level. In a way, we are no longer facing a snake but rather a hydra [31,36]: whenever researchers or the authorities are able to identify and seize a CnC server, others are created to replace it. This shifty behaviour is one reason why botnets became so difficult to detect and take down.

In this section we provide an overview over the different topologies and communication protocols used by botnets. Even though each particular botnet has its own particular characteristics, they typically follow the same steps throughout their life-time [23]. The understanding of which characteristics are shared among different botnet families, including the botnet life-cycle and defensive mechanisms, is of paramount importance to understand the botnet phenomenon. Furthermore, we discuss different approaches for botnet detection including honeypots and passive monitoring and analysis of network data. In this analysis, we leave out pDNS-based techniques that will be addressed latter with the related work in Sec. 8.

2.1 Topologies

The botnet's controller must ensure that the CnC infrastructure is robust enough to control a large amount of bots and to withstand attempts of sinkholing or shutdown. Furthermore, other factors need to be considered while selecting the botnet topology, such as business model and the level of stealthiness desired for the botnet. Decisions made related to the botnet topology will differently influence the botnet complexity, message latency, survivability and detectability [7, 16, 23].

2.1.1 Single-Star

A single-star CnC centre is by far the simplest topology to implement. It consists of one single CnC centre, typically hosted at one physical server. Whenever a new host is compromised, hence becoming a bot, it will connect to the centralised CnC (see Fig. 1).

The simplicity of the design, associated with the low message latency originated from the direct connection among bots and CnC, make such a scheme very appealing. Furthermore, the botmaster has a single point of control: one control server for all the bots in the botnet. When considering that botnets might have thousands and thousands of bots this might be a very compelling reason to choose such topology.

On the other hand, a single point of control also implies a single point of failure. If the botmaster loses control over its CnC centre it will lose control over the entire botnet. In addition, each bot will connect back to the same CnC server, this generates a lot of traffic towards a single point which is more likely to attract attentions from legal authorities.

2.1.2 Multi-Star

CnC servers with multi-star topology emerged to circumvent the problems of a single-star topology. Instead of one CnC server, the botmaster has multiple servers that can be used to manage the botnet. These control servers will communicate among one another to control the botnet and also give the appearance of a central point (see Fig. 2).

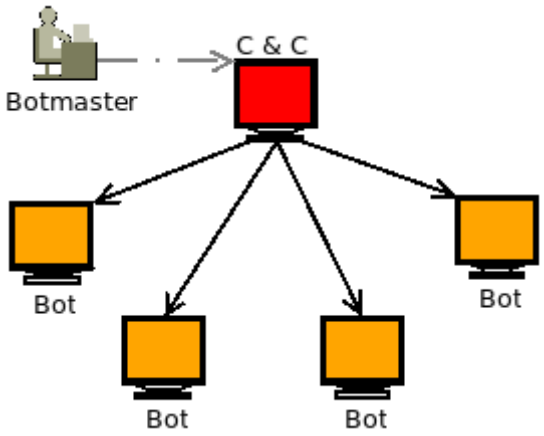

Figure 1: Single-Star Topology.

One of the advantages of such approach is that it requires no changes in bots' code (if these used to operate as a single-star topology). The only required changes are at the CnC level. This topology introduces not only redundancy at the CnC servers, but also improves scalability (allowing load balancing) and, in certain situations, may reduce message latency using the geographical position of the bots and CnC servers. One additional reason to migrate from a single-star to a multi-star topology is related to the business model of the botnet. Some cyber criminals sell or rent parts of the botnet for illegal activities, which is easier with a more modular and robust botnet topology. However, modularity and robustness come with an additional cost: complexity. Creating and managing a multi-star botnet requires additional effort and a more complex structure.

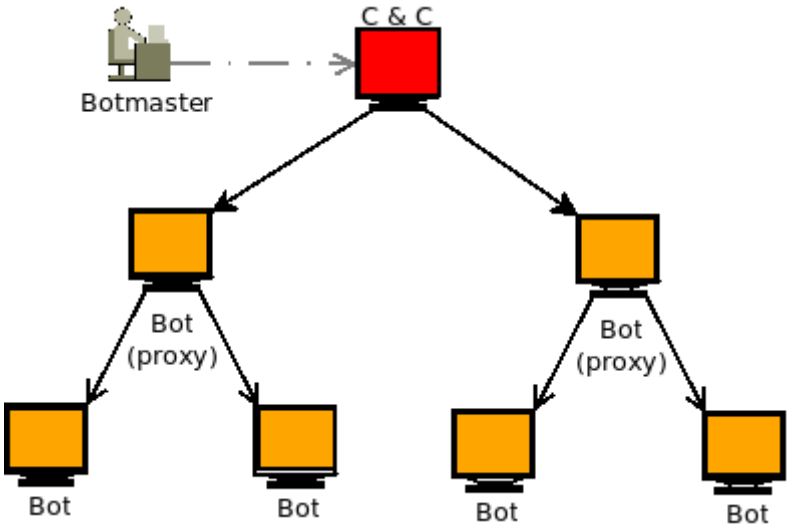

Figure 2: Multi-Star Topology.

2.1.3 Hierarchical

The hierarchical topology is an intuitive adaptation of the multi-star topology. In this case, instead of multiple CnC servers to manage the botnet, there is one 'main' CnC server which has some bots acting as proxy. A bot acting as proxy will also behave as a 'secondary' CnC server for the nodes that connect to it to reach the CnC centre (see Fig. 3).

One advantage of such architecture is that no single bot is aware of all the other bots in the botnet. This makes it not only more difficult to estimate the real size of the botnet, but also more difficult to control and seize the botnet, since only part of it can be reached and the address of the 'main' CnC server is only known by few bots. Furthermore, this topology makes it suitable to adapt the propagation tactics as the botnet population spreads. For instance, initial propagation techniques with drive-by download infections, that shift to worm like infections inside an enterprise network [32] (see Sec. 2.3.1). As such, this design makes it perfect to rent or sell parts of the botnet. On the other hand, bots that now have the ability to proxy CnC instructions may suffer from latency issues. Consequently, it might become more difficult to perform real-time attacks.

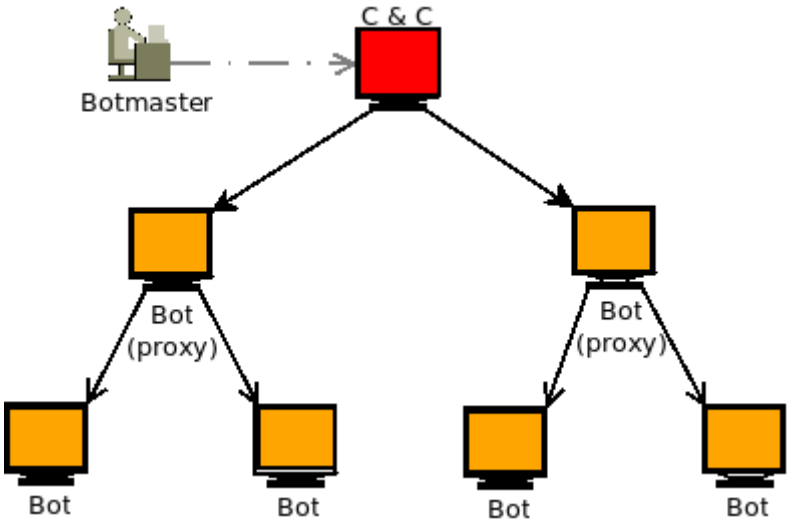

Figure 3: Hierarchical Topology.

2.1.4 Peer-to-Peer (Random)

Up to now, the topologies that were discussed focus on botnets that have a central entity, or a group of central entities, to control the botnet, but that does not always have to be the case. Botnets that follow a random, also designated decentralised topology, do not have a central CnC that issues the instructions. They typically operate as Peer-to-Peer (P2P) networks, such that each command may be delivered by any member of the botnet. As such, each bot is only aware of its neighbours (see Fig. 4).

This topology makes the botnet more difficult to analyse and study due to the lack of a central CnC server. In addition, the multi-path, ad-hoc and unstructured characteristics of P2P networks make it very difficult to determine the size of the botnet. Two big disadvantages of the lack of structure in P2P botnets are the message latency and its complexity. Although, message latency can be mitigated by creating multiple communication links among the botnet members.

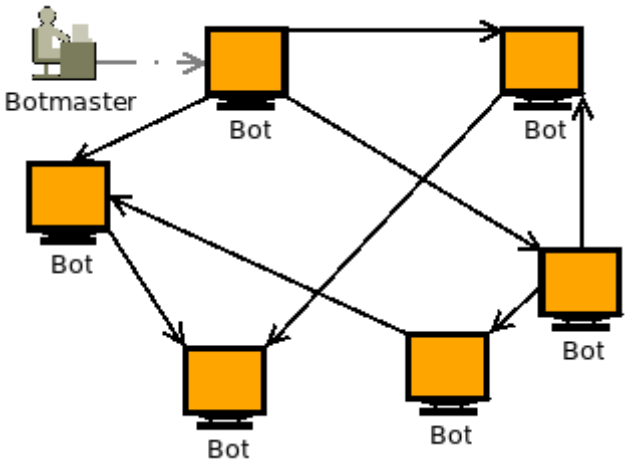

Figure 4: P2P Topology.

2.2 Communication Protocols

Different botnets use different protocols for their CnC communication [18]. This makes the detection, prevention and research of botnets more challenging. Moreover, botmasters tend to use regular communication protocols and applications, and try to manipulate them in unexpected ways to achieve their goals. Also, as the CnC communication might be disguised as regular and benign traffic, it might go unnoticed. Nevertheless, cyber criminals still have to circumvent security measures which benign applications do not. Some of these circumvention techniques will leave traces that provide enough information for botnet detection.

Almost any kind of protocol can be used for CnC communication. Common options include IRC, HTTP, DNS or P2P. Other communication protocols have also been used but not so commonly, for instance IM or Skype.

IRC: Internet Relay Chat (IRC) was one of the first methods used by botmasters to control their botnets. IRC operates in the client-server model, in such a way that bots will register at the server (controlled by the botmaster) and wait for commands. The simplicity, flexibility and the availability of open-source/free IRC client and server software makes this protocol very appealing for attackers. However, botmasters started to migrate to different protocols as IRC botnets became more and more known, and due to the fact that IRC communication can be easily blocked.

HTTP: Hypertext Transfer Protocol (HTTP) is another communication protocol used by botnets. The clients (bots) will contact a HTTP server and the botmaster will reply with the command to execute (pull method). The commands issued can then be blended in regular HTTP traffic in order not to raise suspicion. More recently, alternative designs have also been seen in which the botmaster publishes the command on a public available website that allows the user to upload some form of content. Later on, when the individual bots connect to the Internet they will check for recently published commands (push method[2]). This new approach is also known as Web 2.0-based attacks [23].

[2] In other variant, the botmaster can also push the commands directly to the bots.

DNS: Domain Name System (DNS) can also be used to propagate CnC commands. Botmasters achieve this by hiding commands inside normal DNS traffic. DNS servers which follow the specification should not notice anything abnormal with the DNS requests, while the botmaster controls a malicious DNS server which will know how to interpret the covert message. This technique is also designated as DNS Tunneling.

P2P: Some botnets make use of Peer-to-Peer (P2P) decentralised communication protocols to propagate the new commands to the botnets, for instance, Storm Worm botnet [26]. How these protocols work may vary, as some botnets use open source P2P implementations or may create their own P2P protocols.

It is important to notice that even if the afore mentioned communication protocols do not use encryption by default botmasters are able to encrypt their CnC traffic by using HTTPs or any other kind of encryption on top of the protocol specification. Furthermore, unlike IRC which can be easily blocked, HTTP and DNS traffic cannot be completely blocked as they are of paramount importance to the operation of the Internet.

2.3 Life-cycle

Botnets propagate themselves across the Internet using similar approaches to other malware. They can propagate like a worm or might mask themselves like a Trojan. Even though botnets might be very heterogeneous, typically they all follow the same steps throughout their life-cycle, namely propagation & injection, control, attack and update (see Fig. 5) [23]. In this section, we will discuss each of these stages individually and how they relate to one another.

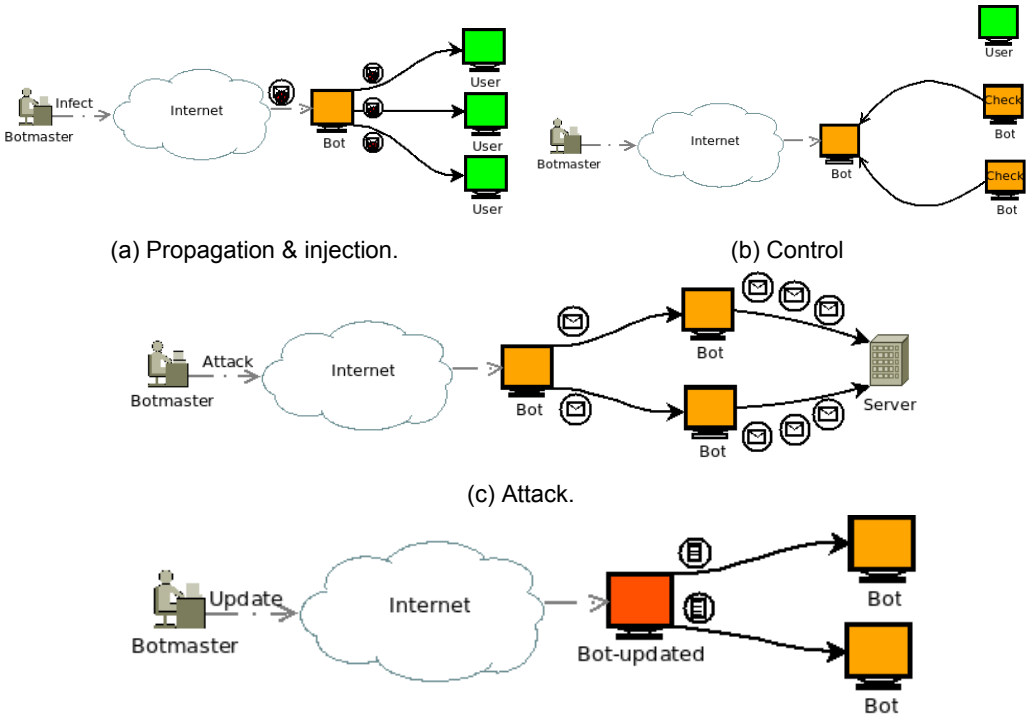

Figure 5: Botnet life-cycle overview.

