Popularity-based Detection of Domain Generation Algorithms

Or: How to detect botnets?

by

J. Abbink

to obtain the degree of Master of Science
at the Delft University of Technology.
to be defended publicly on Tuesday September 19, 2017 at 01:00 PM.

Student number:	4002237	
Thesis committee:	Dr. C. Doerr,	TU Delft, daily supervisor
	Associate Prof. Dr. ir. J.C.A. van der Lubbe,	TU Delft
	Dr. C.C.S. Liem,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Parts of this thesis have been published in the proceedings of the 12th International Conference on Availability, Reliability and Security [1].

Abstract

In order to stay undetected and keep their operations alive, cyber criminals are continuously evolving their methods to stay ahead of current best defense practices. Over the past decade, botnets have developed from using statically hardcoded IP addresses and domain names to randomly-generated ones, so-called domain generation algorithms (DGA). Malicious software coordinated via DGAs leaves however a distinctive signature in network traces of high entropy domain names, and a variety of algorithms have been introduced to detect certain aspects about currently used DGAs.

Today's detection mechanisms are evaluated for botnets that make the next obvious evolutionary step, and replace domain names generated from random letters with randomly selected, but actual dictionary words. It can be seen that the performance of state-of-the-art solutions that rely on linguistic feature detection would significantly decline after this transition, and an alternative novel approach to detect DGAs without making any assumptions on the internal structure and generating patterns of these algorithms is proposed.

Preface

After six years as a Computer Science student, this thesis marks the end of my time at Delft University of Technology. I want to use this opportunity to thank several people that assisted me along the way.

First, I would like to thank Christian Doerr, my daily supervisor for this thesis project, whose guidance and feedback helped me during the entirety of this project. I also want to thank Jan van der Lubbe and Cynthia Liem, the other two members of my thesis committee, for their time and effort they spent on my graduation. I also thank Suzy Sirks-Bong for proofreading early drafts of this document and providing me with helpful feedback and motivation during the time I worked on this.

Besides the people directly related to this thesis, I would like to thank all the friends I have met during my time in Delft for making my years at this university truly enjoyable. Finally, I would like to thank my family for their continued support over the years and helping me out where and whenever they could.

J. Abbink
Delft, September 2017

Contents

1

Introduction

Cyber crime is an ever growing problem in recent years. It has been estimated that in 2013 alone cyber crime cost United States companies and private parties $100 billion [13] and estimations for 2019 predict that this will rise to over $2 trillion [36] world wide. These numbers include everything from targeted attacks on companies and governments to large scale broad attacks on personal computing devices, including average desktop computers and laptops, but also routers and Internet-connected "smart" devices (Internet of Things (IoT); e.g. fridges, watches, washing machines). Nowadays, these personal desk- and laptop computers are often targeted to be infected with software that includes the host in a botnet, a network of computers, also called bots, executing tasks as controlled by a central authority. Examples of botnets running on lower level devices are known as well [21] and will probably be used more often in the future [23]. Well known examples of traditional personal computer botnets include Conficker [29], ZeuS [24] and Torpig [39]. At their peak, these botnets had tens of millions of individual computers infected and were mainly responsible for sending spam messages.

1.1. Evolution in Botnet Control

To receive their task, botnets connect to a command and control server (C&C). A basic problem in the deployment of botnets that malware authors (botmasters) face is that they need to find a way so that the distributed clients can find and connect to the C&C server, and although the address is known to anyone in possession of the malware the channel should not be easily disrupted by network owners or law enforcement. This is for example the case with the most simple way of linking clients to the C&C server by means of a static and hard-coded IP address, which could trivially be blacklisted in firewalls or taken out of circulation by a hosting provider on request from the authorities.

In order to be able to dynamically update the location of the C&C infrastructure, botmasters came up with multiple approaches to dynamically locate the command and control server. One observed approach [9] uses publicly available websites to distribute information about commands to be executed, or ways to find the actual C&C infrastructure. As it is to be assumed, that these public websites will not be taken down by authorities, botmasters only have to rely on a lack of moderation by those sites in order to stay active for long periods. As soon as the webmaster of a used website notices these malicious access patterns, these botnets can be efficiently sink holed. Another common approach to dynamically control infected hosts in a botnet is to create a peer-to-peer (P2P) network with no clear central authority. While these kind of networks are less effective for a botmaster, than a setup with a clear reachable central authority, as using a P2P network may result in significant delays for finding other infected peers and distributing commands to all nodes in the network, these type of botnets can often be taken down by inserting a majority of simulated peers, also called a "Sybil attack" [10], which will simply not relay commands [26, 39]. Unlike taking down a central botnet authority, this does however require substantial amounts of work by researchers, instead of relying on network operators or law enforcement to blacklist a single address on the internet. In order to prevent being susceptible to take downs by such a Sybil attack, some botnets have moved to use the Bitcoin blockchain [25] as command and control data storage [31]. Due to the size of computing power of this network, it is infeasible to apply a Sybil attack on this network. A minor drawback of this approach for botmasters is, that any com-

mand sent to nodes costs money, and researchers can easily monitor all commands issued without running the malware itself, as the data of the blockchain is publicly available.

A final method, which this thesis will discuss in depth, are botnets finding the control server based on domain names. Although a domain can also be seized by law enforcement, the process takes significantly longer, as it may require cross-border actions. To further limit this angle, botnet owners are also not relying on a single static domain name in their client software, which after being seized would leave the entire system inoperable, but rather dynamically generate domain names on the fly.

The domains at which the C&C server is contacted are computed using a domain-generation algorithm (DGA) based on the current time or some other information publicly available across all hosts, each only valid for a short amount of time. In addition to the short validity, DGAs frequently generate also hundreds of candidate domains per time interval. Infected hosts look up all candidate values to find the domain that was actually registered by the botnet owner, which from a defense perspective makes this mechanism very difficult and costly to suppress, as registering, seizing or sink holing thousands of domain names per day is often too administratively complex, costly or not scalable.

Current DGAs generate domain names by concatenating random letters and morphemes. This has the clear advantage that the resulting domain names are with very high likelihood available, but has the drawback that they leave in the network a characteristic trace of a series of unsuccessful look ups (NXDomains) for domain names, each with a high entropy name such as yfewtvnpdk.info or rwyoehbkhdhb.info that is unlikely requested by normal users. A variation to this from literature [11], applies this same technique of generating domain names, to generating user names on websites that allow publishing arbitrary data (e.g. twitter.com and github.com). In practice, malware employing this technique has not been observed, yet.

1.2. DGA detection in literature

In this thesis, the focus will be on detecting parts in a network, seemingly infected by software contacting generated domains frequently, hence finding active botnets in said network. In this section, a small overview into current detection methodologies is presented. Approaches have been made to detect botnets by analyzing statistical features of domain names (e.g. domain name length, used top level domain and entropy of the string) [3, 33]. While these systems are specifically designed to detect currently known botnets employing DGAs, the assumption is, that they do not perform necessarily (well) for botnets generating domain names that are pronounceable or are created with the help of a (English) dictionary. Other attempts do not make these assumptions [8, 17], but rely on a fixed periodicity in botnet communication instead. This fixed periodicity can be circumvented by a botmaster randomizing the timing parameters of all bots in such a way that no fixed periodicity can be determined. Research into IRC botnets [2] has already shown that botmasters try to communicate the least amount possible and that botmasters help each other to defeat new detection mechanisms. A more in depth look into the history of botnet detection is presented in chapter 2.

1.3. Future problems with DGA detection

From the previous section, it can be seen that current DGA detection technology makes strong assumptions about the methods of operations of DGAs. Assuming that botmasters will, in the near future, cease to use obvious randomly generated domains with a fixed periodicity, a gap of knowledge with current detection mechanisms that work with the only assumption that DGAs use the domain name system (DNS) and create new domain names for use in the botnet very often (on a daily basis) can be noticed. This thesis tries to design a novel approach to detecting DGAs active in a network, and therefore fill this gap of knowledge. Apart from the assumption that future DGAs try to circumvent current detection methods, designing a system that does this now is necessary, as there might already exist malware that is currently undetected.

1.4. Requirements for DGA detection

Before creating a new DGA detection mechanism, clear requirements for the functionality of such a system should be defined. From literature, common requirements set by network operators for detection of malicious activities on their network can be deduced. Assuming future DGAs will no longer rely on fixed periodicities of DNS traffic or using fully randomized domain names on a per character basis, which immediately

stands out to a human observer, the goal is to create a detection scheme that also follows these requirements:

- High true positive rate for detection, in order to catch all malicious activity.
- Low false positive rate, to avoid flooding network operators with non-malicious data.
- Fast detection speed, to allow network operators to quarantine infected hosts swiftly.
- Correct clustering of infected hosts, in order to give insights about subsets of the network infected with the same malware. This allows network operators to easily backtrack how malware entered a system.

During the evaluation, contained in chapter 4, results will be presented in such a way that these requirements can be evaluated, in comparison to the current state-of-the-art of DGA detection.

1.5. Research goal

This thesis focuses on, as explained in previous sections, detection of DGAs by analyzing DNS traffic. By reviewing literature the following problem statement is determined:

> Currently DGA detection by DNS traffic makes, currently correct, assumptions of the behavior of malware. Research has shown that botmasters actively try to circumvent new detection methods and help each other to stay ahead at all times. Therefore it is likely that current detection mechanisms will be circumvented in the near future.

To solve this problem, the following research question has to be answered:

RQ1: How does the current state-of-the-art of DGA detection work, and how can a DGA circumvent detection?

After answering RQ1, a new system filling gaps in the current state-of-the-art of detection technology can be designed. This new system should also abide by the requirements listed in section 1.4 and therefore the following extra research questions are defined.

RQ2: How can a new DGA detection system be designed, that does not rely on linguistics or short term periodicities?

RQ3: How can detected DGA domains be clustered together with this new system?

Over the course of this thesis, these questions will be answered and summarized in chapter 5.

1.6. Research approach

In this research, a new methodology to detect DGAs based on DNS traffic is proposed. To do this, a new DGA, which tries to hide itself from all current approaches, while still being efficient for actual use, is created. Subsequently, a system that attempts to detect DGAs that behave like the aforementioned DGA in DNS traffic is designed. Effectiveness of this method will be tested on DNS traffic from the Delft University of Technology network. Detection rates of other systems designed to detect DGAs are also compared to this new detection mechanism.

1.7. Thesis outline

In chapter 2, the current state-of-the-art of botnet and DGA research is outlined. Chapter 3 contains an in depth explanation of the approach and methodology towards designing the DGA detection system. Next, in chapter 4, the design choices made in the implementation are enlightened. Also, the effectiveness of the new approach and a comparison of the results with other algorithms designed to solve this problem is presented. Finally, chapter 5 concludes this research and reflects on the chosen design. Recommendations for future research to improve on this work is shown in chapter 6.

2

Background and related work

In this chapter, previous research about botnet detection in a network and more specifically detection of domain generation algorithms (DGAs) will be introduced. First, an explanation on how botnets operate is given in section 2.1, followed by an overview about previous research about detection of botnets in section 2.2. Section 2.3 outlines previous work into detecting botnets by detecting their domain generation algorithms.

2.1. Introduction to Botnets

A botnet is a network of computers, also called bots in this context, executing tasks as controlled by a central authority, a so called command and control server (C&C). Commonly, the term botnet is associated with malicious intentions, but legitimate uses do exist. Legitimate uses of botnets include distributed computing applications that communicate with each other to reach the result of a calculation more quickly. More common are illegitimate uses of botnets. These are often used for distributed denial of service (DDoS) attacks and phishing user credentials. In these cases the command and control server is often not located within the same network. Instead it is hosted externally and often communicates over unsuspicious protocols as Internet Relay Chat (IRC) and Hypertext Transfer Protocol (HTTP). The use of IRC for this purpose stems from the 1990s when IRC was still largely used as a legitimate chat protocol. Over time, IRC was being used less for legitimate purposes and countermeasures to specifically prevent these malicious activities on IRC networks had been deployed. Therefore botmasters started to transition to HTTP(S) which also allowed to more easily change the C&C location. A spin off of these techniques involve peer to peer (P2P) botnets, which operate differently from botnets with a central C&C infrastructure. This shift over time in research focus can be seen in figure 2.1. The presented numbers also include literature reviews discussing multiple techniques, but as they discuss aspects of all shown methodologies, all counts are evenly increased by these publications. As can be seen, by far the largest amount of research is done into detecting botnets via its DNS traffic, while research into IRC and P2P detection techniques is on the decline since 2014.

Most recent botnets use HTTP to stay in contact with all infected computers. Historically, bots have a hard-coded IP address or domain name implemented in the malware binary. Taking down single IP addresses or domain names is usually a trivial task for law enforcement and botnets utilizing this technique often don't exist for long periods of time. To circumvent getting taken down, while still using the ease of HTTP (instead of moving to a distributed peer-to-peer system), actors deploying botnets (bot masters) started to pseudo-randomly generate domain names based on a deterministic seed. This seed is often the current date, but can also be non-predictable public information (for example "trending topics" of twitter.com). This way, a bot would try to automatically connect to a new (set of) domain(s) every day which would make it harder to take the botnet down. In the case that these algorithms create more than a single domain per day, it would also be very expensive for third parties to pre-register all domains in order to take over a botnet. When no public data is used as seed for the DGA, it is also impossible for law enforcement and researches to predict domains that will be in use in the future.

In most cases, these botnets generate domain names that are, for a human, "random looking". For example Torpig, a large botnet that has been taken over for ten days in 2009 by Stone-Gross et al. [39], generates domains that look like `ueupulj.com`, `xgrrunj.net` and `vgpsuag.biz`. Often, infected hosts query the gen-

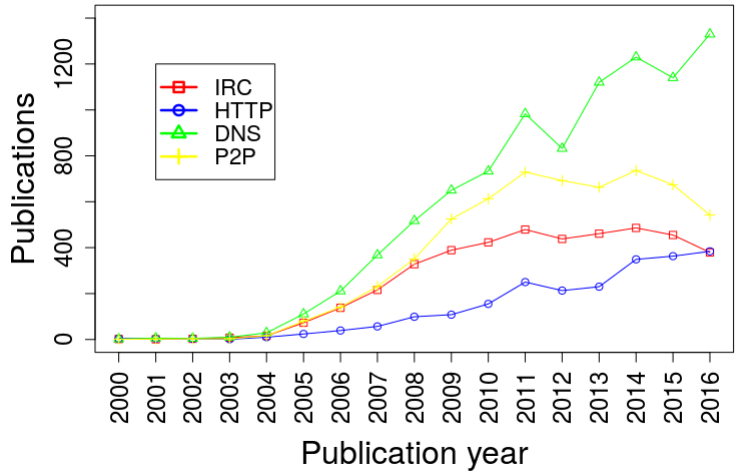

Figure 2.1: Publications about botnet research split by protocol (via Google Scholar)

erated list of domains in a locally random order, whereas the botmaster only registers a single (or at most a few) domain. This results in many NXDOMAIN replies from DNS servers (returned when a client requests a **n**on-e**x**istent domain) when the bots try to connect to their C&C.

2.2. Introduction to detection of botnets

There are multiple ways to detect a botnet. Traditionally, locally installed malware scanners would be used to detect malicious binaries on a computer. Initially, static detection methods were used for this [41]. Creating static detection signatures requires work for every sample, but most signature based anti-virus solutions can easily be bypassed by malware as the creator often just has to recompile the software in order to not be detected anymore. Anti-virus software can also check for partial signatures. These can often be bypassed as well by unpacking the actual malicious binary at runtime. Therefore research moved towards dynamically detecting malware by its behavior [32, 35].

Problems with local techniques (static analysis and behavior analysis) include being susceptible to being attacked by malware itself in such a way that it is not functioning anymore. Another problem is installing reliable detection software on all clients within a network. This is unfeasible in networks where users are allowed to bring their own hardware and connect to the network, as overseeing the state of these devices (from a security point of view) is often not possible. Also in large networks many different operating systems are in use, which would require comparable detection software for all used operating systems (mobile operating systems and laptop/workstation operating systems).

External detection algorithms, installed as network intrusion detection systems (NIDS), do not suffer from these problems, but are often limited to analyzing network traffic. This comes with the problem that inspecting encrypted traffic is not trivial in aforementioned bring-your-own-device networks as most clients will not accept encrypted communications when the contents have been read out by a third party via a legitimate man in the middle attack (MITM).

2.2.1. Detecting botnets via network traffic

In 2006, Rajab et al. [2] summarized the life cycle of 192 different IRC botnets to better understand techniques commonly employed by botmasters. They do this by collecting malware samples via honeypots and then apply gray box testing to these samples. They learned that, in 2006, 27% of all malicious traffic to their networks were a direct result from these IRC botnets. They could also determine the average lifespan of 47 days and noticed that bots already tried to keep the IRC traffic to a minimum, possibly to avoid detection. Goebel et al. [12] detect bots on IRC servers by their nicknames. By analyzing a few malware samples they noticed that nicknames of IRC bots often contain many numbers, special characters and the country of origin of the computer. To detect these bots they deploy software to the network to automatically analyze and score any NICK-command (to assign your own username on an IRC server) being used from within their network.

When the detected score is too high, all traffic for this client is stored and network administrators are warned to manually investigate this issue.

Around the same time, Binkley et al. [5] created a system to automatically cross match characteristics of IRC messages with TCP SYN packets (commonly used for denial-of-service attacks) in order to detect botnet channels and infected hosts. Livadas et al. [19] compare multiple classifiers on network flows of IRC traffic to subsequently distinguish legitimate chat messages from botnet traffic. Strayer et al. [40] see problems with botnet detection techniques analyzing IRC traffic on a static port (6667, the default for IRC). Therefore they propose a solution that detects patterns in network flows. Specifically used bandwidth, timings of packets and burst duration are measured. The proposed model can detect IRC botnet traffic running on any port without analyzing all messages directly.

Gu et al. [14] also use network flow analysis to detect botnets, but instead of detecting specific IRC botnets, they try to correlate network flows of multiple clients in a network to detect similar traffic. This relies on the assumption that multiple instances of the same malware will behave the same and show similar network patterns. AsSadhan et al. [4] also see problems in the scalability of analyzing network flows of single hosts and approach this problem with aggregated flows of multiple hosts.

Almost a decade later, Singh et al. [34] see the amount of data generated by network based detection systems as a challenge. Therefore they propose a system that analyzes full network traces in a horizontally scaling manner (with Apache Hadoop and Apache Mahout). Specifically they apply their system on detecting peer-to-peer botnets. This is done by automatically creating a Random Forest Decision Tree and applying this model to the captured data. Other work has also shown the effectiveness of analyzing network flows and applying classic clustering methods or machine learning principles (on network flows) in order to detect botnet traffic [16, 18, 22, 37, 45].

BotHunter [7] uses a correlation engine combining data of three points in a network. The goal is to detect the common botnet infection life cycle (inbound scan, exploit usage, malware distribution, outbound bot coordination, outbound attack propagation) and determine new infections within a network. Problems with this technique include botnets being very obvious in their communication and the authors expect botmasters to circumvent (parts of) this system by scanning in a stealthy manner or by using encrypted communications to the C&C.

2.3. Detecting domain generation algorithms

Detecting botnets that utilize domain generation algorithms is always done by analyzing DNS traffic of clients in a network. This makes sense because DNS traffic is not encrypted and can easily be intercepted on a network.

In literature, multiple approaches to solving this problem can be seen:

1. Analyze NXDomain DNS responses
2. Analyze features of DNS queries (e.g. features of the domain name itself)
3. Analyze features of DNS responses
4. Analyze timing patterns of DNS traffic

A fifth option would be to reverse engineer samples of currently used malware and pre-generate the list of to-be-used domains and block these in a firewall. This option requires a lot of manual labor per instance of a malware and would only detect new samples after they have been found and reversed by researchers. Doing so would also result in very long block lists which mostly contain domain names that will never actually be used for malicious intents. Following is a discussion of multiple attempts of utilizing these four approaches and analyze the advantages and disadvantages of these approaches are presented.