2.3.1 Propagation & Injection

When designing a botnet the botmaster has to take into consideration how he will infect the victims' machines. As most users would not be willing to cooperate with such activities, botmasters have to come up with a way to infect machines which can only be remotely accessed and are very heterogeneous. Consequently, attackers will try to explore vulnerabilities such as backdoors left by previous malware, bugs on the operating system or other vulnerabilities present in the software running on the host.

Recently, a shift has been seen that implies that botmasters are evolving from exploiting a single vulnerability to exploit several known vulnerabilities that might be present [7]. This extends the propagation vectors and increases the chances of infecting the targeted machine. Furthermore, botnets seem to be able to more precisely aim at which machines to infect instead of a random/brute-force scanning. Such list might be related to the specific applications running on the targeted host or the specific users that access that host [7].

Target	Design Complexity	Detectability	Propagation Speed	Population Size
Operating System	Medium	High	Low	High
Services	Medium	Medium	Medium	Medium
Applications	High	Low	High	Low
Social Engineering	Low	Medium	Low	High

Table 1: Propagation Methods (adapted from [7])

Summarised in Tab. 1, one can observe how the propagation vectors (targets) relate with the design complexity, detectability, propagation speed and size of the botnet. Thus, one can analyse how the choice on the propagation methods influences several characteristics of vital importance for the botnet life-cycle. For instance, a botnet that propagates through the use of social engineering has does not require a complex design but its propagation speed is slow. On the other hand, targeting applications might allow a botnet to propagate faster (automatic exploit of an application vulnerabilities is faster than social engineering), but its population is dependent on the number of hosts running a vulnerable application for a known exploit.

As afore mentioned, botnets have different means available to propagate across the Internet. Such means include the use of malicious emails, vulnerabilities left by other software or by other botnets, file sharing, instant messaging and others. Some of these techniques might require user action, in which case the botmaster attempts tricks the user to download or install some piece of software, making its behaviour similar to a Trojan. It is also possible that the propagation and infection happen totally automatically by scanning the networks for known vulnerabilities in the software, weak passwords or misconfiguration of certain services. When such vulnerability is found the possible future bot will attempt to download a malicious binary which will try to install the software required to control the host.

2.3.2 Control

As the bot finishes to download and install the malicious binary, it is time to check-in with the botmaster using the communication channel used by such botnet (see Sec. 2.2). The botmaster will be informed of the successful infection and the bot will wait for further instructions on how to proceed.

2.3.3 Attack

Having the individual bots under its control, the botmaster can now issue commands to the botnet. Botnets can be used for a wide set of malicious actions, such as create a Distributed Denial-of-Service (DDoS), generate and send spam, phishing, malware/adware propagation, espionage, hosting of malicious websites or content, and proxy activities.

DDoS: These attacks consist of contacting a target server with a large number of machines at the same time, in order to disrupt the server activity. Botnets provide an easy way to control many machines simultaneously, while remaining anonymous.

Spam: Botnets are used by spammers due to the ability to quickly and easily change among different legitimate IP addresses that are used to send out the spam messages.

Phishing: Similar to spam, cyber criminals can benefit from the IP 'agility' provided by botnets, such that if a phishing website is taken down a copy of it can be brought back online almost instantaneously.

Malware Propagation: Botmasters are always interested in expanding the size of the botnet. As such, malware propagation is a common use for botnets.

Espionage: Bots are able to sniff information on the network and from the host machines. This means that botmasters are able to collect important and sensitive information such as passwords, user accounts or any other kind of sensitive information accessible to the bot. This information can then be sold, used to generate profit or for identity theft.

Hosting Malicious Content: Due to their resilience, botnets can be used to host malicious content, such that it becomes harder for the legal authorities to find and remove the content. The botmasters cannot only profit from hosting the content but also from advertisement related to the hosted content.

Proxy: Each bot can be used to proxy the botmaster commands. This gives the ability to the botmaster to be always protected behind a proxy, such that the botnet commands do not give away his location.

2.3.4 Update

After executing the botmaster commands each bot awaits in an idle state for new commands. These commands might include new target systems or a new update to the bots. Bots are in constant update, either to pass undetected on anti-virus scans, improve propagation vectors with new vulnerabilities and methods, to fix any possible bug in the code or to improve robustness and modularity in the botnet.

2.4 Botnet Detection

Botnet detection techniques can be split in two big categories: honeypots and passive network monitoring [48]. Current mechanisms to detect botnets have to circumvent the most recent defensive mechanisms implemented by botmasters. To do so, one can use DNS, Netflow, time-zones, packet inspection, URL inspection, etc.

In this section we provide an overview over the different techniques used to detect botnets and the basic defensive mechanisms deployed by botmasters to frustrate detection. As the main focus of this research lies in detecting botnets using pDNS traffic we provide a more extensive overview over such techniques in Sec. 8 [3].

2.4.1 Honeypots

Security researchers deploy honeypots with the intention to use them to collect information on botnets, such as the origin of CnC communication, botnet membership or attack behaviour. Honeypots are machines which are purposely deployed such that they are vulnerable to malicious attacks and have high chances of becoming infected in a short window of time. Often honeypots are deployed with software such as honeywalls, which can be used to analyse, log and control traffic in and out the honeypot [47].

To this end, researchers deploy honeypots with a wide range of known vulnerabilities, such that these become easy to compromise. After infection, the bot behaviour can be easily monitored and used to extract information on the botnet. For example, using the honeywall to inspect the network traffic, one can try to identify the botnet CnC IP address or domain name. Furthermore, by monitoring all outgoing and incoming connections to the bot it might be possible to identify other compromised machines, to estimate the size of the botnet, study the botnet life-cycle, etc [34]. The added value of honeypots is related to the fact that it allows us to execute the bot code and monitor the behaviour from a known and controlled environment. Although, as honeypots and honeywalls are managed by security researchers and legal authorities, their actions are bound by the existing legal frameworks. That said, honeywalls should be implemented such that they block all malicious outgoing traffic in order to not compromise other machines. However, attackers are aware of such constraints and can modify their bot behaviour such that it behaves differently whenever it detects that it is been running from a honeypot [45].

Honeypots play an important role in botnet detection as they are a source of information by passive monitoring techniques, namely to analyse and synthesize the signatures used by signature-based techniques (see Sec. 2.4.2). Even though honeypots may be used for detection purposes, its usage makes it ideal for tracking and measurements purposes [47]. Examples of honeypot-based detection techniques may be found in [1, 6, 8, 16].

2.4.2 Passive Monitoring on Network Traffic

Another approach used for botnet detection consists of passively monitor Internet traffic and extract/identify network flows related to botnet activity. There is a wide range of traffic which can be monitored to detect bots presence in the network, such as DNS, Netflow, address allocation data, proprietary enterprise data and others [47]. Such techniques can be further classified as signature-based, anomaly-based, DNS-based and data-mining based approaches [18].

Signature-based: Signature-based botnet detection relies on the knowledge of known and previously studied botnets. Once a botnet is discovered, researchers try to analyse the bot code and behaviour in order to produce signatures which can then be used by an Intrusion Detection System (IDS). A signature is a collection of information which contains all the relevant details of an infection or malicious communication. It comprises information such as

[3] Network-based defensive mechanisms of botnets, such as FF and DGA, will be discussed in Sec. 3.

file names, URLs, sequence of bits or any piece of information that might be useful to detect botnets. The IDS will then inspect the payload content and raise an alert whenever a signature is matched. Snort [4] is an example of such IDS.

The main shortcoming of this technique is that it requires knowing the bots' signature to detect it. Hence, it can it can only be used to detect known and previously researched botnets. Examples of signature-based detection systems include Snort and Rishi [21].

Anomaly-based: This technique is based on the identification of 'abnormal' behaviour, in this context 'abnormal' refers to everything that deviates from the 'normal' and regular behaviour specified in the appropriate templates, such as RFCs [42]. As previously mentioned, typical botnets uses include DDoS attacks and spam messages. As such, botnets should often generate high volumes of traffic and increase networks' latency. Such characteristics and others (such as unusual port numbers, unusual number of connections) can be considered 'abnormal' and be used to detect botnet infections.

Examples of anomaly-based detection techniques may be found in [11, 22, 27].

DNS-based: DNS-based techniques for botnet detection are a hybrid between behaviour-based and data-mining based detections on DNS traffic. Botmasters use DNS in order to manage their botnet. Furthermore, botmaster take advantage of DNS agility to hide their bots and frustrate take down efforts. As such, DNS queries are performed throughout the botnet life-cycle and can be used to differentiate botnet DNS queries from legitimate ones. Examples of DNS techniques include [17, 37].

Passive DNS is a replication technique that passively captures DNS messages and allows the storage of such data. A more detailed explanation of pDNS can be found in Sec. 3.3. In addition, Sec. 8 contains a more detailed overview of pDNS-based detection techniques.

Data-mining based: Even though identifying botnet CnC communication is an effective way to detect botnet activity, it is difficult to separate it from regular traffic. This is due to the fact that botnets typically use regular protocols for their CnC communication and that it generates only a small amount of traffic. Furthermore, botnets are in constant update, which may reduce the efficiency of signature and anomaly-based detection systems.

To overcome this, researchers have been exploring the possibility of applying data-mining techniques to the network traffic to detect botnet communication. Such techniques include clustering, classification, regression, pattern recognition, etc [40, 47].

2.4.3 Obfuscation and Deception Mechanisms

Botnets have several defensive mechanisms to frustrate detection and take down efforts. In this section we will provide an overview over different techniques [9].

Render AV ineffective: In order to avoid being detected by the host anti-virus, bots try to disable the anti-virus as soon as they assume control of the target machine. Different methods can be used for such purpose, raising different levels of attention. A bot can download an 'anti' anti-virus tool from the CnC centre to prevent the anti-virus from executing, but that could raise too much undesired attention. Other solutions include the use of a DLL to make the anti-virus ineffective or prevent the anti-virus from updating its vulnerability database.

[4] www.snort.org

Rootkits: Rootkits are a set of tools that the bot will try to download upon a successful infection and provides the bot with the necessary means to hide himself from the operating system.

Reduce host security: Upon installation, the bot code may try to reduce the security rules in the firewall and try to disable any other sort of security software that might be used to track it down.

Anti-debug & anti-virtualization: Some bots have mechanisms that will trigger whether the bot binary is being run on debugger or on a virtual machine. These mechanisms are designed to mimic non-malicious activity and, in that case, to frustrate analysis efforts from legal authorities and researchers.

Polymorphism & metamorphism: These two techniques are devised to circumvent signature-based detection. With polymorphism, each time the bot binary propagates it will re-encrypt itself such that it will exhibit a different signature each time. With metamorphism, instead of encryption the code re-writes itself slightly different each time, such that its signature also changes and is able to avoid detection.

DNS: Botnets do not use DNS only for their CnC communication but also to frustrate detection efforts. By using small values of TTL for each DNS record, botmasters gain the ability to rapidly change the IP addresses of CnC servers or changing the domain name to which an IP address is pointing to, for instance using Dynamic DNS [44]. Such techniques will be further discussed in Sec. 3.

3 The Domain Name System

The Internet as we know it today is highly dependent on the DNS. Even though computers are able to operate on the bases of IP addresses, human users find it less intuitive. In this section we provide an overview over the DNS, how it operates, how it can be abused and what kind of information we can collect in order to detect malicious usages.

3.1 An overview

The DNS is a distributed naming system that provides a mapping between IP address and domain names. In fact, DNS also stores additional information related to domains, such as Authoritative Name Server (ANS), domain aliases, mail exchanger, etc. Each piece of information is stored in its own Resource Record (RR), for instance A, AAAA or NS. Each RR is used for a different purpose, namely the A RR is used to obtain the IPv4 addresses associated to a domain name, the AAAA RR returns IPv6 addresses and the NS RR returns information related to the Name Server (NS) of a given domain name.

DNS organises namespaces in a tree structure, in which the root node is represented by a single dot. Each node of the tree hierarchy represents a level, is associated to a domain name and has zero or more associated RRs. For instance, in the domain name 'www.example.com.' we can observe three distinct levels: the Top Level Domain (TLD) 'com.', the Second Level Domain (2LD) 'example.' and the Third Level Domain (3LD) 'www.'. Additionally, the 2LD 'example.' can also be referenced as subdomain of 'com.'. Alternatively, one may look at DNS in terms of zones, starting at the root zone ('.'). Each zone contains an ANS, which is responsible to resolve domain

names under that same zone. However, an ANS might not have information on all domains and it might choose to delegate the authority of its subdomains.

The process of looking up a domain name in the DNS is also referenced as *resolving*. When queried, the DNS server of each ANS will return the addresses under its authority which are closest in the DNS hierarchy to the desired domain name. With the same domain name as example, the ANS responsible for the *root* zone might not know the exact address of 'www.example.com.' but knows the address of the ANS in charge of the TLD 'com.' . Analogously, the ANS responsible for the TLD may know how to resolve 'www.example' or might only have the address of the ANS in charge of the 2LD, 'example.'. This process is also designated as *recursive resolving*.

In short, DNS provides a distributed mapping between IP addresses and domain names which is modular, robust and has no central entity. Although, the resolving mechanism, as aforementioned, requires too much effort on the ANSs. A solution to circumvent this drawback, that also reduces the amount of DNS queries on the Internet and improves the overall performance, is the use of caches in DNS servers. These servers store the DNS query results for a determined amount of time specified at the domain level as Time-to-Live or TTL. After the TTL value expires the cache entry becomes invalid. As such, the resolver does not have to start recursive queries at the root zone each time a domain name is queried. Instead, it can simply query a DNS cache server that will return the desired answer if found valid in its cache. If there is no valid entry for that domain name, the DNS cache server will proceed to recursively resolve the domain name, update its cache and return the results to the client.

3.2 How can it be abused

In the previous section, we provided an overview over the DNS and its most common use: to translate user-friendly domain names to IP addresses. Although, the DNS overview discussed in the previous section contemplates only a simple scenario, in which each domain name is resolved solely to one IP address. For reasons such as load balance or robustness it is beneficial to have one domain name resolving to multiple IP addresses. Hence, it is common that each domain name may have more than one entry for each RR type, in which case each entry is returned in a Round-Robin fashion. Some other uses were also reported in Sect. 2.2 and 2.4.3, such as DNS Tunneling and DNS as CnC communication.

The designed use of DNS is defined in RFC documents. Well-known and widely-used applications include:

- Management of domains blacklists (RFC 5782)
- Key management (RFCs 4034, 4025, 2535, 2930, 2230, 4255)
- Universal Plug and Play (UPnP) (RFC 3927)
- Email validation (RFC 4408)
- Email resolvers (RFC 1035)

Other examples include the use of DNS for incident reporting, used by some anti-virus vendors, or to query domains blacklists. Nevertheless, DNS may also be used for malicious purposes which are not documented in RFCs, such as DNS spoofing, creation of fake websites for advertisement and CnC communication.

In order to frustrate detection efforts, cyber criminals devised certain evasive techniques that are based on the fast change of DNS records. Such techniques allow the attackers to rapidly change the DNS information of their machines such that they are constantly changing IP address and changing the domain name they connect to. Examples of such techniques are Domain Generation Algorithm (DGA) and Fast-Flux (FF).

Botnets may use DNS at several steps throughout their life-cycle [15]. Such steps are summarized below:

Rallying Procedure: Upon successful infection, the recently converted bot will have to check-in with the main CnC centre and as such, it requires to know its IP address. The IP address of the main CnC server might be hard coded or the bot might resort to the DNS in order to find it. In the former case, the botnet is somehow limited in the way it is managed due to the lack of flexibility (re-allocating the CnC centre would require to notify and update all the bots in the botnet). Furthermore, security researchers and legal authorities might be able to extract the static address from the code. In the latter, at least one DNS query has to be generated which can be used to obtain some information on the botnet.

Malicious Procedure: Malicious activity, such as DDoS and spamming generate several DNS queries that can also be used to extract information on the botnets behaviour.

CnC Link Failures: Whenever there is a network or link failure that restrains a bot to communicate with its CnC server additional DNS queries will be generated in the attempt to restore such connection. Such queries provide important information on the botnets communication protocols and behaviour.

CnC Server Migration: Each time the botnet re-allocates the CnC server the bots will have to execute DNS queries to re-connect to it.

CnC Server IP Change: If the CnC centre of the botnet uses Dynamic DNS [44], the botmaster is able to change its IP address at will. Bots will then resort to DNS in order to find the new IP address of the CnC centre.

3.2.1 DGAs

A common way to prevent CnC communication is by implementing a static blacklist. With such technique, each time a domain name is queried it will be tested against the blacklist to detect if it was associated with any illegal activity, such as botnet, spam, phishing, etc. As such, this technique requires the domain names associated to the CnC server to be known. In order to circumvent this static measure, botmasters devised what it is known as Domain Generation Algorithms (DGAs). By using DGAs with a centralised topology (see Sec. 2.1) botmasters achieve a balance between simplicity of a centralised CnC and the robustness of P2P networks. In addition, DGAs can also be used with other botnet topologies.

Each time a DGA bot wishes to contact the botmaster it generates an extensive list of randomly generated domain names. Each randomly generated domain name is a candidate for the botnet CnC centre of the botnet. From this list, the bot will try to resolve each domain name. Using the same method, the botmaster is also able to generate the exactly same list of candidate domain names, pick and register just a few for actual CnC communication. Only this small set can then

be resolved by the bots. In addition, each registered domain name will only be active for a small amount of time (small TTL value).

Identification and control of DGA botnets is not easy. The randomly generated domain names might be spread across different TLDs, registration authorities and countries, making legal measures more difficult to apply. Moreover, each domain name will be active only for a short period of time. Also, the fact that more than one domain name is used for CnC communication at each moment, means that researchers and legal authorities have to be able to register all of them before the botnet controller, if they wish to seize the botnet completely. As such, ideally one would try to reverse engineering the DGA code to be able to generate the domain names in advance and block the CnC traffic. However, reverse engineering is often time-consuming, expensive and not always feasible due to the botnet constant updates and deployed obfuscation techniques. Therefore, other techniques must be developed to be able to control the raise of DGA botnets.

3.2.2 Fast-Flux

The goal of FF is for a domain name to be associated with many IP addresses [14]. Botmasters may hide their CnC centre with this technique and use the bots to proxy the requests. The CnC centre is associated with a domain name which will be resolved to a different IP address each time. As such, the same IP address is never used for long periods of time and reduces the number clients connecting to the same CnC server, drawing less attention to it. To achieve this, botmasters use bots to act as proxy to the CnC centre that can be added or removed from 'flux' at will. Furthermore, taking advantage of Round-Robin DNS and small values of TTL the botmaster is able to manipulate the DNS records such that each time a certain domain name is requested it resolves to a different IP address[5] (see Fig. 6). Unfortunately, it takes too much time for legal teams to be granted permission for effective take downs.

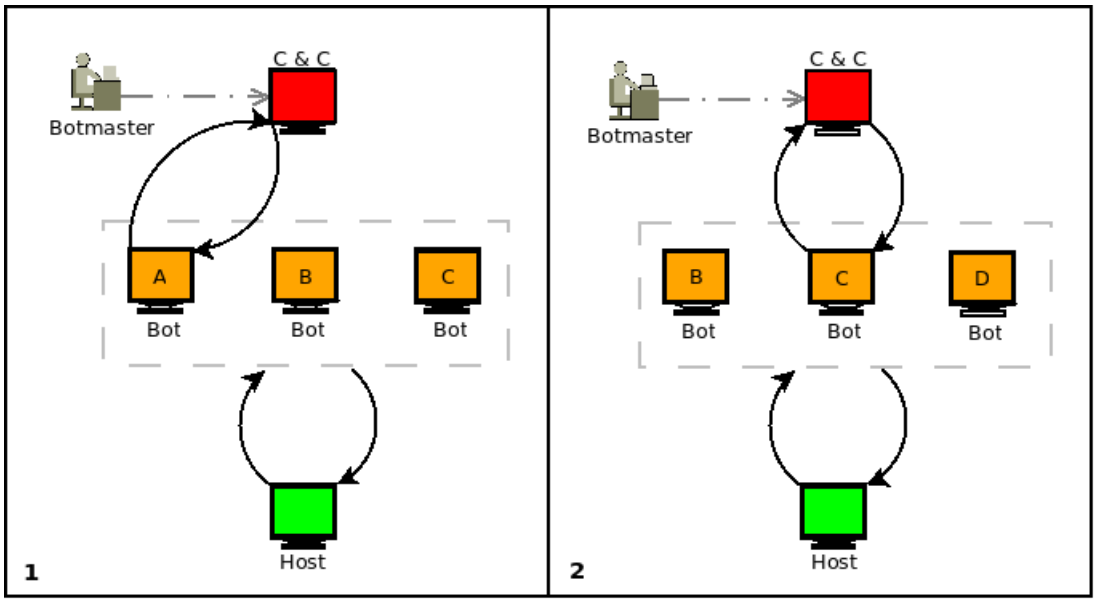

Figure 6: Fast-Flux network: all nodes inside the dotted grey box correspond to the same domain, and at each request a different node (with different IP address) is used to proxy the communication. Notice that bot 'A' was removed from the flux by the botmaster and a new node 'D' was added.

[5]The small value of TTL implies that the DNS cache is updated much more frequently, reducing its advantages.

Double Fast-Flux is a more sophisticated type of flux and works similarly to single Fast-Flux [14]. It further frustrates the take down efforts by additionally changing the IP address of the NS associated with the domain used for CnC traffic. In addition, the NS records also have a low value of TTL, but not as small as the TTL value of the A records. This is due to the restrictions imposed by the TLD registers. In this technique, all the IP addresses associated with the botnet CnC are continuously being fluxed.

Additionally, there is a third variant called Domain Fast-Flux [9], often found together with DGAs. In Domain Fast-Flux the idea is to rapidly change the domain name associated with the CnC server. It is usually used to reduce the focus of security researchers over a particular domain name. Typically, there is a list of domain names which will be used to flux, this list can either be downloaded by each individual bot or it can be generated in real-time using DGAs. This technique has the additional advantage that each domain name is not active for long periods of time, hence raising less suspicions over the domain.

Fast-Flux techniques allow botnets to evade detection more easily and increase availability. Furthermore, it provides load balancing, redundancy and frustrates take down efforts. These attributes make FF very appealing for non-malicious websites as well, specially for high volume websites such as search engines or CDNs.

3.3 Passive DNS

DNS resolvers do not store DNS information other than the one which is stored in the DNS cache. As the TTL values of each entry expires, their information is no longer valid and new one needs to be fetched. Likewise, ANS are meant to only store DNS information which is still valid and as new information is associated with a domain name, the previous one is discarded. Let us consider a domain with a TTL value of five minutes. Every five minutes all the local information related to that domain is discarded and new one has to be fetched to resolve that domain. At higher levels in the DNS hierarchy, whenever new information is available and submitted the old one is discarded. These changes will then be propagated to the lower levels as the TTL values of the DNS caches expire and the DNS resolvers attempt to retrieve new and valid information. This implies that, with exception for the period time it takes to propagate the DNS information, there is no information on the domain history.

In 2005, Florian Weimer presented Passive DNS (pDNS) to address this concern [46]. A pDNS database contains historical DNS data and it is built by passively listening to DNS queries and answers. It provides an easy, fast and non-intrusive way to access historical information that it is no longer present in the DNS records. In addition, DNS allows only one type of queries: giving a domain name and the desired record type, fetches all the information available. The fact that pDNS is built on top of a relational database and contains historical data provides a much wider range of queries to perform.

As previously mentioned (see Sec. 3.2), botnets use DNS for their CnC information. As such, we would like to identify traces of that traffic while analysing DNS information. However, malware is usually identified after the necessary DNS queries have been performed. Due to the DNS agility achieved by botnets using the techniques mentioned in this section, researchers and legal authorities have a tiny window of time while the DNS information will remain without updates. Passive DNS allows to extend this windows of time by storing all the domain history. In addition, pDNS information allows one to access information which was previously unknown, such as the list of IP addresses that a domain name ever resolved in its life-time.

4 Data Collection

The pDNS data used in this research project was collected and provided by Fox-IT [6] in Delft, the Netherlands. In order to passively collect DNS data Fox-IT deployed the code presented in [19] on a Dutch DNS server. This passive DNS tool [19] outputs the pDNS data with the following fields: timestamp (moment when query was detected), DNS-client (originator IP address), DNS-server (responder IP address), query class[7], query (domain name), query type (RR), answer (depends on RR), TTL and count (number of times the same query-answer pair was seen). Whenever a new DNS request and answer is received, it is written to the log file and stored in cache. Such an entry will not be written again during the next 12 hours unless its value changes or if the logging tool runs out of memory. Furthermore, after 24 hours without activity the entry is removed from the cache and written to disk. Each entry is kept in the cache so that results can be aggregated, avoiding the 'flooding' of the log files with information about each query. It is important to notice that due to the fact that the data is aggregated there is the chance that data useful for detection is lost. Data in the cache will be aggregated as follows:

Timestamp : The oldest timestamp of each entry is the one written to the log file.

TTL : The highest TTL of each entry is the one written to the log file.

Count : This values increases each time the same DNS question and answer are observed.

Fox-IT further aggregates and anonymizes this data by week. As a consequence, each entry on the pDNS logs contains the following fields: query, query type, answer, first seen (within the week), last seen (within the week), maximum TTL (within the week) and the count of how many times the same entry was seen (within the week). Afterwards, we parse these logs and insert them in a PostgreSQL database [8]. During the insert process, each entry on the logs will either generate a new entry in the database or update an existing entry. In the latter case, data is updated such that the last seen and TTL have the absolute maximum observed value within the observation period, while the first seen the minimum. The count value is updated such that it contains the number of times that each pair <query, answer> is seen. The pair <query, answer> is also used as unique identifier of each entry. For performance purposes, we have two additional fields in our database that contain the TLD and 2LD of the query.

Data used in this research represents data collected over a period of two weeks, more precisely the first two weeks of May 2014. This data contains more than 27,000,000 unique domain names, spread over 5,800,000 2LD and 397 TLD.

5 Classification

In this section we provide some background on machine learning and on the different supervised learning algorithms used to build our classifier.

[6] www.fox-it.com.
[7] Refer to RFC 1035.
[8] http://www.postgresql.org/

5.1 Background

Machine learning is the field of computer science that develops ways that allow systems to learn from data. Most of the machine learning approaches have as primary goal generalization: ability to accurately perform on new and previously unseen examples, based on what was learned from the training set. Machine learning and data-mining are two close scientific fields, but have one big difference: while machine learning uses known properties extracted from the training sets, data-mining focus on extracting this features (previously) unknown from the training sets.

There are many different machine learning algorithms, which can be classified in three main groups: supervised learning, unsupervised learning and reinforcement learning. In the case of supervised learning, each sample in the training data contains the correspondent target value (labelled samples). If the goal is to assign to each input sample a category (target) from within a finite set of categories, the task is called classification. On the other hand, if the goal is to obtain a description of one or more continues variables, then it is a regression task. In unsupervised learning, samples from the training set do not possess a target value and can be used for a diversity of tasks, such as clustering, density estimation or visualization. At last, reinforcement learning is related to tasks in which the goal is to come up with a set of actions that allow to maximize some notion of reward [12]. It is also possible to combine supervised and unsupervised learning algorithms, which results in semi-supervised learning. The goal of such algorithms is not generalization but to grasp the structure of the data from samples in the training set without target values.

For the purpose of this research, we used a machine learning library for python called scikit-learn [38]. This library provides several different machine learning algorithms and an extensive API for test and results analysis.

5.2 Algorithms Overview

In this research we use two different supervised learning approaches to solve our problem: k-Nearest Neighbours (kNN), Decision Tree classifiers and Random Forests. The scikit-learn library provides two additional approaches that could also be used as classification tasks in a supervised learning setting, namely Naive Bayes methods and Support Vector Machines [39]. However, in initial experiments the classifiers built with such approaches had an overall performance inferior to 50% of correct predictions. These results lead us to exclude the use of such algorithms in our experiments.

kNN is a clustering algorithm that, in a binary classification problem, attempts to build clusters of positive and negative samples using the labels from the data in the training set. On the other hand, Decision Trees approach the same problem creating a tree in which each node is made of a decision that allows to split the data into smaller groups. The Random Forests algorithm consists of running several Decision Trees, randomizing some of their parameters, to build a more robust model. The remainder of this section contains an overview over the algorithms

5.2.1 k-Nearest Neighbours

The k-Nearest Neighbours (kNN) algorithm can be used for both classification, regression and clustering tasks. In our scenario, each sample in the training set has the category (or label) to which it belongs, either positive or negative. Since we our goal is to be able to classify unknown

samples according to those categories, we will use the kNN algorithm as a classification task. In addition, each sample in our training set contains the 36 feature values that are associated with a determined domain (a detailed overview over the extracted features can be seen in Sec. 6.1).