2.3.1. Analyze NXDomain DNS responses

Analyzing only NXDomain DNS responses relies on the assumption that malware using DGA algorithms will regularly try to contact non-registered domains. This would only happen when an infected host (bot) generates many domain names and queries them in a locally random order or if a bot master abandoned their botnet and does not register currently queried domains anymore.

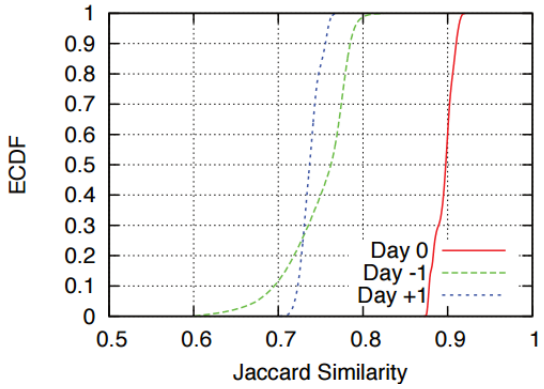

Figure 2.2: Jaccard similarity for Conficker domains by Thomas et al. [42]

Antonakakis et al. [3] assume that computers in a network, infected with the same malware, will generate similar DNS queries that result in similar NXDomain responses. They created a system, called Pleiades, that works in multiple steps. At first, it collects NXDomain responses generated during a fixed time, e.g. a day. Then these collected responses are clustered based on 33 statistical features, e.g. similar length, similar level of 'randomness', similar character frequency distribution and the overlapping set of requesting clients in the network. As this step is completely unsupervised, the result will also contain common typos of legitimate domain names, e.g. `googel.com` instead of `google.com`. To prevent detection of these typo domains, a filtering step is applied that removes clusters of previously detected DGA domains and domains that are very similar to popular legitimate domains. Clustering of domains is done using random groups of ten NXDomain responses with the X-means clustering algorithm [28]. X-means is an extension to the k-means clusterer with an automatically determined value for k.

After detecting these potential DGA domains, the actual classification and C&C detection step starts. For this, two statistical models are used. The first uses a classifier to assign a specific DGA label to a set of generated NXDomain responses requested by a host h_i. The second one uses a Hidden Markov Model (HMM) focusing on finding an active domain queried by host h_i belonging to a DGA. While this approach will often work for current DGAs, the statistical features can easily be made to look like legitimate domains. This can be done by not concatenating random characters in the domain name, but instead concatenate pairs of characters as they would occur naturally in, for example, the English language.

Another approach by Thomas et al. [42] proposes to analyze NXDOMAIN DNS traffic "at several premier Top Level Domain (TLD) authoritative name servers". They do this by calculating the Jaccard index[1] of two sets of recursive name servers A and B requesting two different domains X and Y. According to the authors, the Jaccard similarity for NX domain responses of the same DGA is often above 0.9.

Figure 2.2 shows the Jaccard similarity of the generated domains of Conficker [2] for three specific days (accounting for infected machines with an incorrect date) based on their /24 recursive name server traffic. As can be seen, the similarity for the set of recursive name servers is very high for NXDOMAIN responses for domains generated by Conficker for the current day (Day 0 in figure 2.2)

This approach works because many botnets randomly query a list of generated domains. By preventing the bot from querying domains that are not registered yet, there would hardly be any data (NXDOMAIN responses) to compare and the Jaccard index would be very low.

2.3.2. Analyze features of DNS queries

Schiavoni et al. [33] created a system called Phoenix. Phoenix uses a set of linguistic features on the domain name to initially determine whether or not it might be a legitimate domain. To do this, the researchers first calculated these features for the top 100000 Alexa[3] domains. For any subsequent DNS request, these features are also calculated and the Mahalanobis distance [20] to the centroid of the cluster of the Alexa domains is calculated. When a fixed threshold is reached, IP-based features are extracted and Phoenix tries to cluster will cluster domains together that "resolve to the same pool of IPs". When a cluster is found, the fingerprint of the

[1] $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$
[2] `https://en.wikipedia.org/wiki/Conficker`
[3] `http://www.alexa.com`

requests is stored to later classify new domains immediately as belonging to a certain cluster.

This approach assumes that botmasters only have access to a limited set of hosting providers, which does not have to be the case as botmasters are known to use, untraceable to the botmaster, illegally taken over webhosts to temporarily act as command and control server [30]. It also assumes, as Antonakakis et al. [3] do, that the generated domain names seem to be completely random when compared to natural languages.

2.3.3. Analyze features of DNS responses

Choi et al. [8] propose a system, the BotGAD framework, which collects all DNS traffic passing through. It stores this traffic with the requesting IP address and the corresponding timestamp. Per domain name, BotGAD creates a matrix of timeslots and requesting IP addresses. To find computers that are infected with the same malware, the cosine similarity coefficient [44] is calculated in order to determine the similarity of these matrices. Then the data is enriched to calculate lexicological features and the distinctive features of the DNS queries. For DNS features, BotGAD focuses is mainly on the Time To Live (TTL) of the DNS records, information about the autonomous system (AS) and hosting country (resolved by the database of MaxMind [4]. A short TTL is a good indication for Dynamic DNS (DDNS) [6] or Fast-Flux Service Network (FFSN) [15] usage. To function correctly, BotGAD relies on linguistic features and comparable timing patterns for DNS requests between different infected hosts. Both can be hidden by a botmaster by more carefully generating domain names and by randomizing the local timing parameters of the bot.

2.3.4. Analyze timing patterns of DNS traffic

PsyBoG [17] approaches the problem of botnet detection by assuming that botnets have periodic and simultaneous behavioral pattern. To reduce the data to be processed, only DNS traffic is used. In order to find the periodicity of a host, all DNS traffic is stored with the requesting IP address and timestamp. Then a periodicity analyzer determines the power spectral density (PSD) [38]. This PSD will contain multiple local peaks, but not all of them are significant (see fourth graph of figure 2.3). To find significant peaks, the system analyzes if the PSD "has a distribution proportional to χ^2 in two degrees of freedom". If this is not the case, the system assumes that the highest peak is a significant periodic component.

After finding hosts with a significant periodicity in DNS traffic, PsyBoG tries to determine groups of hosts with the same pattern. This is done by calculating by applying the similarity measurement algorithm pDist (Power Distance) [43]. When the power distance between two periodograms is lower than a previously defined threshold, two hosts are marked as belonging to the same group of a botnet.

Unlike most other recent research, PsyBoG does not continue on the path of assuming certain aspects of randomness in the generated domain names and it only assumes that bots have a fixed periodicity of contacting a domain name. While this is probably more future proof, it would still allow a botmaster to locally randomize these periodicities or only use DNS whenever non-malicious software also uses DNS (hiding in legitimate traffic) and become effectively undetected.

2.3.5. Reflection on used techniques

In short, comparing the previous four options show that detection mechanics based on characteristics used by current domain generation algorithm could possibly be circumvented in the future by botmasters trying to specifically craft new algorithms to prevent detection. Assuming botnets will continue to use DGAs in the future to easily move their C&C infrastructure around, the only characteristic required to stay is frequent DNS traffic towards generated domain names. Therefore, in chapter 3 an alternative method of detection is proposed based on detection of suspicious patterns in DNS traffic.

[4] `http://dev.maxmind.com/geoip/legacy/downloadable/`

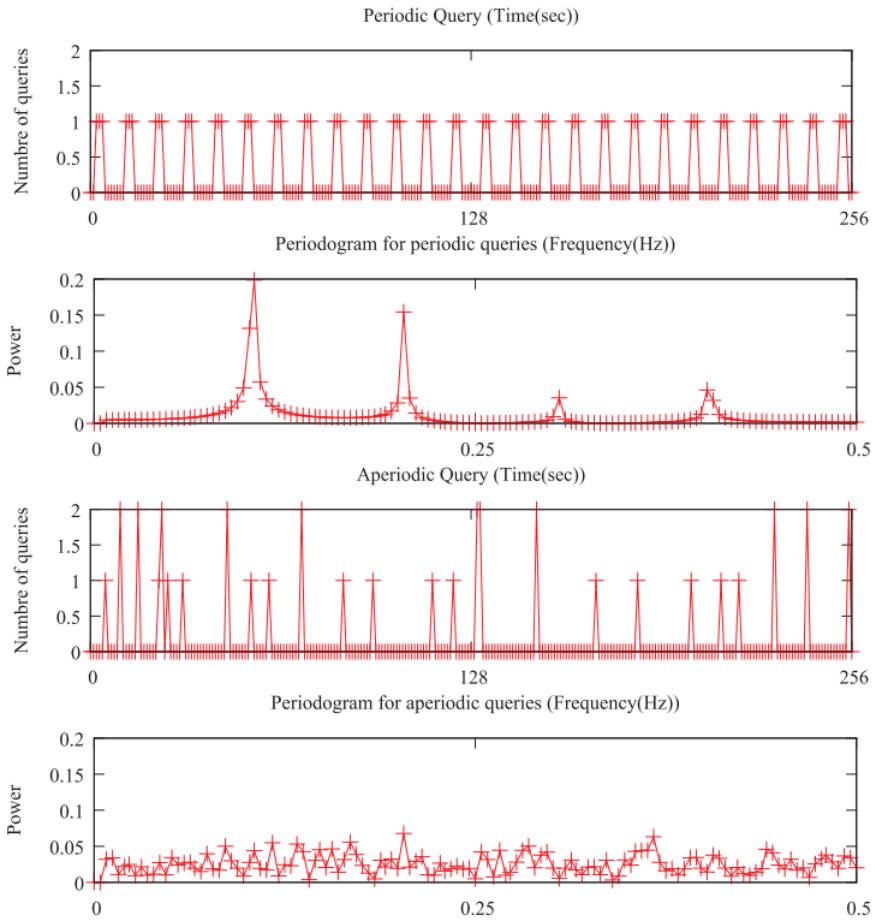

Figure 2.3: Periodograms for periodic (upper two) and aperiodic (lower two) signals by Kwon et al. [17]

3

Methodology

In this chapter, a high level overview of the new DGA detection method is described, and an explanation on how the functionality will be verified is given. This is started, by explaining how future DGAs might try to circumvent current detection methods. Secondly, details about implementations of other algorithms the new detection system is compared to are explained. Finally, a high level explanation of the novel approach to detect suspicious clusters of domain names is presented, accompanied by a computational complexity analysis of said algorithm.