The k-Nearest Neighbours (kNN) algorithm considers each sample of the training set as a point in the n-dimensional space $\mathbb{R}^n$, in our case $n = 36$, the number of features. The idea behind the algorithm is to find the k points in the training set that are closest to the point we are trying to classify. After identifying the k nearest neighbours, each neighbour will 'vote' with its category to determine the category of the sample to classify. The category with the majority of votes, is the prediction of the classifier for the given sample. It is during the training phase that our classifier will learn the points to be used during the classification task. These points are the representation of each sample of the training set in the n-dimensional space. There is no specific rule on how to specify the method used to calculate the closest points, as in most cases in the literature we will use the standard Euclidean distance [30]. This method falls into the category of non-generalizing machine learning algorithms because instead of using any generalization rule it simply stores all samples in the training set to later be used during the classification process.

When considering the kNN algorithm for a classification task there are three main parameters to take into account: the metric used to calculate the distance between two points, the value of k that corresponds to the number of neighbours to consider and the weight of each vote.

Calculate distance: We will use the Euclidean distance to measure the distance between two points, which is calculated as follows:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}, p = (p_1, ..., p_n) \; and \; q = (q_q, ..., q_n)$$

Value of k: The value of k is very important as it defines the number of neighbors that will take part in the 'voting' process. A value too small and the classifier becomes very sensitive to noise, a value to big makes the boundaries between classes less definite. The value of k depends on the data to be classified and must be adapted according to it, during the training stage.

Weights: We will evaluate two different scenarios: one in which all the votes have the same weight and a second where each vote is weighted based on distance from the sample to classify, such that further away neighbours have a smaller weight.

5.2.2 Decision Trees

Decision Trees create models that allow to classify a given sample by creating a set of decision rules that are extracted from the feature sets of the samples in the training data. Each node on the tree can be seen as an if-then-else decision node. The conditional test used is made over the possible set of features and their respective range of values. As such, to classify a given sample one starts testing the feature value of the root node and proceeds down the tree following the branches correspondent to the value for that feature. This process is then repeated for the sub-tree that starts at the new node, until finally reaching a leaf node. The classification result is given by the classification of the samples from the training set that belong to the same leaf node. A Decision Tree can also be thought of as a disjunction of conjunctions over the tests on the features values of the samples. Each path from the root to the leaf node represents a conjunction of these constraints and the tree as whole is the disjunctions of such paths [30].

Decision Tree algorithms implemented in the library scikits-learn implement the CART (Classification and Regression Trees) algorithm. This algorithm creates binary trees that at each node use the feature and threshold of the feature value that yields the biggest information gain [39]. The process of generating the tree will stop whenever the CART algorithm is not able to detect additional gain or when the tree reaches a predefined depth. In order to avoid the tree to overfit the data, by describing noise instead of the properties present in training data, we vary certain parameters of the tree that allow us to prune it. Such parameters will be described below:

Maximum Depth: This parameter allows us to stipulate a maximum depth for the tree to grow.

Minimum Samples per Split: This parameters defines the minimum number of samples required to perform a split. If a node has less samples than the specified value, that node will become a leaf.

Quality of split: Previously we explained that the CART algorithm picks the feature and threshold of the feature value that yields the biggest information gain. This information gain can be measured using two different criteria using our python library: the Gini impurity or entropy. We will use the Gini impurity which can be calculated as follows:

$$G_I(f) = \sum_{k=0}^{K} f_k(1 - f_k), k \in \{0\ (benign), 1\ (malicious)\}$$

Where f_k is the the ratio of items labelled with value k in the set. This measure encourages splits in which a large proportion of the samples are assigned to only one class [12].

5.2.3 Random Forests

In addition, we also run the algorithm Random Forests. This algorithm creates several Decision Trees and each tree will be trained on a subset of the full training set. The samples to be included in each training set are selected using sampling with replacement (where a set may contain an element more than once) [5]. Furthermore, at each node only a random subset of features is available, in order to avoid that the trees become correlated. The result of predicting a given sample is then given by the average prediction of the individual decision trees [39].

The parameters used in this algorithm are the same as in Decision Trees. In addition, we can control the number of Decision Trees (estimators) which will be created. The purpose of using several Decision Trees is to improve the robustness and generalization ability of the algorithm. The idea is to create several inferior models (due to the randomness introduced) and combine them by averaging the results to produce a more powerful model.

6 Analysis

We have described in what way botmasters use DNS to manage their botnets and how it differs from regular and legitimate uses (see Secs 2.4 and 3). We can use this knowledge to devise a set of features that reflect these differences and that can be extracted from our pDNS data. This set of features can then be used to create a classifier to classify domain names as 'benign' or 'malicious'. For simplicity, we will refer to domains associated to malicious activity as 'malicious domains', while the remaining domains will be referred as 'benign domains'.

In this section we will describe our feature set and explain why these features can be used to differentiate between benign and malicious domains. Additionally, we provide an overview over some important concepts of machine learning and statistics used during our experiments and support the analysis of the results.

6.1 Features Extraction

From the set of entries from each domain name in our pDNS data we were able to extract a total of 36 features that compose our feature vector, summarized in Tab. 2. The resulting feature set is comprised of two big groups of features: lexical features of the domain name (from feature F1 to F18) and network features of the domain (from feature F19 to F36).

	ID	Designation
Lexical	F1-3	1-gram (mean, variance and standard deviation)
	F4-6	2-gram (mean, variance and standard deviation)
	F7-9	3-gram (mean, variance and standard deviation)
	F10-12	4-gram (mean, variance and standard deviation)
	F13-14	Shannon entropy in 2LD and 3LD
	F15	Number of different characters
	F16	Number of digits / domain name length
	F17	Number of consonants / domain name length
	F18	Number of consonants / number of vowels
Network	F19-21	TTL, Window (first seen - last seen), Ratio (window / count)
	F22-24	Number of different first seen, answers and TTL
	F25-27	Maximum Ratio, TTL and Window
	F28-30	Mean Ratio, TTL and Window
	F31-33	Variance of Ratio, TTL and Window
	F34-36	Number of IP subnetworks

Table 2: Features extracted.

6.1.1 Lexical Features

We extracted a total of 18 features from the domain name string. The rationale behind this set of features is related to the fact that many malicious domain names look like randomly generated (particularly in the case of DGAs) and often tend to look different [28]. Furthermore, legitimate domain names are typically user friendly, composed of native words and easy to remember. On the other hand, malicious domain names are not intended for human use so do not share such characteristics.

Features F1 to F12 are meant to grasp the distribution of individual characters, 2-grams, 3-grams and 4-grams. As such, we extract the mean, the variance and standard deviation of each n-gram, $n \in [1, 4]$. This can be useful to identify domain names that look alike and domain names that were generated by similar DGAs. As afore mentioned, benign domain names tend to be easy to remember, often in native language. As such, one can expect that the features F13 to F18 will

exhibit rather different values for domain names that were randomly generated and domains that are composed of words from a native language [20].

Features F1 to F14 are based on the work presented by *Antonakakis et al* [4] (see Sec. 8.4). In addition, features F15 to F18 were based on the work presented by Frosch [20].

6.1.2 Network Features

From the information contained in our pDNS data, we were able to extract 18 network-related features. The biggest advantage of collecting pDNS data is the ability to look at the history of the domain and analyse its evolution. Consequently, the selected network features should take into consideration all the entries associated to a domain dame. To achieve such, while features F19 to F21 are calculated over the oldest entry of a given domain, the remaining network features are the result of aggregating each entry of a given domain name (F22 to F36). The rational to use the oldest entry for features F19 to F21 is related to the fact that if a domain name resolves to each IP address just during small periods of time, picking the oldest entry gives us a higher change of using an entry which has reached its maximum window (last seen - first seen).

First, we extract the value of TTL (F19), window (F20) and ratio (window/count) (F21). The majority of the legitimate websites that provide high availability services uses TTL values between 600 and 3,600 seconds [14]. On the other hand, malicious domain names tend to have smaller TTL values (F19) [10]. In addition, Dynamic DNS providers frequently used to host domains related to malicious activity also use low TTL values [43]. The window, allows us to know for how long a certain answer has been provided for a given domain. Legitimate domain names, that deploy Round-Robin DNS, continuously cycle through different machines and this set of machines does not often change. Malicious domain names used in botnets FF networks typically do not have a fixed set of machines. Over time, botnets will acquire new bots which will be introduced in the flux and certain bots might eventually be cleaned or disassociated from the botnet. As such, the window of such entries should stop growing after some time. The ratio combines both the number of times that a certain answer was provided and the elapsed time during which such answer was being provided. In the case of FF networks, one expects to have a high number of counts for small a window.

The second set of features, F22 to F24, contains the number of different first seen (F22), number of different answers (F23) and number of different values of TTL (F24). With these three features we expect to measure how a domain changes over time. When analysing a domain, we have to take into consideration all the entries that a domain possess, even if not active any more. Legitimate domains that have load balancing due to the amount of traffic received, often do so by having multiple hosts associated to each domain name. Such is the case for CDNs. On the other hand, botmasters do not want to attract attention to the generated CnC traffic, as a consequence they often cycle the IP addresses associated to the CnC servers. In addition, the number of different TTL values can be used to identify malicious domains due to the fact that malicious domains might exhibit several changes, while one could expect benign domains to remain rather unchanged. The group of features F25 to F33 contain the maximum, mean and variance of the set of features F19 to F21.

The last three features, F34 to F36 are meant to approximate the number of different subnetworks that exist in the list of answer of a given domain name. While corporate domains tend to allocate their machines within the same subnetwork or a small finite set of subnetworks, botnets harvests their bots from different locations, often randomly. As such, one can expect that domains

used for FF, and associated with botnet activity, will have answers that belong to several different IP subnetworks. To translate this information into features which can be used in our classifier, we chop each answer of a given domain into 3 different strings, using the dots that separate each octets in the IP address. Let us take as an example the IP address '101.102.103.104'. The first string would contain '101', the second string '101.102' and the third string '101.102.103'. Feature F34 will then contain the value of how many different first octets exist, feature F35 how many different combinations of first and second octets and at last, feature 36 will have the number of different combinations of first, second and third octets. Notice that considering the number of different combinations with the four octets is already considered by feature F23.

Most of our network features, with exception of F34-F36, are based on the TTL features used by EXPOSURE [10] (see Sec. 8.2). On the other hand, most of the systems presented in Sec. 8 measure the diversity in the IP addresses associated with a give domain name using information such as BGP prefixes or AS numbers. Since such information was not available in our setting we use an approximation to calculate the diversity that corresponds to the use of the features F34 to F36. Although, it is important to notice that to have a more precise measure one needs to consider the three features together. For instance, a domain name might have a high value for feature F36, which might indicate the existence of multiple subnetworks, although if the value of the features F34 and F35 is 1, we are actually looking at a $\backslash 16$ IP subnetwork.

6.2 Assembling the Training Sets

To train each of the algorithms we need to create training sets, and the quality of the predictions of each algorithm is strongly related with the quality of the training set [41]. For our experiments we will use the supervised learning setting, as such each sample in our training sets has to be labelled as either 'benign' or 'malicious', representing benign or malicious domain names respectively.

In order to find which domains are associated with malicious activity and domains that are not we created a whitelist and a blacklist. Our whitelist is composed of domain names with TLD nl., Google domains[9], Akamai domains[10]and the Alexa TOP 200[11]. Experts at Fox-IT claim that the TLD nl. has a very pro-active take down policy and requires a lot of personal information from the registrants. Both Google and Akamai are two known entities that are unlikely to host domains associated to malicious activity. The list provided by Alexa contains an overview of the most popular websites and we limit the number of domain names to be used to 200. Overall, the whitelist contains more than 370 entries, most of them 2LD. Our blacklist contain blacklisted domain names from the following public blacklists: abuse.ch (including, SpyEye, Zeus, Palevo and Feodo), malwaredomains.com and SNORT rules of blacklisted domain names. Overall, the blacklist contains more than 18,000 entries. In an initial phase we also considered the use of sinkholed domains present in our pDNS data, such as the Microsoft sinkhole. By directly querying our database for sinkholed domains we have a fast way to collect information on domains related to malicious activity. However, as soon as a domain is sinkholed its DNS information is updated and most of the 'malicious behaviour' will fade away. It is important to notice that we only have two weeks of pDNS data, which accentuates this issue even more, as there are less chances of being able to collect enough 'malicious behaviour' of a domain before it gets sinkholed to be used in classification. As such, we no longer use sinkholed domains to build our blacklists.

[9] http://www.google.com/supported_domains
[10] http:en.wikipedia.org/wiki/Akamai_Technologies#Primary_domains
[11] http://www.alexa.com/topsites

In order to find which domains in our pDNS could be used for our training sets, we queried our database for these domains names . To obtain a bigger training set, each 2LD present either in whitelist or blacklist would return the full list of 3LDs that exist on the database associated with the given 2LD. Furthermore, we verify if both lists are completely disjoint, and if they are not, the domain names that overlap are removed. This allows us to obtain bigger training sets but it also increases the changes of introducing noisy data. From the resulting list, a smaller set is selected such that each entry in the whitelist/blacklist (either 2LD or 3LD) will result in at least one domain in the training set. Other than that, the selection process is random.

6.3 Terminology and Metrics

Conceptually, in our experiments we make use of three different sets: the training set, the validation set and the test set. The training set is meant to create the initial classifier and train it. The validation set is used to test different parameters for the classifier. Lastly, the test set is used to measure the classification accuracy. However, such approach requires splitting the data set into 3 different sets, which would reduce the amount of samples available for the training process. A different approach is cross validation:

Cross Validation: K-fold cross validation avoids the need of a validation set, by partitioning the training set into K folds. $K-1$ folds are used to train the classifier and the remaining one is used as validation set. This process is then repeated for each of the k folds and the result is the average values of each iteration [39]. A common value for k is 10 [29].

For each set of parameters that we would like to test, we use cross validation to train the classifier and to measure its performance. Afterwards, we can select the set of parameters that obtained the best performance in the cross validation and measure its performance on a test set. The reason why we do not use the performance measured in the cross validation is because we were able to 'brute force' the parameter values until obtaining the maximum accuracy. Such approach, cannot be applied on real-time analysis. As such, we choose the best parameter set expecting that it will yield the best results against the test set.