3.1. Future Domain Generation Algorithm

As described in chapter 2, currently DGA detection is based around fine tuned timing and lexical characteristics of DNS requests. It can be expected of botmasters to create DGAs that do not show any of these traditional patterns. Therefore, in order to eventually verify the effectiveness of the new detection system, a DGA working in such a way needs to be created. The proposed DGA design has a, categorized by nouns, verbs, adverbs and adjectives, English word list of more than 150,000 words. With the current date as seed for a random number generator, these words are then concatenated in a grammatically correct order. To prevent desynchronizations between clients, the current date is retrieved from the webservers of popular websites. Specifics about the implementation of this DGA can be found in section 4.1.

Examples of domains generated by this algorithm include `somekithedtaguan.be`, `the-estranged-express.com` and `besprinklingbanc.org`. When superficially glancing over logs of DNS queries, a network administrator would most likely not see any obvious red flags for these domains. As these domains purely consist of dictionary words available in a modern English dictionary and the lengths of the generated domain names highly fluctuate, it should be impossible to detect these domains purely by analyzing statistical lexicological features. A problem with this DGA is that it occasionally generates domains that already exist. In practice this can be ignored as malware implementing this DGA would check if the domain is controlled by the botmaster instead of assuming a valid reply from the nameservers implies that the tested domain name is valid for this DGA.

3.2. Verification of detection

To verify the correct functionality of a new detection algorithm, detection rates of current state-of-the-art detection mechanisms on this new type of DGA are compared to this new system. Also, a verification of whether or not the requirements listed in section 1.4 are met is presented. For comparison, specifically Phoenix [33], Pleiades [3], BotGAD [8] and PsyBoG [17] were implemented during the work on this thesis, providing the ability to run test cases on all algorithms easily. These algorithms were already briefly described in section 2.3, but in the following subsections, implementation details of these algorithms are discussed and an explanation why these might possibly not work with the proposed DGA used in the future are given.

3.2.1. Phoenix

Phoenix tries to find clusters of malicious IP addresses. In order to find these, it combines a blacklist of domain names or domain reputation system and calculates linguistic characteristics of all domain names it sees. Once it finds a cluster of malicious IP addresses, it does not rely entirely on detecting new domain names linguistically anymore as it maps any new domains resolving to the same set of IP addresses. As the generated domain names by this new DGA do not differ from legitimate domain names (linguistically), Phoenix should be unable to find these domains, as it uses a model trained on illegitimate domain names. These domains could end up on a blacklist, but Phoenix only marks a domain to be processed (to extract features of the DNS replies) when a query for a specific domain is seen in the network. Therefore, Phoenix will never link those old blacklisted domains to the ones currently in use by the botnet. Because of this, the used implementation does not make use of any blacklist or domain reputation system. In the case that a new DGA uses dictionary words, the model would be trained on these new type of domain names, which would also linguistically look like legitimate domains, resulting in the model flagging legitimate domains as well as illegitimate domains.

3.2.2. Pleiades

As described in section 2.3.1, Pleiades uses a statistical model to classify domains as either malicious or benign. With the help of WEKA[1], a model based on known DGAs from the DGArchive and the top 100000 alexa.com domains was pre-calculated. As classification is mostly determined by the linguistic aspect of domain names, this algorithm cannot perform well with domains generated by natural language dictionaries. Also, as can be seen in section 4, detection is generally very inaccurate as it randomly clusters sets of ten domains together in order to speed up the algorithms execution.

3.2.3. BotGAD

Choi et al. designed BotGAD to work in two steps. The first creates a binary matrix per occurring domain name marking in which time slot, indicated by the columns, which IP addresses, indicated by the rows, requested the given domain name. This matrix is used to measure temporal similarity. In order to not be susceptible to missing domains by a DGA not having a fixed periodicity, or by parts of a botnet being added or removed at certain moments in time, outliers are removed from these matrices. After this filtering, per requesting IP address, the periodicity is determined, and if deemed similar to the periodicities found for other IP addresses, it is marked as suspicious. The second step of BotGAD calculates the cosine similarity between all suspicious matrices and compares features of the DNS traffic (linguistics, query and answer features) in order to cluster domains belonging to the same DGA together. This last step was not implemented. This does not have any negative influence on the results presented in chapter 4 as during comparison, an optimal clustering is assumed.

3.2.4. PsyBoG

In order to detect hosts infected with malware utilizing a DGA, PsyBoG determines significant periodicities in DNS traffic by calculating the power spectrum density (PSD) with the help of fast Fourier transformations (FFT) over a sliding window of traffic over any given day. Performing these PSD calculations is straight forward, as many software tools already exist to do just that. Finding significant peaks within these PSDs is also just a matter of comparing the values in the PSD to a, by Kwon et al. determined, threshold. If that threshold is surpassed, the algorithm flags the input signal consisting of all timestamps of a host issuing a DNS request. The difference with the other implemented and tested algorithms is, that PsyBoG does not directly detects suspicious domains but instead only flags hosts showing strong signs of periodic DNS traffic. Afterwards, a network operator can look into which specific domains any flagged host contacted during the flagged period and, if necessary, disable the affected hosts.

3.2.5. Summary

As described in the previous four sections, for most algorithms, only the detection part of other algorithms were implemented. This was done to speed up development and not spend time on fine tuning the parameters used during the clustering phases of these algorithms, which could lead to inaccurate results, compared

[1] http://www.cs.waikato.ac.nz/ml/weka/

to results retrieved by their respective authors. The downside of this choice, is that effectiveness of the clustering part cannot be compared to the full extent. During comparison, the assumption was made that all algorithms cluster related domains or hosts together as optimally as possible, as long as the detection part, flagging individual domains or hosts marked such a domain or host as malicious in the first place. Partially, the quality of the clustering can still be reasoned about, as for example when a DGA contacts 200 domains, yet the detection algorithm only detects a single domain of the malicious set, but substantially more non-malicious domains, it is evident, that in the most optimal clustering scenario, this would leave the single malicious domain in a cluster on its own, with only that single domain. In practice, when the full cluster would have a size of 200 domains, the single domain would not stand out in between the noise of non-malicious traffic marked as malicious. In depth results about the performance of the detection performance of these algorithms can be found in chapter 4.

3.3. Proposed alternative method of detection

The key differentiating part of the algorithm presented in this thesis is that no assumptions on the structure and mechanics of the domain name candidates used within the DGA are made. However, implicitly it is assumed that domain names used for botnets are only used for a short period of time, as long-term malicious domain names would otherwise be trivially filtered by a defender. As the adversary will avoid this situation, DGA-based DNS traffic will be transient. Therefore, the idea is to calculate the popularity of domain names and detect sudden increases and decreases of traffic over multiple days. This allows for a very simplistic, yet efficient method in detecting DGA anomalies. The functionality of this method is described in detail in the sections below starting with the required input data for the algorithm to function in section 3.3.1, followed by the method used to filter out the vast majority of legitimate domains in section 3.3.2. After this initial classification, sections 3.3.3 and 3.3.4 discus how the algorithm handles clustering related domains together, which is essential for network operators in order to make the result easier to interpret. Finally, a theoretical analysis of the time complexity of the algorithm is presented in section 3.3.5.

3.3.1. Used input data

The problem with most other methods of detecting DGAs in networks lies with making very strong assumptions about the characteristics of currently active DGAs. In order to detect these characteristics, many features of the occurring DNS traffic are used. As detection of DGAs should not be restricted to highly specific instances, the proposed algorithm only uses a very limited amount of data. The minimum information required for the algorithm are triplets of a host I requesting a domain name D on specific date T. Using more information could be detrimental to the detection rate in the future, as botmasters might come up with techniques to effectively mimic either legitimate DNS questions and answers, or randomize this information enough to not raise suspicions. Currently, this is often not the case, and domains registered for a DGA have similar patterns in registration and DNS records, due to automatization on the botmasters end.

3.3.2. Suspicious domain detection

Assuming that a network mainly consists of non-malicious hosts, the majority of the traffic encountered on any given day will be non-malicious as well. Processing this non-malicious traffic will only have a negative effect on the time required for the algorithm to complete. Therefore, the proposed algorithm tries to filter out all non-malicious traffic before it clusters similar traffic together. All filtered traffic can then be assumed to also be clustered together in a “non-malicious” cluster.

After importing all data, the list of domain names is filtered to only include domains that are only requested on one to three consecutive days and not before or after that. Doing so, (permanently) filters out popular domains that are very unlikely to be used as DGA. Examples of legitimate popularities of domain names can be seen in figure 3.1. While the vertical axis is scaled differently per plot of the popularity, a similar pattern on the three largest shown services can be observed. Also smaller services like a (small) mobile game have traffic on several days in a row and would therefore be effectively filtered out. In contrast, a typical pattern of a non-existent domain used by a DGA in figure 3.2 stands out. The dip on January 13th, 2017 can be attributed to a lack of storage capacity, which resulted in the network capture not storing data between 6 AM on the 13th of January until 11 AM the following day.