In order to measure the performance of our classifier we make use of different metrics, namely Hamming distance, Matthews Correlation Coefficient (MCC), Accuracy, Precision and Recall.

Metrics:

- **Hamming Distance:** We calculate the Hamming distance between two vectors containing binary entries. The Hamming distance is then the number of positions that have different values for the same index in both vectors. It is used to calculate the number of errors given the real classification of the samples and the predicted classification. For evaluation purposes we will use the concept of Hamming distance (HD), although we always present it in the form $1-HD$, which retrieves the number of correct classifications (also known as accuracy).
- **Matthews Correlation Coefficient (MCC):** The MCC give us a measure of quality of our classifier using the predicted values and the real values of each sample. It takes into consideration both false positives and false negatives, which makes it suitable for tests

even when the categories are not balanced with respect to the number of samples. The MCC ranges between -1 and 1, where 1 represents a perfect prediction, -1 implies that all predictions were wrong and 0 suggests that the classifier is a good as a random prediction. It can be calculated as follows:

$$MCC = \frac{tp * tn - fp * fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}$$

In which, tp/fp stands for true/false positives and tn/fn for true/false negatives.

- **Precision:** Precision corresponds to the ability of a classifier not to label a negative sample as positive. Thus, the higher the precision the smaller the number of false positives. Precision can be calculated as follows:

$$Precision = \frac{tp}{tp + fp}$$

- **Recall:** The Recall (or True Positive Rate) is the ability of the classifier to find all positive samples. Thus, the higher the recall the smaller the number of false negatives. Recall can be calculated as follows:

$$Recall = \frac{tp}{tp + fn}$$

In a binary classification problem, such as ours, the ratio of 'benign' (positive) samples that were correctly or incorrectly predicted are given by the values of tp and tn, respectively. Analogously, for the malicious (negative) samples and the values of fp and fn. As such, one might be interested in visualizing the relation between these values, which is given by the Receiver Operating Characteristic (ROC) curve.

ROC curve: The ROC curve is constructed using the values of True Positive Rate (Recall) and False Positive Rate (Fall-out). The latter corresponds to the ratio of negative samples that are labelled as positive. The Fall-out can be calculated as follows:

$$Fall - out = \frac{fp}{fp + tn}$$

In a perfect scenario, the value of True Positive Rate (TPR) is close to one (no false negatives) and the False Positive Rate (FPR) is close to 0 (no false positives).

We devised a total of 36 features, divided in two big groups. Each feature is likely to contribute differently for the classification process. For instance, in Decision Tree algorithms the feature used for each decision node is chosen using the Gini impurity or entropy (see Sec. 5.2.2). One other possible way to measure the feature performance is to calculate the ANOVA F-value:

ANOVA F-value: Intuitively, the ANOVA F-value is a one-way analysis of variance that allows to measure how well a certain variable (feature) 'explains' the differences between two groups of samples, in our case, benign and malicious samples. The ANOVA F-value is the ratio between the explained variance (variance of each group means) and the unexplained variance (mean of groups variance). Let us take as example a data set with 4 samples, 2 benign and 2 malicious. Let us say that each sample contains one feature value, and that for the malicious samples this features take the values $1, 2$ and for benign ones it takes the values $2, 3$. Applying the definition, one could calculate the F-value of this feature as follows:

$$F_{value}a = \frac{var(3/2, 5/2)}{mean(0.5, 0.5)} = \frac{0.5}{0.5} = 1$$

Now let us consider another feature for each sample, in which the malicious samples take the same values $1, 2$ but the benign samples take the values $3, 4$. Following the same procedure:

$$F_{value}b = \frac{var(3/2, 7/2)}{mean(0.5, 0.5)} = \frac{2}{0.5} = 4$$

Based on this results, one would pick the second feature ($F_{value}b$) as it has the highest F-value. Looking at the data set, this can be justified due to the fact that there is no overlapping of the feature values from samples in different categories. As such, this feature alone is able to characterize the whole data set.

7 Evaluation and Results

We evaluate the performance of the different algorithms using two distinct training sets, set A and set B. Each training set consists of 20,000 distinct domain names that are selected from the first week of our pDNS data, and labelled either as benign or malicious. The difference between these two sets is the ratio of benign versus malicious samples: the training set A consists of 10,000 benign and 10,000 malicious samples and set B consists of 18,000 benign samples and 2,000 malicious samples. The reason to use these two different sets is that, while the set A contains the same number of benign and malicious samples, the proportions used in set B are much closer to the proportions in real-life DNS traffic, in which the number of benign samples is much higher than the number of malicious ones. In addition, when appropriate, we use a third training set, set C, that consists of 15,000 benign samples and 5,000 malicious ones. This third set will mainly be used for comparison results and to support our analysis. The process used to select the domain names in the training sets is discussed in Sec. 6.2. Notice that this process is executed independently for each set, therefore the sets A, B and C are not disjoint sets.

The remainder of this section is organized as follows. We start by evaluating our feature set for the different training sets in order to investigate how each individual feature is able to characterize our data set. Afterwards, we perform a search in the parameters space of each algorithm to tune it particularly to our data set. Finally, we measure the performance of the classifier against a test set made out of the two week of pDNS data and analyse how good it performs for domain names that are not part of the training data.

7.1 Feature Importance

In order to measure which features perform better in our data set we calculated the ANOVA F-Value (see Sec. 6.3) for each of the 36 features. Since this test depends on the training set we perform this evaluation on both training sets A and B. The individual scores of each feature are plotted in Fig. 7 and 8, respectively.

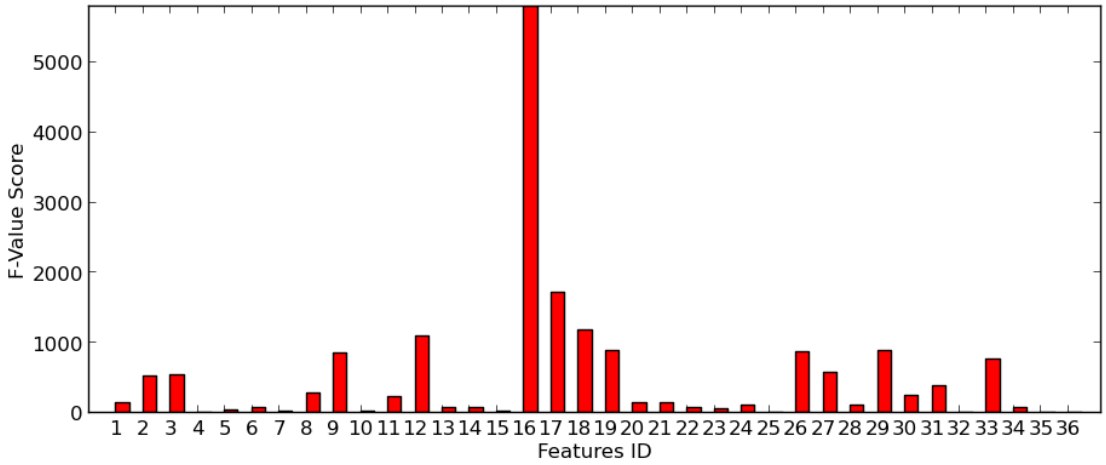

Figure 7: Feature Importance on Training Set A.

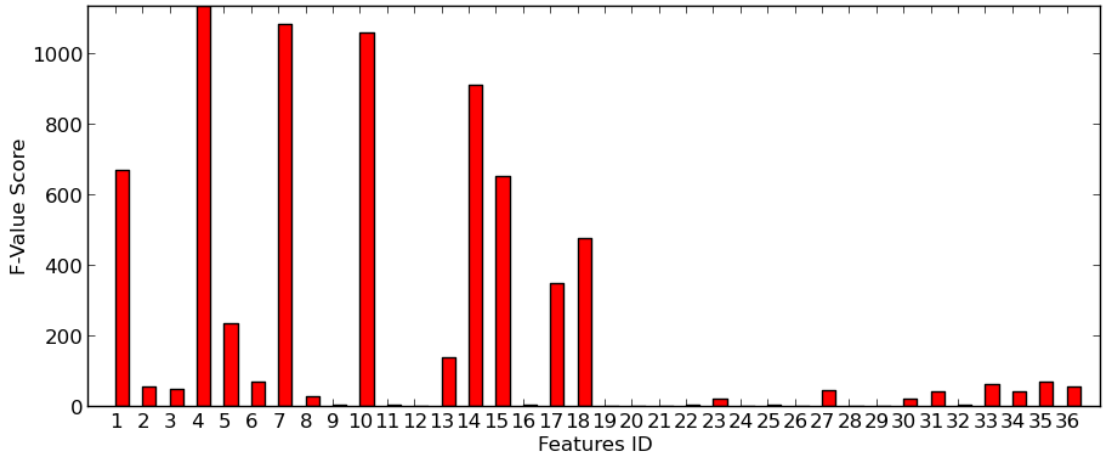

Figure 8: Feature Importance on Training Set B.

From these graphs we are able to observe that the features importance varies greatly with the selected data set. Moreover, not only the individual F-Values change but also relative importance of each feature against the others. For instance, while in the Fig. 7 the feature with the highest importance (F16) has an F-Value of more than 5,000, in Fig. 8 the highest F-Value (FT4) is significantly smaller, about 1,100. Furthermore, on training set B, F16 (digit ratio) has a F-score of 2.5. In addition, when using the training set B one can observe that the lexical features (F1-F18) tend to have higher importance than the network features. However, with the set A not only the overall features tend to have higher F-Values but also the focus is more distributed over both lexical and network features.

A possible explanation for such results is related with the number of malicious samples in our training sets. While the training set A contains 10,000 malicious domain names, the training set B contains only 2,000 malicious samples. Furthermore, we devised our features in order to detect both DGAs and FF techniques used by botmasters, however our blacklists contain domain names related to other malicious activity which is not necessarily related to DGAs or FF. This implies that using a data set with fewer malicious domain names might disregard domains that use one of these techniques. As such, there might not be enough samples in our training data in order

to raise the importance of network features. Furthermore, many network features are related to the values of first seen and last seen. As our training sets contain features extracted from only one week of pDNS data, it is possible that such period is not sufficient to make the network features relevant enough.

In order to corroborate our intuition, we performed the same experiment in training set C, which contains 25% of malicious samples, as opposed to training set B that contains only 10% of malicious samples. The results are illustrated on Fig. 9.

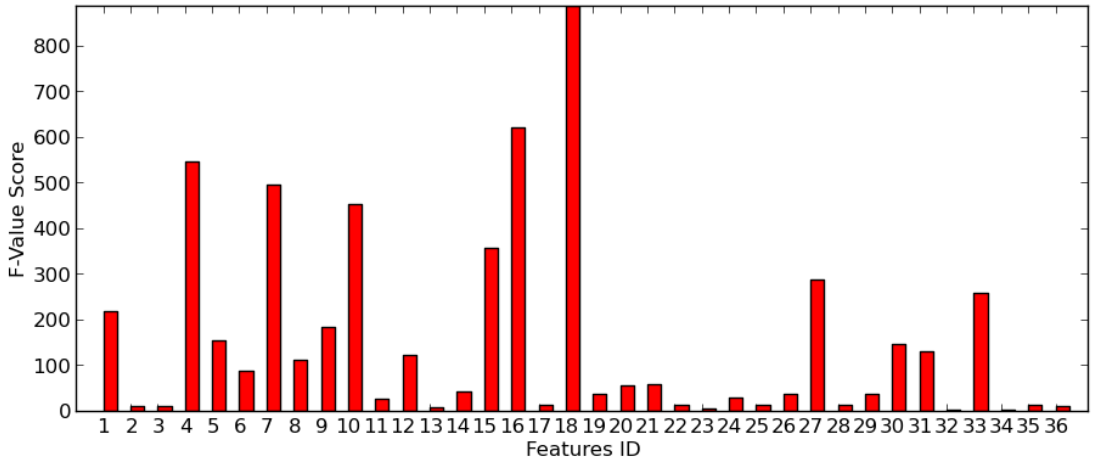

Figure 9: Feature Importance on Training Set C.

From Fig. 9 one can observe that, even though lexical features continue to perform better than network features, the latter have now a bigger impact. The same effect can also be observed when comparing the results of training sets B and C with the training set A.

When interpreting the mean values of features F26 and F29, we noticed that the average values of TTL-related features of the benign samples are higher than the values for malicious samples (see Tab. 7). Such results do not corroborate our initial hypothesis that malicious domains tend to have a lower value of TTL (see Sec. 6.1.2). After a closer look we noticed that many legitimate domain names like google.com, yahoo.com and mozilla.com have an average TTL value smaller than 300 seconds. Possible explanations for using small TTL values are related to the desired level of availability of a certain service and, for instance, minimize the damage in case of DNS redirection. It is also interesting to notice that the variance of such features for malicious samples is half of the variance for benign samples, meaning that malicious domains choose the TTL values from within a more restrict range of values [10].

Another interesting aspect is to analyse the variation of the importance of the same features, over the different training sets. Let us take as example, F16 which has approximated F-Values of 5,793 and 622 for the training sets A and B, respectively. The variation of the F-Values of this feature can be explained by the number of malicious samples that contains digits in the different training sets. For instance, in the training set A there are more than 7,000 malicious domains that contain digits in the domain name while only around 3,600 of the benign samples had digits on their domain names. On the other hand, in the training set B the ratio of malicious samples with digits in the domain and benign samples is 0.1, which is quite different from the ratio of 2 found in training set A.

7.2 Analysis Parameters and Features

From the results in previous section, we are now able to sort the features according to their importance, relative to a specific data set. We can then select different subsets of features and analyse how they perform as a whole against our test set. However, before we are able to evaluate the performance of our classifier we first need to examine the subset of features and the parameters configuration (see Sec. 5.2) that yields the highest classification results.

For each algorithm we tested different possible values for its parameters. The ranges of each parameters are specified in Tab. 3. In order to evaluate which set of parameters performs best, we performed for each algorithm a grid search (or exhaustive search) in which we test all possible combinations of the parameter values. Each algorithm is then evaluated using 10-fold cross validation (see Sec. 6.3) and parameter combination that achieved the highest score of Hamming distance is retrieved.