Filtering all traffic in the previously described manner reduces the amount of domains used for further

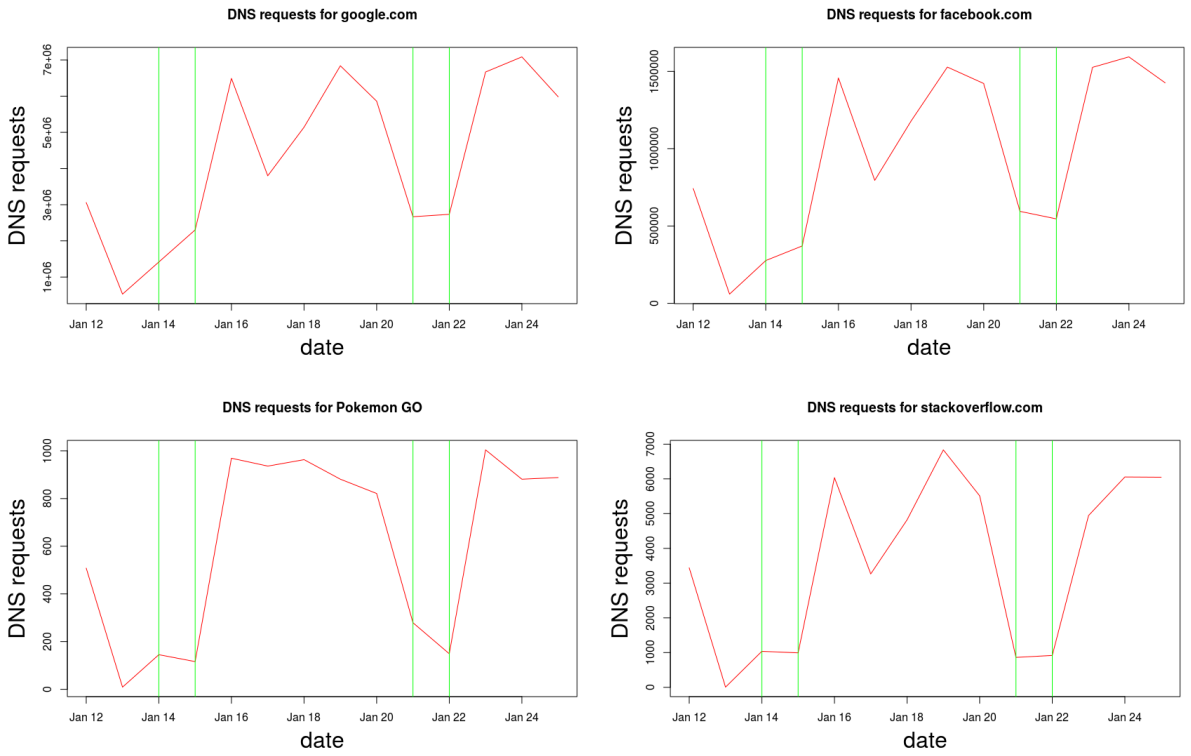

Figure 3.1: Popularities over time of Google services, Facebook, Pokemon Go and StackOverflow within the network. Saturdays and Sundays are indicated by a green vertical lines.

steps in the algorithm to approximately 20% of the original size. By the computational complexity shown in section 3.3.5, this has a significant impact, as the slowest part of the entire algorithm has a time complexity mostly attributed to the amount of domains left in the clustering step.

3.3.3. Related host detection

After importing all data, the list of domain names is filtered to only include those that are only requested on one to three consecutive days and not before or after that. From these remaining domains, all IP addresses that requested similar domains are looked up. The rationale behind this approach is to not miss any domains contacted by a DGA contacting domains in a locally random order which might not get contacted by all infected hosts. If this would not be done, these domains would still get recognized as malicious, but cannot be clustered together as belonging to the same botnet afterwards. A drawback of this method is that unrelated domains can easily be involved into a cluster consisting of DGA domains.

In practice, the list of related hosts is found in the following way:
Given a domain D:

1. Find all IP addresses I that requested a domain D
2. Find all domains D' requested by all IP addresses in I
3. Find all IP addresses I' that requested domains in D'

A visual representation of this process is given in figure 3.3. After this process, the algorithm has a list of domain names with corresponding sets of hosts. Another description of this process can be described as having a bipartite graph, with one set of vertices representing all suspicious domain names, and another disjunct set of vertices representing the hosts requesting these domain names. Then, retrieving the set of broadly related hosts I' for any suspicious domain D can be interpreted as traversing the graph in a breadth-first manner starting with a single suspicious domain from D and going to a depth of three, marking all passed elements in the process. After reaching a depth of three, halting in the set representing the hosts, this process

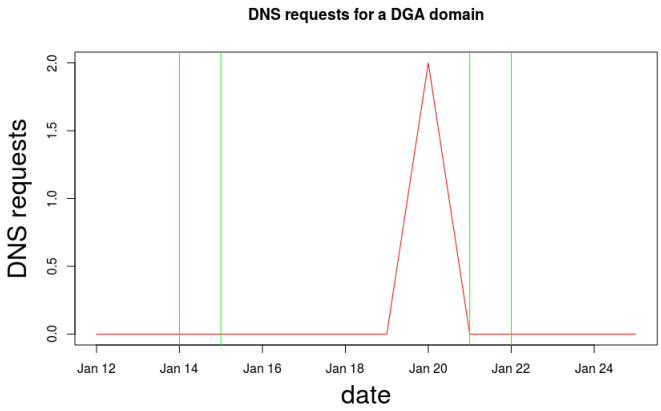

Figure 3.2: Popularity over time of a single DGA domain

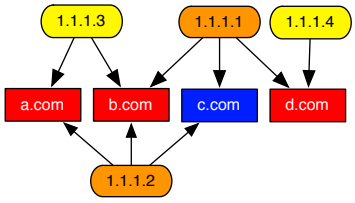

Figure 3.3: Method used to find IP addresses I' broadly related to a domain D. In this figure, D is indicated in blue, D' in red and blue. The elements of set I are colored orange, I' yellow and orange.

is stopped and all marked vertices define the set I'. All vertices are unmarked again, and the process repeats for the next suspicious domain.

3.3.4. Clustering suspicious domains

After collecting all broadly related IP addresses per suspicious domain name, the Jaccard similarity of these sets of IP addresses to all other sets is calculated. Based on an adjustable threshold, the algorithm is now able to cluster domain names together based on the similarity of the requesting hosts within the network. In order to limit the amount of output, the algorithm makes use of two more variables: the average amount of unique hosts requesting a domain in a detected cluster and the minimum of hosts requesting a domain required for it to be included in a cluster. The rationale for this is to limit the output to clusters that are requested by multiple hosts within the system, rather than focusing on single hosts. This will help network operators to identify large, highly impacting DGAs active in their network quickly.

3.3.5. Computational complexity

The input to the algorithm designed in this thesis consists of a list of domain names, with per domain name a set of tuples consisting of the date and host requesting that domain name. The first step counts the amount of requesters per domain name per day. The input data is stored in `HashMaps` in Java, with one key per domain with all occurring dates and requesting hosts in a new HashMap as value. The computational complexity for this data structure guarantees constant time adding of elements, and the size, representing the amount of unique hosts requesting a domain name on a day, is stored as a separate value, hence requesting this size also only requires a constant time. Therefore, given the set of domains D with unique dates M, this computation has a computational complexity of $\mathcal{O}(|D| \cdot |M|)$.

The result is a list of domain names with a corresponding list of the amount of requesters per day, hereafter named the popularity of a domain. Per domain name, the algorithm iterates through these popularities and tries to find domains, that only have a popularity of more than 0 for one, two or three consecutive days. After finding these domains, the algorithm effectively has a graph of hosts requesting these suspicious domains. In order to find the related hosts, as described in section 3.3.3, a breadth-first search is applied until

the third level of the graph. In the worst case, at this time, the algorithm still has all domains D, with all hosts H requesting all domains marked a suspicious. As nodes in the graph are alternating between domains and hosts, and this step always starts with a domain, the time complexity is at most $\mathcal{O}(|D| \cdot |H|)$. During the traversal of the graph, sets of hosts related to any given suspicious domain name are created, which is the input of the next step.

From these sets, the Jaccard similarity between all other sets is calculated. Calculating a single Jaccard similarity between two sets takes at most $\mathcal{O}(|H|)$ and at most $\frac{|D| \cdot (|D|-1)}{2}$ of these similarities need to be calculated, resulting in a combined complexity of $\mathcal{O}\left(|D|^2 \cdot |H|\right)$. The final step of the algorithm creates sets of domains of which the related hosts have a Jaccard similarity above a threshold, as determined in section 4.3. In the worst case, all sets of hosts abide by this feature and the algorithm needs to iterate over all domains, yielding a time complexity of $\mathcal{O}(|D|)$. All mentioned parts are executed in sequence, resulting in a total maximum time complexity of $\mathcal{O}\left(|D| \cdot |M| + |D| \cdot |H| + |D|^2 \cdot |H| + |D|\right)$, of which the largest component is $|D|^2 \cdot |H|$. This non-lineair increase in time complexity can become problematic on very large datasets, but during the evaluation of the algorithm in chapter 4, no issues could be noticed.

3.3.6. Summary

In short, the proposed algorithm uses a simplistic approach in order to detect DGAs in a network. The only assumption this approach uses is the fact that DGAs request different domains on a daily basis, with shared requested domain names over all infected hosts. In practice, DGAs change their set of valid domains anywhere between 3 and 24 hours. While the algorithm allows a grace period of three days for a domain to be contacted, before it is whitelisted as non-malicious, extending the validity period of a set of domains to a longer period would be detrimental to the functionality of a DGA as this would give authorities more time to preemptively block registration of the used domains. The second part of the algorithm tries to cluster domains into subsets based on the similarity of the set of hosts requesting these domains.

4

Evaluation

To show the effectiveness of the algorithm proposed algorithm described in chapter 3, it is tested it on both real datasets and various constructed networks. Section 4.1 begins with explaining how a potential new DGA will function and how it was implemented during the work of this thesis. This DGA will be used in most of the tests applied to the various algorithms in this chapter. Following, in section 4.2, retrieval of a real dataset from a network containing more than 22,000 hosts followed by counter measures taken to prevent deanonymization of that data is shown. Then, in section 4.3, the new detection algorithm is applied on this real dataset and a feasible threshold required to cluster suspicious domains together, as explained in section 3.3.4, is determined. Finally, in section 4.5 to 4.7, various artificial networks are constructed, in order to compare the influence of the behavior of a botnet on the detection accuracy, runtime and memory consumption between other detection techniques and the proposed algorithm.

4.1. Natural domain name generation

Currently, domain generation algorithms concatenate random letters and morphemes, to obvious unnatural domains such as `bqwqeiswupyny.org` and `xhlwkqdawjdpi.info`. While a human can easily detect domain names such as these to be suspicious, the problem becomes larger given the incredible amount of legitimate domain names in use: in 2016 approximately 326.4 million domain names were registered. Another aspect of detecting these unnatural domains is the variety of languages used around the globe. In the past, DGA detection was often based on detection of seemingly arbitrary concatenation of characters, and some authors even claimed that creating a DGA based on concatenating words from a natural language would not yield a large enough set of domains to be functional in practice [33], yet during the work on this thesis, a working prototype of such a DGA was constructed as follows.