Parameter	Algorithm	Possible Values
Number of Neighbours	k-Nearest Neighbours	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50, 100, 200
Weights	k-Nearest Neighbours	uniform, distance-based
Maximum Depth	Decision Trees & Random Forests	3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35
Minimum Splits p. Sample	Decision Trees & Random Forests	2, 4, 5, 10
Number of Estimators	Random Forests	5, 10, 20, 50, 100

Table 3: The values tested for each parameter and respective algorithm.

In addition, we take advantage of our grid search to test which subset of features retrieves the best classification results. Although, performing an exhaustive search in the feature set would require too much computational power and time. Instead, we follow a greedy approach that starts by training the classifier with the best feature and performs the grid search on the parameters space. Afterwards, we add the second best feature and perform grid search once again. This process is then repeated until all features are taken into consideration. Afterwards, we select the parameter combination and feature set that achieved the highest scores during 10-fold cross validation.

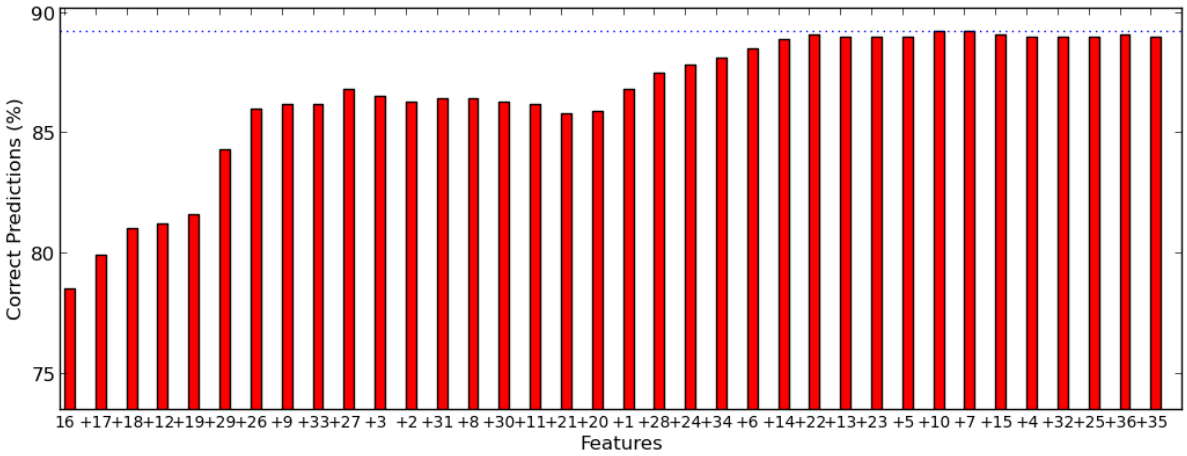

Figure 10: K-Nearest Neighbours [training set A] - highest score of 0.892 ($\pm$ 0.004), with 29 features.

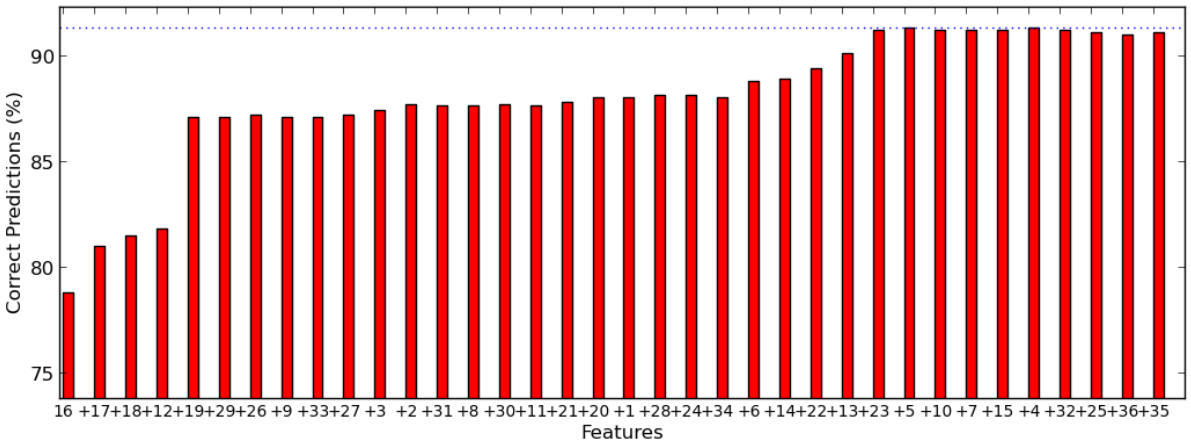

Figure 11: Decision Tree Results [training set A] - highest score of 0.913 ($\pm$ 0.003), with 28 features.

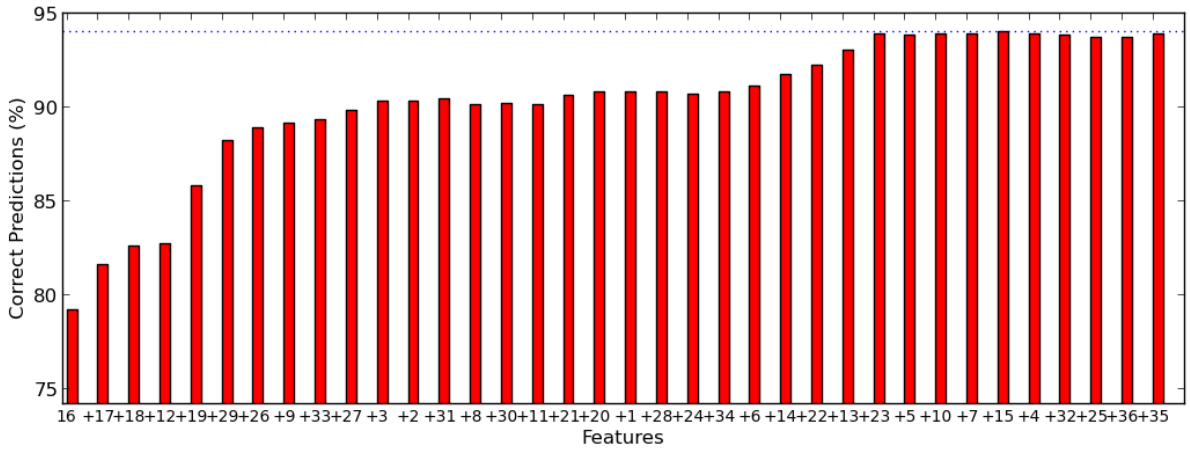

Figure 12: Random Forest Results [training set A] - highest score of 0.940 ($\pm$ 0.004), with 31 features.

Once more our analysis is two fold as will validate our classifier with both training sets A and B. The Fig. 10, 11 and 12 contain the results of using training set A with the different algorithms. Notice that each bar in the graphs represents a different feature set, which is the result of adding the feature indicated in the x axis to the feature set used in the previous iteration. The left most bar is an exception, as it only contains the indicated feature. The highest Hamming distance, 0.940 ($\pm$ 0.004), was accomplished by the Random Forest algorithm, with maximum depth $=$ 20, minimum samples p. split $=$ 4 and number of estimators $=$ 100.

	Precision	Recall
Benign	0.92	0.96
Malicious	0.96	0.91
Weighted Average	0.94	0.94

Table 4: Precision and recall values for the best parameters and features setting of Random Forests algorithm [training set A].

Table 4 summarizes the precision and recall values for each category using such parameters. From the recall one can conclude that there is a higher number of benign samples being correctly classified than malicious ones. On the other hand, the precision of malicious samples is higher than the benign ones. This implies that, the ratio of false negatives of benign samples is higher than the malicious samples. As we mainly focus on the detection of malicious domains, this is a favorable situation because we are more interested in reducing the number of malicious samples classified as benign.

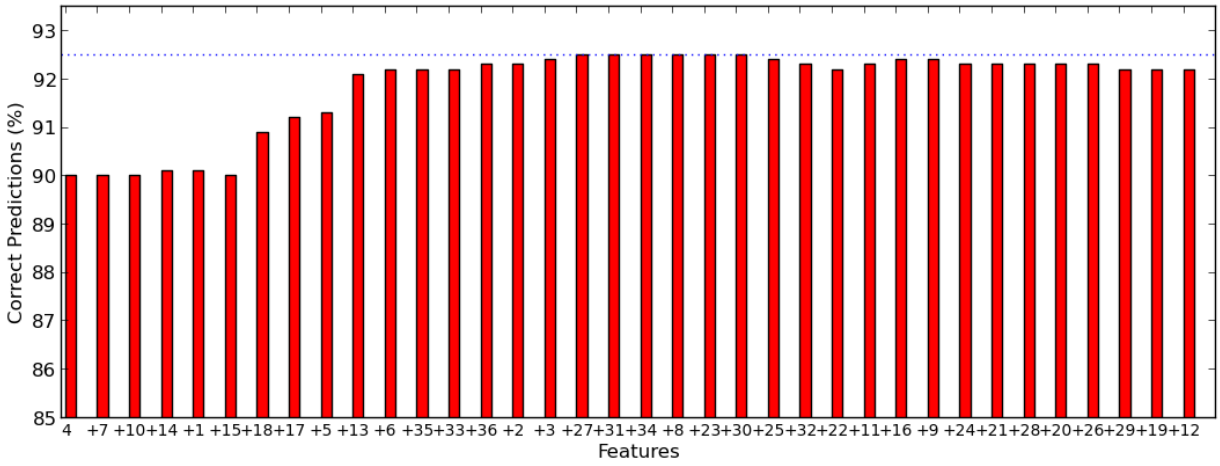

Figure 13: K-Nearest Neighbours [training set B] - highest score of 0.925 ($\pm$ 0.002), with 20 features.

The results of the exhaustive search using training set B can found in Fig. 13, 14 and 15. The best results, with a Hamming distance of 0.938 ($\pm$ 0.002), were also found with the Random Forest algorithm, with maximum depth $= 30$, minimum samples p. split $= 4$ and number of estimators $= 100$. The values of precision and recall are reported on Tab. 5.

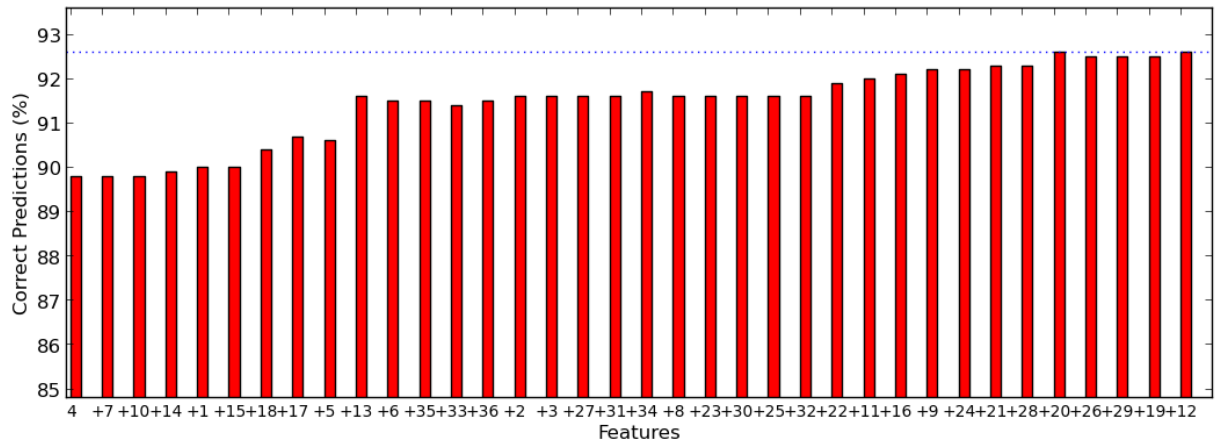

Figure 14: Decision Tree Results [training set B] - highest score of 0.926 ($\pm$ 0.002), with 32 features.

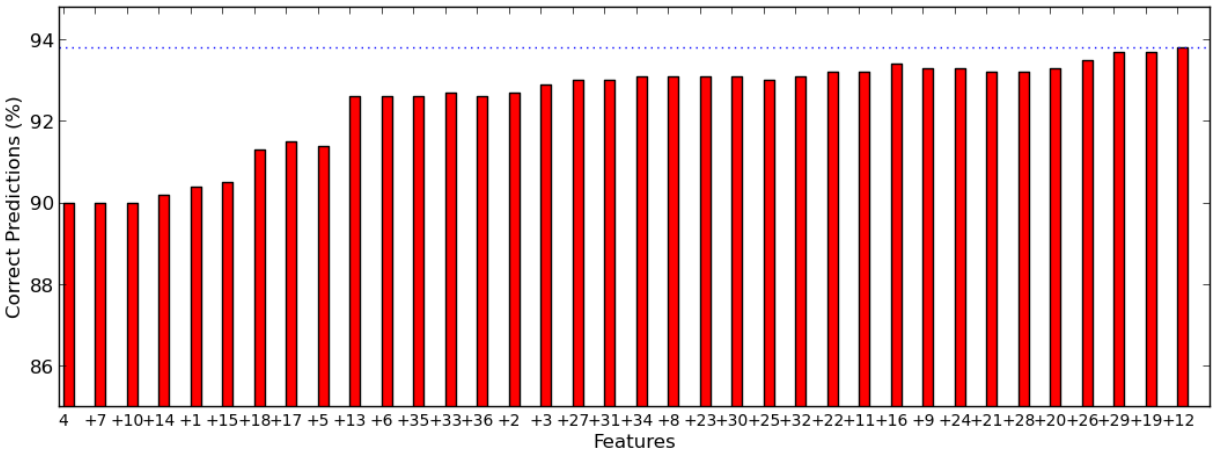

Figure 15: Random Forest Results [training set B] - highest score of 0.938 ($\pm$ 0.002), with all features.

	Precision	Recall
Benign	0.95	0.99
Malicious	0.79	0.50
Weighted Average	0.93	0.94

Table 5: Precision and recall values for the best parameters and features setting of Random Forests algorithm (training set B).

By using training set B we are able to achieve higher values of precision and recall for benign samples. Although, to the cost of lower values of such metrics for malicious samples. As afore mentioned, training set B as an unbalanced number of benign and malicious samples (90% - 10%). Due to the fact that we have much more benign samples than malicious one, our classifier

will be able to use all those benign samples and achieve better classification results for such class. On the other hand, as the number of malicious samples is reduced, also is the classifier's ability to properly classify those elements. Note that the weighted average values provided in Tab. 4 and 5, uses the number of samples from each class to calculate the weights. As such, it is not surprising that in Tab. 5 the average results are similar to Tab. 4 even though the classifier does not have similar performance for the classification of malicious samples.

7.3 Classifier Performance

In the previous section we performed experiments on two training sets and with different parameters values to measure the performance among the different settings. As result, we will now use the setting that achieved the highest performance and measure its classification results on a test set.

Our test set is calculated using both weeks of pDNS data. The idea is to recreate a real life situation, in which we are able to train our classifier on collected data and use it to classify data that will be gathered in the future. In this case, the collected data corresponds to the first week of pDNS, used in our training sets, and data to be collected corresponds to the second week. It is important to notice that when we generate the test set the feature values are calculated over the two weeks of pDNS data. In addition, the proportion between 'malicious' and 'benign' samples is not important when assembling a test set. As such, our test set contains 20,00 domain names, with 10,000 samples from each class. Notice that even though this setting allows us to measure the performance of our classifier, it does so in a rather different setting from real-life traffic, in which the proportion between benign and malicious samples is unbalanced.

From the results of each algorithm, we can now select the option that achieves the best Hamming distance for the best feature set. Consequently, we will use the Random Forests algorithm, with a $\text{maximum depth} = 20$, $\text{minimum samples p. split} = 4$ and $\text{number of estimators} = 100$, which when trained with training set A achieved a Hamming distance of 0.94.

When using this setting in our test set, our classifier achieved a Hamming distance of 0.97% and failed to correctly classify 651 out 20,000 samples. It may seem counter intuitive to achieve a better classification performance against the test set than with the experiments using the training and validation sets although, it can be explained by the longer observation period. The fact that two weeks of data are now used to calculate the feature's values allows the features to 'mature'. As mentioned in Sec. 6.1, one expects that domain names associated with botnet activity, and in general, with malicious activity, to remain active for shorter periods of time or to update their DNS information more often. When using longer observation periods we increase the chances of accentuating such behaviour for the domains associated with malicious activity.

The ROC curve (see Fig. 16) illustrates the relation between the recall and the fall-out ratio (see Sec. 6.3). In this case, a perfect scenario would be to achieve the highest value of recall for the lowest fall-out, resulting in an area under curve of value 1. Our evaluation shows that the area under the curve is of 0.996, which is close to the ideal scenario.

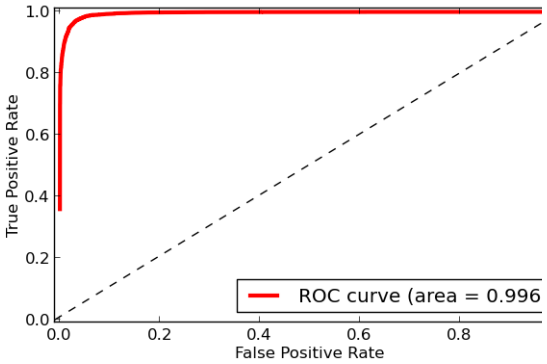

Figure 16: ROC Curve

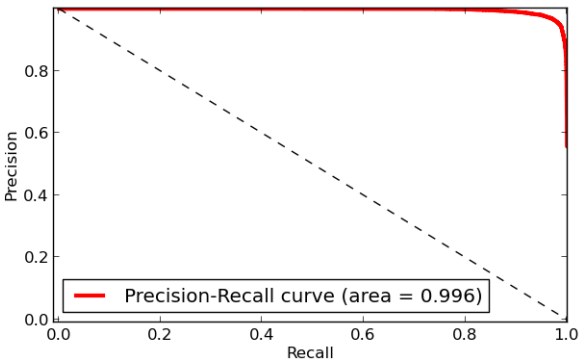

Figure 17: Precision-Recall Curve.

The overall precision and recall values can be found in Tab. 6 and the relation between these two metrics is illustrated in Fig. 17. When comparing such results with the ones presented in Sec. 7.2, we also observe higher precision and recall values than during our validation stage. The precision and recall formulas (see Sec. 6.3) differ only by one element, the number of false positives (fp) and the number of false negatives (fn), respectively. By plotting both measures one can observe how this two values, fp and fn, vary with each other. In an ideal scenario, the area under the precision-recall curve would reach the value 1, for which both the values of precision and recall would assume the maximum values. In Fig. 17 the area under the curve is 0.996, also close to the ideal scenario.

	Precision	Recall
Benign	0.96	0.97
Malicious	0.97	0.96
Weighted Average	0.97	0.97

Table 6: Precision and recall performance on the test set.

The confusion matrix Fig. 18 is a visualization method in which the columns correspond to the results of classification and the rows the real classification. Ideally, we want to maximize the values at the main diagonal, tp and tn while minimizing the values at the secondary diagonal, fp and fn. From the 651 samples misclassified, 268 were benign samples classified as malicious and 383 malicious samples labelled as benign.

At last, the MCC (see Sec. 6.3) provides a quality measure for our classifier ranging from -1 (poor classifier) to 1 (perfect classifier). In this setting, our classifier had a MCC score of 0.93.

7.3.1 New Domain Names

As mentioned in Sec. 5, a core property of a classifier is its ability to generalize from the data present in the training sets. Since we make use of both weeks of our pDNS data to build our test set, it is likely that domains present in the training set will also appear in the test set. To measure the generalization ability of our classifier, one could create a subset of the test set that contains domains that appear only in the second week of data. Measuring the performance of our classifier

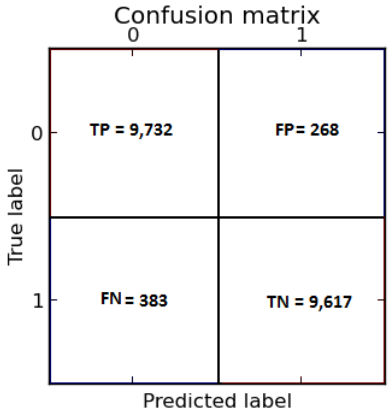

Figure 18: Confusion Matrix - the benign category is represented by 0 and the malicious category by 1.

in this new test set, allows us to measure the classifier performance for domains never seen in the training data (training set A).

Consequently, we verified which blacklisted domains show up on the second week of our pDNS data and do not show up on the first one. We look particularly to the blacklisted domains in order to assert the classification performance of malicious domain names, the ones we are more interested on. We found a total of 283 new domain names in such conditions. Afterwards, we submitted these domains names to our classifier and measure the performance of its classification. Our Random Forest classifier was able to correctly classify 81% of the samples, resulting in approximately 50 misclassified samples.

As one could expect, classifying domain names that do not make part of the training set is more difficult than classifying domain names from which we already have some knowledge about. When analysing the misclassified samples we noticed that many of the blacklisted domain names do not exhibit any malicious behaviour, from our feature perspective, for instance several IP address given as answers, small values of TTL, etc. Most of the domains where queried just once during the observation period. A possible explanation is related to the fact that the test set was created using domain names that only appear in the second week of data. Therefore, and as mentioned in Sec. 7.3, this might not be enough time for the network features to 'mature'. In addition, our blacklists do not contain only botnets-related domains (likely to exhibit such characteristics) but also domains associated with phishing or spam that might exhibit different characteristics.

Furthermore, while one would like the lexical features F15 to F18 and all network features to hold different values for benign and malicious samples, the same cannot be expected from F1 to F14. The latter set of features is meant to grasp the character distribution and recognize similar domains (see Sec. 6.1.1). As such, if a certain domain is not present in the training data one cannot expect these features to be of use in the classification of such domain name. Instead, if another domain name with similar feature values is present, the classifier might be wrongly influenced to output the label of such domain.

7.3.2 Botnet-related Domain Names

Our classifier reported an accuracy of 81% for domain names seen for the first time during test phase. Not only these results are influenced by the observation period (see Sec. 7.3.1) but also by the way we build our data sets. For instance, while our feature set is devised to classify domain names associated with botnet activity and the use of FF, DGA techniques, our blacklists, hence training, validation and test sets, contain domain names associated with malicious activity in general, including phishing, spam, Trojans, etc.

To measure the performance of our classifier for domain names associated with botnet activity, DGA or FF we created a second subset of our test set. This new test set contains only blacklisted domain names that are known to be associated with botnets and/or the use FF techniques. We used the same blacklists as mentioned in Sec. 6.2 to build the test set. In addition, evidences of the Asprox botnet [13] using double FF were found in the two weeks of pDNS data. The resulting blacklist contained 1,304 domain names, from which 58 are present in our pDNS database and can be used for the classification purposes.

Training our Random Forest classifier with training set A and test it against the specified test set yields a classification accuracy of 0.90, with only 7 samples being misclassified. Unfortunately, with such a small sample it is impossible to extract statistical results or conclusions that hold meaning. Nevertheless, looking at the feature values we concluded that the average TTL of the domains names in this test set was inferior to the one reported in Tab. 7. In fact, domains names associated with FF and botnet activity have an average TTL of 10,000 seconds, while malicious domains in general have an average value 13,000 seconds. This implies that even though some of our assumptions no longer hold (see Sec. 6.1.2 and 7.3.1) on our general purpose blacklists, when we zoom-in the botnet and FF related domains these assumptions proof to be true.

Sets	Training set A				Botnet-related set	
Feature	Benign		Malicious		Malicious	
	Mean	Variance	Mean	Variance	Mean	Variance
22	2.3094	65.17287	1.3773	61.79834	23.67241	5429.25476
23	4.4521	260.01631	2.2563	639.98321	113.72414	87007.9239
24	1.3635	8.40097	1.0647	0.60471	1.81034	11.15369
25	9.24697	209.8896	8.86555	141.53657	20.76085	851.44256
26	13467.6402	4816164355.85954	38039.3404	2158807933.57893	10572.93103	638799800.02973
27	81.38652	5126.201	58.20099	4134.83258	106.26437	11451.03741
28	6.9106	120.2256	8.52796	130.29294	5.96605	110.00707
29	13316.84678	4806750011.15273	38024.2976	2158674892.5297	10055.2126	602928089.99317
30	71.47114	4574.2763	56.78111	4023.11721	69.13747	9963.6868
31	21.55929	9257.3683	2.09335	664.86646	25.73328	4512.31023
32	1210944.58224	852826191508204	300253.24149	303488199165753	15516746.93362	13723816794780400
33	419.59886	1483460.28242	58.64818	218415.05308	717.76972	8236982.7506
34	1.455	1.02578	1.1966	7.58055	16.91379	644.49257
35	1.5152	1.73297	1.5376	195.72499	63.82759	18287.90131
36	1.9349	10.81826	1.8012	473.39948	106.48276	74892.56005

Table 7: Mean and variance of aggregating network features per category [training set A and botnet-related set].

8 Related Work

Botnet detection has drawn much attention over the scientific community in the past few years. Several different approaches have been used in the detection of botnet activity. In this section we

present a summary of different detection techniques that resort pDNS data to identify domain names associated with botnet activity or, more generally, with malicious activity.

8.1 Notos

Notos [2] is a domain name reputation system that makes use of network, zone and evidence-based features. While network features are meant to describe how a domain has its resources (domains name and IP addresses) allocated, zone-based features are characteristics extracted from the list of IP addresses associated to a domain name and from the history of the domain name itself. Lastly, evidence-based features take into consideration the number of known malware samples that contact a given domain name or IP address, and the number of blacklisted IP addresses resolving to a given domain name. The extracted features include, but are not limited to, the number of distinct BGP prefixes related to the list of IP addresses associated to a given domain, the number of different countries, the number of different AS, the number of domains associated to a single IP address, the average domain name length and different registration dates for IP addresses associated with 2LD and 3LD of a given domain name.

Notos uses a clustering technique (X-means [33]) that during the training stage learns how to identify different network behaviours. The authors devised five different classes of domains, namely popular domains, common domains, Akamai domains, CDN domains and dynamic DNS domains. The resulting system has a TPR of 96.8% and 0.38% of FPR.

8.2 EXPOSURE

Bilge et al developed EXPOSURE [10], a system that makes use of pDNS data to detect domains involved with malicious activity. Their system is has a total of 15 features distributed over the 4 categories: time-based, DNS answer- based, TTL value-based and domain name-based.

Time-based features split the time into intervals and perform two kinds of analysis. The first one, global analysis (using all intervals), is used to measure whether a domain is a short-lived domain (active for a small amount of time) or if it exhibits different behaviours for small intervals of time. On the other hand, local analysis (using a specific interval) is used to measure the domain's behaviour over time. Extracted features also include daily similarity, access ratio and repeating patterns. DNS answer-based features measures the heterogeneity in the list of IP addresses associated to a domain name. Extracted features include the number of different IP addresses, number of different counties, number of distinct domains that share the same IP address and reverse DNS results. As malicious systems tend to have lower values of TTL to improve availability and frustrate take down efforts, a group of features is devised to identify this behaviour. Such features include average and standard deviation of the TTL value, number of different TTL values, number of changes in the TTL values and usage of specific TTL ranges. Domain name-based features measure the number of digits in the domain name and the length of the longest meaningful string. The rational behind such feature is that benign and legitimate system typically choose easy-to-remember domain names.

Using this feature set and a J48 Decision Tree classifier, EXPOSURE is able to achieve a detection ratio higher than 98% with only 1% of false positives.

8.3 Kopis

As opposed to Notos and EXPOSURE, Kopis [3] collects DNS data from a higher point in the hierarchy (TLD and ANS) than the local recursive DNS servers. Ignoring recursive queries and moving up in the DNS hierarchy provides less granularity of the data due to the effects of DNS caching. For instance, the `count` value used in our feature set would no longer be available (see Sec. 6.1). On the other hand, monitoring certain ANS and at TLD servers allows us to collect all DNS requests made to domain names that use one of those servers as ANS or delegation point.

Kopis builds three different feature sets: requester diversity, requester profile and resolved IP reputation. The first set of features aims to characterize the diversity in the poll of IP addresses that requests a domain name and uses of the BGP prefixes, country codes, AS and others, associated to a given IP address. The requester profile is designed to distinguish among different sources of DNS requests. The idea is to give higher weights for source IP addresses that make requests on behalf of several clients, such as an Internet Service Provider (ISP), and lower weights for a single end-user machine. The last set of features measures the reputation of the resolved IP addresses using whitelists, blacklists and the number of known malware-related domains that point to the AS and BGP prefixes of those IP addresses.

Kopis divides the time into epochs and at the end of each epoch calculates the feature vectors for the streams of DNS traffic. Afterwards, each observed domain and respective feature set is feed to a classifier which will return a label and a confidence score. The resulting system achieves a detection ratio of 98.4% with false positives of around 0.5%.