In order to create natural looking English domain names, a dictionary, which has information about linguistic properties of the contained words (classification of nouns, verbs, adverbs and adjectives) is retrieved. Such a list can be found with the Moby Project[1]. The used lists contain 90,963 nouns, 30,802 verbs, 6,276 adverbs and 28,479 adjectives. Using a seeded random number generator, to ensure generation of the same domain names over multiple systems at all times, random amounts of words from these lists are concatenated in a grammatically correct order. The only element that is guaranteed to be in a generated domain name by us, is a single noun. This chance is extremely slim, though.

To ensure having the same seed across multiple systems, the current date is used. As this might be set incorrectly by clients, the algorithm attempts to retrieve it from four large websites, namely google.com, facebook.com, baidu.cn and stackoverflow.com. The assumption being, that at least one of these websites is reachable by an internet-connected host. When contacting these hosts over the Hypertext Transfer Protocol (HTTP), the servers should reply with a “Date” header, as specified by the Internet Engineering Task Force’s (IETF) Request for Comments (RFC) 7231[2]. If all hosts are either unreachable or refuse to reply to our clients, the DGA falls back to the clients local date setting as retrieved from the operating system.

[1] http://icon.shef.ac.uk/Moby/
[2] https://tools.ietf.org/html/rfc7231#section-7.1.1.1

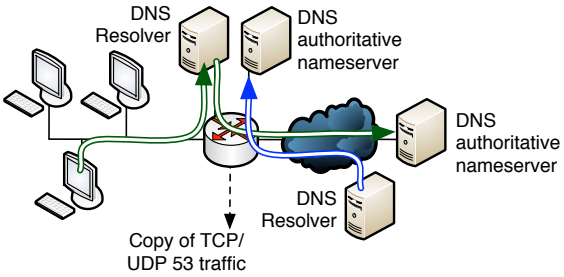

Figure 4.1: DNS queries and responses were collected through port mirrors on the core routers, thereby intercepting both internal and external request/response pairs from internal hosts and external DNS advertisers and resolvers.

As the domains generated by this algorithm may look like domains used by humans, it is very possible for a domain to be already registered. In order to not be influenced by this, malware using a DGA like this should generate enough domains per day to make sure there is at least one non-registered domain. Also the malware itself should be made to first verify if a contacted domain is legitimate or used by the botmaster. This could easily be done by cryptographically signing commands sent to bots with public-key cryptography. That way, when e.g. generating the domain facebook.com, bots will immediately know that the retrieved data is not meant for the bot. The implementation of such a feature lies outside the scope of this research. The original claim of this approach of randomly generating domain names not having a large enough set of possible domain names can not be confirmed, as the approach described here yields approximately $6.076 \cdot 10^{29}$, or 607.6 octillion, different domain names. In practice, that amount is enough to keep a DGA running for a long time.

4.2. Data collection from real network

The evaluation of anomalies using a synthetic evaluation context runs the risk that an algorithm's performance is heavily influenced by the rules and procedures used for creating the evaluation dataset. In order to create a realistic evaluation scenario, the DGA detection algorithms in this thesis are compared against an actual snapshot of DNS traffic obtained from a research network backbone. This section discusses the method used for collecting and anonymizing the dataset to safeguard the privacy of the network's users. Anonymizing the dataset is essential, as without anonymization detailed profiles could be created about specific individuals, without the affected network's users explicit consent, which would be a great breach of trust towards the user.

4.2.1. Data Acquisition

In order to eliminate potential biases and artifacts from a synthetic workload and network traffic, an evaluation scenario based on the actual DNS traffic generated by a population of 22,329 hosts was developed. The data trace was obtained by installing a filtering rule on core routers in a research network that would deliver a copy of any TCP and UDP traffic on port 53 leaving and entering the network as well as in between the network's organizational zones. Figure 4.1 shows the abstract topology of the network.

The filter on the core router would hence see every DNS request leaving the client network either towards the network's DNS resolvers (or the Internet if the client were reconfigured to use an alternative recursive resolver), packets querying the authoritative name servers hosted within the research network, as well as any recursive look ups from the network's internal resolvers to other name servers on the Internet. The analysis presented in this thesis spans a time period of 14 days, in which a total of 612,011,001 DNS look ups and responses were recorded. Although data was collected from January 12th, 2017 until January 31st, 2017, the analysis was limited to only 14 days to speed up the analysis. Large parts of the algorithm can be run in parallel, but as the network captures were stored on hard drives, and had to be read sequentially per file, this time consuming step could not be parallelized. From the first day, data was only available starting from 12 pm, therefore analysis was started a day later. The total available dataset is 1.63 terabyte in size, while the chosen 14 days together comprise of 1.11 terabyte, or approximately 68% of all available data.

4.2.2. Data Anonymization

As DNS look ups precede the bulk of network traffic and operating systems conduct minimal caching of responses, the sequence of DNS requests provides a highly accurate view of the activities of a client and thus individual user. This makes it critical to anonymize the data sufficiently to safeguard the privacy of the users, this section will describe the anonymization protocol used, which was developed with research network's privacy officer.

Recall from figure 4.1 that the collection point at the core router provided three basic types of DNS traffic flows: first, DNS requests from the local area network to the network's internal resolvers and soon after responses from the internal resolvers to the internal clients, second, traffic between the internal recursive resolvers to external authoritative DNS name servers, and third, – in case of a look up of a DNS record somewhere else in the Internet that is hosted by the research network – traffic between external DNS resolvers and the network's own authoritative name servers. This means that the network can be characterized into three zones, (a) the internal network of clients, (b) the DNS servers hosted in the research network, and (c) any host external to the network.

Traffic flowing from (b) to (c) is not privacy-sensitive. The authoritative name servers for DNS zones are public knowledge, the recursive look ups contain no content to a local machine and the recursive DNS look ups aggregated over tens of thousands of simultaneous users do not allow an identification of an individual –. Requests from zone (c) to (b) and even more drastically from zone (a) to (b) may however be linked back to an individual person. Local laws and policies stipulate that such a linkage is established via the IP address as a personal identifier (PII), but application of a non-reversible transformation of this function is sufficient, as it prevents a look up of a specific activity and assignment towards to concrete PII and person. If the transformation is deterministic, this provides the added benefit for cross-correlating sequences of look ups – as can be expected from a DGA – while ensuring the anonymity of the originating user.

Such non-reversible but deterministic transformation is trivially accomplished using a hash function, in our implementation SHA-256. Since the set of IP addresses is known and denumerable, a basic hash of the IP address does however not suffice. Instead, a unique salt, that is random data used as additional data in a one-way hashing function, was generated for each IP address x and each IP address on the internal network and the Internet hashed together with its unique salt ($s(x)$ to obtain the anonymized $y \leftarrow \text{hash}(s(x)||x||s(x))$. This procedure guarantees that given an anonymized record y it is not possible to find the corresponding IP address x, and as well as given a known IP address x it is not possible to determine the associated DNS records in the dataset.

As the goal is to find infected hosts on the internal network, it is essential to know between which zones a DNS request and response actually flows. To preserve information about the local of a host in the anonymized IP address, the highest bit of the network address is set to 1 if the address was part of the internal network (zone (a)), a 0 indicating a position somewhere on the Internet (zone (c)). A regular DNS look up to an external domain would thus show the flow pairs {(1??????? → network-resolver, -domain name-), (network-resolver → 0???????, -domain name-)}, while a local client trying to bypass the internal recursive resolver will leave the record (1??????? → 0???????, -domain name-). Although above procedure will slightly increase the chance for collision in comparison to the collision probability of the hash function itself, experimental evaluation has determined the effect to be irrelevant in practice. The original data and salts were maintained and processed by a separate party, and only the anonymized traces used in the research.

4.2.3. Data parsing and filtering

The proposed method of detecting DGAs in networks relies on DNS traffic, therefore it is necessary to extract all relevant features from this traffic before the algorithm can work on the data. DNS traffic enters our system as packet captures (pcap) containing all packets the nameservers encountered over port 53 (used for DNS) as described in section 4.2.1. Each file contains one hour of traffic and is parsed with `libpcap`. Processing is done by iterating over all packets in a given pcap file. As the pcaps contain more than just DNS traffic, the first step to parse the data is to filter out these unwanted packets. The pcaps not only contain the requests from within the network to the networks nameservers, but also from these nameservers to the external authoritative nameservers, and external requests about names that these nameservers are authoritative for. Since only infected hosts from within the network are of interest, traffic to and from external sources is discarded. Also, traffic where the research network's nameservers replied with the authoritative answer (AA) flag, are ignored, as during this thesis, it is assumed that the network itself does not facilitate domain names used for DGAs. For all remaining requests, the requesting IP address, the affected domain name and the exact time are stored

in-memory.

After iterating over all packets in a pcap, the accumulated data is reduced to only store a mapping of IP addresses requesting specific registered domain names on a given day (without the exact time).

CDNs and other load-balanced services create a large amount of fully qualified domain names, where each domain name is only requested a few times across all network hosts. In order to reduce the amount of noise and fluctuation and match requests which in practice functionally belong together, only the name of the registered domain is used to significantly reduce the amount of data as a DGA may chose to prepend more data to lower levels of the domain name. In practice, the Mozilla's Public Suffix List[3] was purposely selected for its relative comprehensiveness of all public suffixes. Given this list of suffixes the algorithm takes one extra level (e.g. test.example.org is reduced to example.org). From this dataset, the total amount of requests to a specific domain on an hourly basis is calculated. Finally, when all files have been processed, this data is further reduced to store the amount of requesters per domain name for every day encountered in the original data set. In summary, after parsing all incoming DNS traffic, what is left are mappings from hosts requesting specific domains and aggregated counts for the amount of requests of any domain name on a daily basis spanning the entirety of the time range of the input data.

4.3. Evaluation on live traffic

In order to evaluate our system, the algorithm is tested on 14 days of DNS traffic generated from the 22,329 internal hosts. Establishing a baseline truth of all true positives is a hard task, as after all it would be necessary to have a complete set of all malicious domain names currently in use by botnets. No such list exists, as it would be an oracle solving the same problem which would make any algorithm trying to detect all DGAs obsolete.

As an alternative, a randomly selected subset of all 22,000 hosts is classified as "malicious", and injected DNS traffic resembling that of a botnet into the original dataset. Each malicious bot would request, in a locally random order, up to 100 domains per day with at least one domain being requested by all hosts. As the algorithm does not take into account the specific structure of the used domain names, the used domain names are all of the same format containing an incrementing counter which easily stand out in the midst of the real traffic, and can also be easily removed from the system again, to not leave a trace in the original data set.

This experiment was applied with different infection rates of the network, specifically 0.02%, 0.1%, 0.5%, 1.0% and 5.0%. This experimental setup thus easily allows evaluation of how much of the inserted traffic is detected by the algorithms, and how accurate the detected clusters represent the entirety of the botnet.

In order to fine tune the detection system, a sensible threshold to use for the Jaccard similarity between the set of requesting hosts used to cluster two domains A and B together as belonging to the same botnet needs to be determined. For this, the two variables described in section 3.3.4, used to limit the amount of detected clusters, are set to low values to not filter out any potential clusters at all. If these variables responsible for filtering the result would be used, the comparison to other algorithms would be unfair, as no filtering is applied to their output either.

In figure 4.3 and figure 4.4 the precision, the fraction of relevant domains detected relative to the total amount of domains detected as suspicious, and recall, the fraction of relevant domains detected in relation to all suspicious domains, of the detected cluster(s) containing the injected test botnet can be seen. In the ideal situation, both precision and recall have the highest value of 100%, as this indicates that all suspicious and no benign domains were detected. As can be seen, a threshold of 70% for the Jaccard similarity results in both a high precision and recall. For the smallest set of infected hosts, 0.02%, even higher similarities result in a high precision and recall, but the algorithm detects the domains contacted by the hosts as multiple different clusters, whereas starting from 70%, the injected domains are detected as a single cluster. In all further experiments, this value of 70% is used which yields desireable results in both the test dataset and randomly constructed networks.

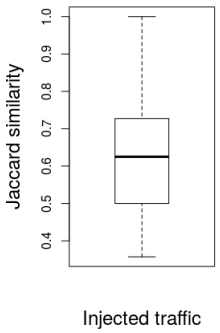

Figure 4.2: Jaccard similarity between all injected domains for 0.02% infected hosts

As can be seen, using a lower threshold for the Jaccard similarity results in significantly lower precisions in the detected cluster for small groups of infected

[3] https://publicsuffix.org/

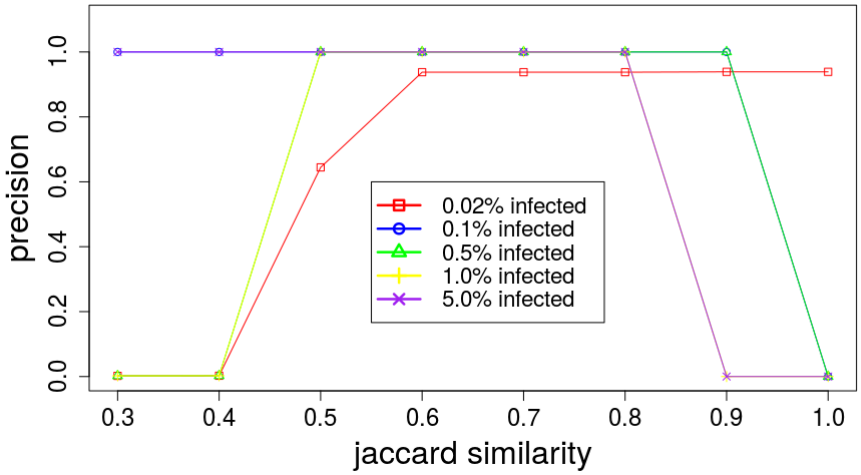

Figure 4.3: Precision for different thresholds for Jaccard similarity and the size of the botnet

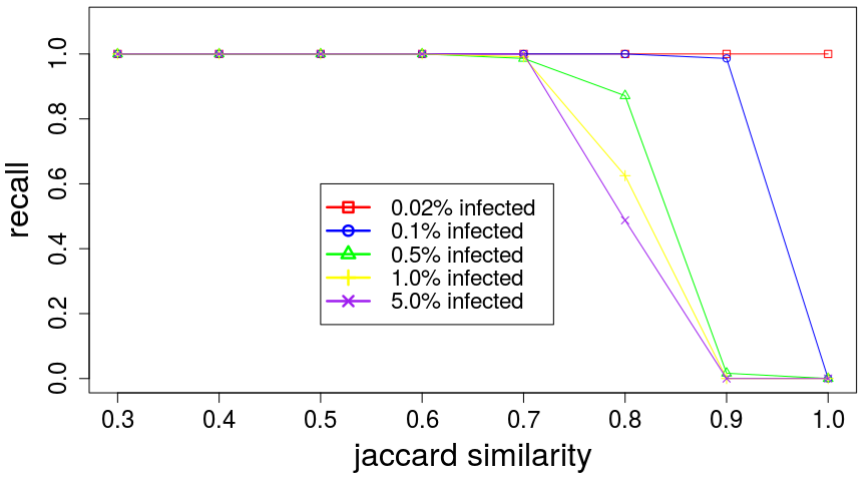

Figure 4.4: Recall for different thresholds for Jaccard similarity and the size of the botnet

Without further filtering, our algorithm detected 7041 additional clusters in the original dataset. Domains in these clusteres were manually inspected and it was attempted to find actual malicious domain names in these. Out of all sets, 18 seemed to be, based on the randomness of the names, suspicious sets. Domains in these sets were crossmatched with DGArchive, but unfortunately DGArchive could not point to the source of these domain names. These findings were also reported back to the research network's operator, but at this time the source is still unknown. Interestingly, all except for one set of suspicious names, was only queried by a single host on one or two days. The final cluster, consisting of 26 domain names, detected by the algorithm, turned out to be from an advertising network utilizing a few random looking domain names to distribute their ads. Also, for all detected clusters, the network traces of the detected hosts were manually inspected to make sure the algorithm found all relevant domain names (as requested by the affected hosts). This was always the case, although the algorithm always included approximately 30% benign domains as well.
From the 17 sets (18 excluding the advertising network), three different types of generated domains were detected, requested by fully disjunct parts of the network on different days, hence they were not clustered together as a single cluster. Samples of domains can be seen in table 4.1. The majority of detected domain names fall in the first category.
Since the goal of the algorithm is to detect botnets active in a network, one of the filtering parameters in the algorithm allows to reduce the output by only returning detected clusters with a minimum average requesters per domain name within a cluster. In the tests with injected traffic, even the hypothetical infection of 0.02% of the network with a DGA would make this average stand out significantly from other detected clusters that are actually benign. These clusters mainly contain domain names of niche websites contacted by a single host. After applying this filter, to an average of 2 unique requesters per domain name in a cluster, the output would be reduced to a manageable volume for a network admin to manually check (22 clusters in the tested traffic).

Box 1: Detections within the tested research network

Sample 1	Sample 2	Sample 3
32cxvjnid1g1i1edicij.hp16c42x90qcn2v69090.com	sajookkjxyjg.net	1459f4a279.space
j0rvirnxd-ldlbd87.9ov9b74di.com	bdqmnlzcpxtiu.net	1917f71a77.club
g2uwl2ryx5cev669j4zk9cn4.wlsz8qxjygqij.com	hdweyujkls.net	2f782a4fa1.pw
9j3m0grsfeqeritazfs.q4g6ah1780-50yw8c.com	nwcrbyncrxz.net	8426fa9a8.club
4t93ob6j28f-muvibjst.j66y6i0t8ch9ot8k.com	zinlzws.net	8426fa9a8.pw
d80lcagxvm-afsxf0s5p3jlw57c.mmh6a0dsmv8c.com	hgoptozw.net	8426fa9a8.space

Table 4.1: Samples of domain names detected by the proposed algorithm

hosts. With larger infected parts of the network, this has a smaller impact on the precision. In figure 4.2, the Jaccard similarities between all domains in the injected traffic can be seen. From this, it can also be seen, that in order to detect all domains belonging to the same injected set of domain names, a threshold for the Jaccard similarity around 60% to 70% is on the upper bound for full detection, as higher thresholds will not see all traffic as belonging to each other. Lower thresholds on the other hand, will cluster many unrelated domains together into the same cluster. The threshold of 70% has been chosen for all further applications, as using the highest possible threshold will reduce the amount of Jaccard similarities required to be calculated and compared in the clustering step, although the data suggest that a lower threshold up until 60% would yield the same results.

Applying our algorithm with a Jaccard similarity threshold of 70% on the gathered data, yields the results presented in box 1. As the results were too inconclusive, and a baseline of infection was not available, the few findings were reported to the Delft University of Technology network administrators, but so far no action was taken based on the reports. The following sections will compare several algorithms to this newly designed method on artificially constructed infections, allowing to calculate exact values for detection precision and recall.

4.4. Comparison with other detection techniques

As comparing the effectiveness of the newly proposed algorithm against the current state-of-the-art is near impossible due to the lack of a correct baseline truth, as explain in the previous section, other options have to be evaluated. One of those options is to inject clearly malicious network traces into the network topology retrieved from the actual network. This allows for an analysis of several algorithms over a dataset having approximately the same size as an actual production network.

In order to be able to compare the newly designed algorithm to other proposed techniques to detect malicious DGAs in DNS traffic, several detection mechanisms proposed in literature were implemented as well during the work on this thesis. Specifically Phoenix [33], Pleiades [3], BotGAD [8] and PsyBoG [17] were selected as they present recent work and are well-cited by their peers. As some other algorithms are very time and memory consuming to run, the comparison to other algorithms was limited to data of a single day. As these algorithms require more features of the DNS traffic than the algorithm designed in this thesis requires, all DNS traffic features were extracted from the raw network captures, most importantly, but not limited to, exact timings, fully qualified domain names (FQDNs) and the DNS answers are used.