8.4 Pleiades

Antonakakis et al developed Pleiades [4], a system that clusters and models DGA-generated domain names using NXDomain (Non-Existent Domain) responses. To create the clusters the authors use two measures: lexical similarity of the requested domain name (see Tab. 8) and the list of hosts that request such domain. As result, the clusters are meant to cluster domains that are likely generated by the same DGA.

Pleiades also uses the the X-means clustering algorithm to generate the clusters to model different DGA algorithms. In addition, Alternating Decision Trees are used to determine whether a given NXDomain response is likely associated with specific known DGA and label it. Afterwards, for each different DGA A, we create a Hidden Markov Model [35] and feed all NXDomain responses associated with A. For each successfully resolved domain name the model will output the likelihood of being generated by such a DGA.

Hidden Markov Models are used to determine the likelihood of such domain being generated by a specific DGA. The authors modeled six different DGAs from their training data and performed performance measures. For FPR up to 3%, Pleiades achieves a true positive rate higher than 91%, for five out of the six modeled DGAs.

ID	Designation
P1	Mean, median and standard deviation of 1-gram
P2	Mean, median and standard deviation of 2-gram
P3	Mean, median and standard deviation of 3-gram
P4	Mean, median and standard deviation of 4-gram
P5	Mean and standard deviation of entropy(d)
P6	Mean and standard deviation of entropy(2LD)
P7	Mean and standard deviation of entropy(3LD)
P8	Mean, median, standard deviation and variance of domain name length
P9	Mean, median, standard deviation and variance of # of domain levels
P10	Number of distinct characters
P11	Number of distinct TLD
P12	Ratio of .com TLD
P13	Ratio of other TLD
P14	Mean, median and standard deviation of the occurrence frequency distribution for the different TLDs

Table 8: Pleiades features to measure lexical similarity among NXDomain responses.

9 Conclusion

In this thesis we propose to use machine learning techniques in order to classify domain names as domains associated with malicious activity or benign activity. In particular, we want to identify domain names associated with botnet activity that make use us FF networks or DGAs. As such, we make use of pDNS data that provides an overview over the domain DNS history. From the collected pDNS data we devised a total of 36 features which can be divided in two big groups: lexical features and network features. While the former analyse the domain name string, the latter group of features attempts to characterize network information (DNS data) of the different domains.

For our experiments, we built three different classifiers, namely the kNN, Decision Trees and Random Forests. Afterwards, we created different training sets, using one week of pDNS data and selected the parameters for the different algorithms. We trained our classifiers in a supervised learning set, hence each sample in the training set has to be labelled as malicious or as benign. Thus, we make use of blacklists and whitelists to select which domains to include in our training sets. Since the Random Forests algorithm achieved the best results at this stage, we used it for our remaining experiments.

We built a test set out of two weeks of pDNS data to test our classifier. From our experiments we can conclude that we can successfully classify domain names as malicious or benign using pDNS data, having a false positive and false negate rate of 3%. In addition, such results are achieved with a privacy-preserving approach, where the IP addresses of the requesters are removed from the data used for classification. We further evaluated our classifier for domain names that do not make part of the training set and for domain names known to be associated with botnet activity, FF networks or DGAs. Although, the resulting test sets were too small, with 283 and 58 domain names respectively, to be able to draw any statistical significance out of the results.

9.1 Future Work

There are several different avenues of research that may be used to further improve our classifier. Firstly, it would be interesting to perform the same experiments on a larger data set, containing more than two weeks of pDNS data. This would not only allow our features to 'mature' (see Sec. 7.3) but also to build larger training sets while being more conservative in the white and black list construction, hence reducing the chances of including noisy samples.

Not only the quality of the training tests influences the classifier performance but also the feature set. As such, it may be of value to consider new or different features. For instance, WHOIS data contains information associated with an IP address, such as the Autonomous System (AS) number, Border Gateway Protocol (BGP) prefix, country code, registrar information and allocation date. The country code value could be used to measure how many different country codes are associated to the IPs of a single domain name. In addition, information stored in the SOA RR of domain name might also be used for classification purposes. Fox-IT anonymizes the pDNS data before it is added to the database. While enhancing privacy, it also removes the chance of using the origin IP address of the DNS queries in our analysis. These IP addresses could be used to analyse group activities [15, 22] or to generate list of IPs likely connected to the same botnet or infected with the same kind of malware. In addition, our passive DNS is aggregated in the sense that we store maximum seen value of TTL, last seen and count for each domain, and the minimum seen value of first seen. In order to improve our classification performance it could be interesting to aggregate data differently, for example by day, so that we have a higher granularity over the data. In addition, our feature set is composed by features calculated by continuous functions. Using continuous domains for the feature values might turn the task of defining the boundaries between classes more difficult. On the other hand, the use of discrete values (or list of ranges) for our features might easy this process.

In this analysis we left out the NX (Non-Existent) Records, generated in high volume by DGAs, and NS records. The analysis of these records could further improve the detection of DGA and in allow the identification of double FF networks. In addition, the Asprox botnet present in our pDNS data (see Sec. 7.3.2) uses double FF, changing the IP records of both domain names and NS associated with the CnC servers. Although, from the point of view of our database we are only able to detect the changes in the IP address, the A records.

Lastly, our Random Forests classifier averages the classification of each individual tree to determine the label of a domain name. Other possible approach consists in a voting process that contains the vote of each individual decision tree [39]. What is interesting about this approach is that we can then specify the minimum number of votes required to consider a domain name as malicious. For instance, if we require less malicious votes to label a domain name as malicious we might be able to reduce the number of false positives. Although, there is also the chance that we increase the number of false negatives.

9.2 Discussion

Ideally, we would like to have such a system on a real-traffic network and on a real-time scenario to measure the classification performance in this scenario and also the temporal complexity. However, with such a high rate of false positives and false negatives, when considering the number of DNS queries per second on a network of reasonable size, there would be a big number of misclassified samples. In addition, while the reported results allow us to measure the classifica-

tion accuracy of our classifier, the setting in which the experiment was conducted slightly deviates from a real-traffic scenario. Such is due to the fact that real-traffic typically contains more legitimate domain names than domain names associated with malicious activity. Consequently, since we train our classifier to minimize the number of false negatives (in deterioration of false positives) and test it on a balanced test set (10,000 benign samples and 10.000 malicious samples), the classification performance on real-traffic scenario is likely inferior to 97%.

In order to improve our classifier we need to improve our training set by reducing the amount of noise and introduce new features that may improve the classification accuracy and precision. However, it is also important to consider the database performance. In our setting, we used a PostgreSQL database that contains more than 41 million entries containing two weeks of pDNS data. These entries correspond to the number of distinct pairs $<$domain name, IP address$>$ collected during the observation period. If we were to consider a different level of data aggregation, for instance daily, database performance might become a big bottleneck. Currently, performance is not an issue for the classification purposes. Although, with so many entries, we are not able update our database efficiently. To solve this problem, instead of updating the first week of pDNS in our database, we created a second table in which we added the two weeks of pDNS aggregated beforehand. Therefore, in order to have a bigger observation period and a different granularity over the pDNS data, we will have to modify our database taking performance into consideration.

Compared to previous work such as Notos [2], EXPOSURE [10] and Kopis [3], that extract their feature set from different data sources, we build the feature set solely from our pDNS data. One of the requirements for our proof-of-concept was that it should be able analyse a DNS feed in real time. As such, we were not able to include other data sources, such as AS data and SOA records, because the services that make such data available typically do not offer time guarantees. Due to that, even though we have similar accuracy levels as such systems, our classifier has higher FPRs.

Our classifier can be seen as a reputation system for domain names, that uses pDNS data to analyse the domain names behaviour. Domain names that follow the same kind of behaviour, with respect to pDNS data and the extracted features, will have the same classification. In particular, our classifier can be used to see if a domain name has a behaviour similar to those known to be related to malware. This information might be useful to identify new domains related to malicious activity which are not yet present in the blacklists. In addition, an ISP could use such a classifier to build a system that would allow to measure the amount of DNS queries that might be related to malware or botnet activity in their network. To operate in such a big scale, we would have to further improve the classifier precision and accuracy to reduce the number of false positives and false negatives.

References

[1] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 41--52, New York, NY, USA, 2006. ACM.

[2] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a Dynamic Reputation System for DNS. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, Berkeley, CA, USA, 2010. USENIX Association.

[3] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 27--27, Berkeley, CA, USA, 2011. USENIX Association.

[4] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, Berkeley, CA, USA, 2012. USENIX Association.

[5] Javed A. Aslam, Raluca A. Popa, and Ronald L. Rivest. On Estimating the Size and Confidence of a Statistical Audit. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 8--8, Berkeley, CA, USA, 2007. USENIX Association.

[6] Paul Baecher, Markus Koetter, Maximillian Dornseif, and Felix Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID*, pages 165--184. Springer, 2006.

[7] M. Bailey, E. Cooke, F. Jahanian, Yunjing Xu, and M. Karir. A Survey of Botnet Technology and Defenses. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications & Technology*, pages 299--304, 2009.

[8] Paul Barford, Yan Chen, Anup Goyal, Zhichun Li, Vern Paxson, and Vinod Yegneswaran. Employing Honeynets for Network Situational Awareness. In Sushil Jajodia, Peng Liu, Vipin Swarup, and Cliff Wang, editors, *Cyber Situational Awareness*, volume 46 of *Advances in Information Security*, pages 71--102. Springer US, 2010.

[9] Paul Barford and Vinod Yegneswaran. An Inside Look at Botnets. In Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171--191. Springer US, 2007.

[10] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE : Finding Malicious Domains Using Passive DNS Analysis. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*, San Diego, ÉTATS-UNIS, 02 2011.

[11] James R. Binkley and Suresh Singh. An Algorithm for Anomaly-based Botnet Detection. In *Proceedings of the 2Nd Conference on Steps to Reducing Unwanted Traffic on the Internet - Volume 2*, SRUTI'06, pages 7--7, Berkeley, CA, USA, 2006. USENIX Association.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[13] R. Borgaonkar. An Analysis of the Asprox Botnet. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pages 148--153. IEEE, July 2010.

[14] A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, and G. Eaton. Real-Time Detection of Fast Flux Service Networks. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications & Technology*, pages 285--292, 2009.

[15] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet Detection by Monitoring Group Activities in DNS traffic. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 715--720, Oct 2007.

[16] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting botnets. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, SRUTI'05, pages 6--6, Berkeley, CA, USA, 2005. USENIX Association.

[17] David Dagon. Botnet Detection and Response. In *OARC workshop*, volume 2005, 2005.

[18] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A Survey of Botnet and Botnet Detection. In *Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies*, SECURWARE '09, pages 268--273. IEEE Computer Society, 2009.

[19] Edward Fjellskål. Passive DNS - gamelinux, 2011. https://github.com/gamelinux/passivedns, Accessed: 2014-05-22.

[20] Tilman Frosch. *Mining DNS-related Data for Suspicious Features*. PhD thesis, Ruhr-Universität Bochum, 2011.

[21] 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'07, pages 8--8, Berkeley, CA, USA, 2007. USENIX Association.

[22] G Gu, J Zhang, and W Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium, NDSS*, 2008.

[23] N. Hachem, Y. Ben Mustapha, G.G. Granadillo, and H. Debar. Botnets: Lifecycle and Taxonomy. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1--8, May 2011.

[24] Yuanchen He, Zhenyu Zhong, Sven Krasser, and Yuchun Tang. Mining DNS for Malicious Domain Registrations. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pages 1--6. IEEE, Oct 2010.

[25] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and Detecting Fast-Flux Service Networks. In *NDSS*, 2008.

[26] 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'08, pages 9:1--9:9, Berkeley, CA, USA, 2008. USENIX Association.

[27] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale Botnet Detection and Characterization. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 7--7, Berkeley, CA, USA, 2007. USENIX Association.

[28] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1245--1254, New York, NY, USA, 2009. ACM.

[29] Geoffrey McLachlan, Kim-Anh Do, and Christophe Ambroise. *Analyzing microarray gene expression data*, volume 422. John Wiley & Sons, 2005.

[30] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[31] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, David Dagon, and Wenke Lee. Beheading Hydras: Performing Effective Botnet Takedowns. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 121--132, New York, NY, USA, 2013. ACM.

[32] Gunter Ollmann. Botnet Communication Topologies. *White Paper, Damballa*, May 2009.

[33] Dan Pelleg, Andrew W Moore, et al. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, pages 727--734, 2000.

[34] Honeynet Project. Know Your Enemy: Tracking Botnets, 2006. http://www.honeynet.org/papers/bots/, Accessed: 2014-06-11q.

[35] L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257--286, Feb 1989.

[36] C. Schiller and J.R. Binkley. *Botnets: The Killer Web Applications*. Elsevier Science, 2011.

[37] Antoine Schonewille and Dirk-Jan van Helmond. The Domain Name Service as an IDS. *Research Project for the Master System-and Network Engineering at the University of Amsterdam*, 2006.

[38] Scikit-learn, Machine Learning Library for Python, 2010. http://scikit-learn.org, Accessed: 2014-06-08.

[39] Scikit-learn, User Guide, 2010. http://scikit-learn.org/satble/user_guide.html, Accessed: 2014-06-08.

[40] W.Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet Detection Based on Network Behavior. In Wenke Lee, Cliff Wang, and David Dagon, editors, *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 1--24. Springer US, 2008.

[41] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.

[42] Amit Kumar Tyagi and G Aghila. A Wide Scale Survey on Botnet. *International Journal of Computer Applications*, 34, 2011.

[43] Ricardo Villamarín-Salomón and José Carlos Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 476--481. IEEE, 2008.

[44] P. Vixie, S. Thomson, and Y. Rekhter. Dynamic Updates in the Domain Name System (DNS update)", RFC 2136, 1997.

[45] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. Honeypot Detection in Advanced Botnet Attacks. *Int. J. Information and Computer Security*, 4(1), 2010.

[46] Florian Weimer. Passive DNS Replication. In *FIRST Conference on Computer Security Incident*, 2005.

[47] Xiaonan Zang, Athichart Tangpong, George Kesidis, and David J Miller. Botnet Detection Through Fine Flow Classification. *Unpublished, Report No. CSE11-001*, 2011.

[48] Zhaosheng Zhu, Guohan Lu, Yan Chen, Zhi Fu, P. Roberts, and Keesook Han. Botnet Research Survey. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 967--972, July 2008.