Of this legitimate data set, 220 hosts were randomly selected to be infected with an instance of malware containing a DGA. The specific DGA would contact, in a locally random order, 200 domains. Of those 200 domains, at least 1 domain is queried by all infected hosts eventually and would result in a positive DNS answer, mimicking a registered and active domain used by a botmaster. To further mimic a feature commonly seen in DGAs, requests were made exactly every ten minutes. Also, hosts would have a starting time in the morning and an ending time in the afternoon or evening, in order to simulate normal computer use, where a user would start their machine in the morning, and shut it down when leaving at the end of the day. The request pattern and domain names were generated by a seeded random number generator and the results are averaged over ten runs, to prevent any bias caused by randomly created easy-to-solve instances. To compare results with current DGA domains and possible future DGA domain names, utilizing concatenations of English dictionary words, as defined in section 4.1, the DGA of the Locky ransomware[4] was implemented.

Pleiades is very time and memory intensive and makes only use of NXDomain DNS traffic. In fact, in the tests attempted to run on the dataset of a full day, it could not finish on a 24-core server with 200 GB of RAM for the dataset of 22,000 hosts, and in smaller sample sizes, it only managed to find clusters related to reverse DNS look ups. As it heavily relies on linguistics, it can be expected to fail to detect domains generated by a DGA specifically trying to evade this. In order to enable a comparison, also a smaller synthetic evaluation scenario was created, which will be discussed in section 4.5.

Phoenix works in three steps, of which the first one tries to determine whether or not a domain might be suspicious. After applying this step to all traffic from the selected day, Phoenix already filtered out more than 60% of the Locky and natural language domains, while still leaving 39% of all domains occurring in the dataset. As the detection of Phoenix is built against a baseline of actual domain names (the Alexa 100000 list), a drop in detection of about 40% can be seen when the used DGA transitions from fully randomized letters to randomly concatenated words.

In contrast to Pleiades and Phoenix, which heavily rely on linguistic features in their detection, BotGAD and PsyBoG mainly build on the synchronization and periodicity of requests. BotGAD also filters domains before attempting to cluster domains together. After this step, it only retained a single domain of our injected botnet traffic, but it also marked 9.74% of all domains as possibly suspicious. However, the injected domain it did detect, is the domain used as common domain over all participating “infected” hosts. After the first filtering step, it would further reduce the set of detected domains based on linguistic features of the domain names and features of the DNS query and answer itself. Furthermore it relies on short periodicities of DNS traffic over multiple hosts and it needs to be fine tuned to a time window, where a larger time window results in significantly less accuracy, and a smaller time window could miss suspicious domains (as a bot might not use DNS in a periodic manner). PsyBoG detects infected hosts by finding periodicities in the timing of DNS requests issued from any given host in the network. As the inserted traffic between the Locky domain names and natural language domain names are on the same times, the results do not differ. It does however detect 33.4% of all hosts as suspicious in terms of having a periodic DNS request pattern. Table 4.2 shows an overview of the exact detection rates of the four evaluated algorithms, for the case of Locky and a word-based DGA.

When injecting malicious traffic into the topology of an actual network it can be seen that the effective-

[4] https://dgarchive.caad.fkie.fraunhofer.de/site/families.html#locky_dga-details

			Linguistic-based		Periodicity-based	
	detected	own algorithm	Pleiades	Phoenix	BotGAD	PsyBoG (hosts)
Locky	DGA	200 (100%)	-	77 (38.50%)	1 (0.50%)	125 (56.82%)
	total	72282	-	271686	68965	7424
natural language	DGA	200 (100%)	-	46.5 (23.25%)	1 (0.50%)	125 (56.82%)
	total	72282	-	271655.5	68965	7424

Table 4.2: Detected locky domains and natural language domains on the compared day of traffic containing 707826 different domains. The detected domain by BotGAD is the "active" DGA domain, contacted by all infected hosts. Domains were generated with a seeded random number generator and results were averaged over ten runs. PsyBoG works by initially detecting suspicious hosts, instead of domains.

ness between the chosen algorithms differs highly. While Pleiades does not scale well enough it was unable to produce any meaningful output, rendering the algorithm ineffectual to be used outside the scope of test networks. Phoenix, only reasoning over the linguistic aspect of domain names, produces a large list of domains it deems to be malicious, while detecting less than 40% of the actual malicious domains. Assuming that the clustering step of Phoenix results in a perfect clustering, i.e. clustering these 40% together as a single infection with no unrelated domains, this might still be unnoticeable in between all other detected domains, purely by the large amount of false positives. A similar problem can be noticed for BotGAD, which does detect significantly less domains unrelated to the injected traffic, but only detects a single domain of the injected malicious set, resulting in a perfect clustering of only a single domain, which again would be unnoticeable in between the approximately 70,000 other domains. PsyBoG manages to detect a majority of the infected hosts, which would most likely stand out after being clustered together, especially as the amount of false positives is lower than the other algorithms. The newly proposed algorithm presented in this thesis detects roughly the same amount of false positives as BotGAD, but also detects all of the injected domains. After clustering, a cluster of 200 domains would naturally stand out in between approximately 70,000 other domains, when compared to the BotGAD results. Therefore it can be said that in this scenario only PsyBoG and the proposed algorithm yield in practice usable results.

4.5. Evaluation on synthetic data set with varying network properties

As determining a baseline truth is infeasible on the amount of data encountered in the test data set from the research network, the implemented botnet detection techniques are also tested on domains artificially created. For this, a network of varying size contacting fabricated domains of non-existent domain generation algorithms is fabricated. To mimic real life scenarios, domain names consisting of randomly selected characters, appended by large top level domains, are generated. Other variable parameters are the amount of domains contacted by the DGA per day, the periodicity of network requests and the standard deviation of the periodicity. In all experiments, the entire constructed network contacts DGA domains.

As none of the algorithms compared to the new detection algorithm makes assumptions over multiple days, these experiments are limited to single days. In these cases, the new algorithm would detect everything as suspicious and cluster all hosts in the entire network together as one suspicious cluster. In section 4.6 and section 4.7, experiments with different rates of infection in a network and experiments with simulated traffic over a span of multiple days are shown.

The baseline for our experiments uses 1500 hosts, contacting 1000 domains in a locally randomized order, with a new domain being contacted every 10 minutes. All hosts have randomized starting and ending periods. Per host a time between 00:00 and 12:00 is used as start time and time between 18:00 and 24:00 is chosen as end time. Also, all hosts contact at least 1 overlapping domain, which simulates an active command & control server. When a host has reached this domain, it will continue requesting it every 10 minutes until the end of the assigned time period. No host contacts any legitimate domain, in short, 100% of the network is infected and only contacts DGA domains, hence in the ideal situation a detection algorithm would flag all domains and hosts as belonging to a DGA. To prevent outliers from specific random seeds used to set up the network and the domain names, all experiments were executed ten times with different seeds, and the averages are shown. In the following sections, specific variations of this artificial network are discussed and the averaged detection accuracies and required runtime are compared.

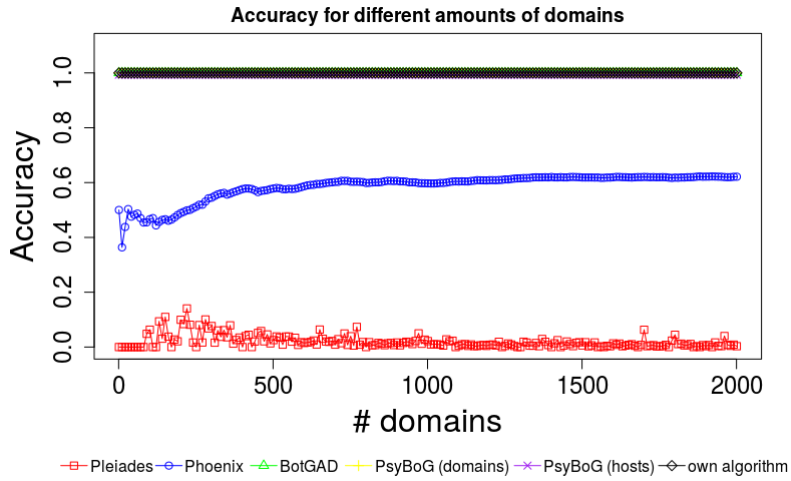

Figure 4.5: Detection accuracy of Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm with varying amounts of domains contacted per day by a DGA. **BotGAD, PsyBoG** (both domains and hosts) **and the algorithm proposed in this thesis fully overlap**.

4.5.1. Accuracy

This section consists of the construction of many artificial networks as described in the previous section, with one parameter changed per experiment. This allows for a comparison of what influence specific types of networks have on the accuracy of all algorithms described in this thesis. The specific parameters changed, in order, are the amount of domains used by the DGA, the amount of hosts active in the artificial network, the periodicity of DNS traffic used by hosts in the network and the standard deviation of the periodicity used by infected hosts.

In figure 4.5 the influence of a different amount of domains is shown. In practice, between 1 and 2001 domains are used. It can be seen, that PsyBoG and BotGAD are not influenced by the amount of domains used by a DGA, which can be expected, as they mainly use periodicities in DNS traffic in order to either detect domains or hosts. Both detect 100% of the domains as malicious and PsyBoG detects 99.3% of the hosts. However, for Phoenix it seems that the more domains a DGA utilizes on a daily basis, the higher the detection accuracy becomes, stabilizing at approximately 62% with more than 500 domains. Pleiades seems to have trouble detecting anything meaningful at all, with the majority of detection rates between 0.3% – 4.0%.

Figure 4.6 shows the variation of the amount of hosts in the network between 100 and 2100. The same pattern as for different amounts of domains can be seen for PsyBoG, BotGAD and Pleiades. Phoenix has a detection accuracy of 59.7% for all instances, while Pleiades performs extremely poorly. As with a fluctuation in the amount of domains used, this can be expected, as the design of the used domain names are not changed, thus not affecting the results of linguistic based DGA detection mechanisms, nor the periodicities are changed, which would impact periodicity based approaches.

As PsyBoG heavily relies on periodicities of DNS network activity, one of the variable parameters of the constructed network is the used periodicity. In figure 4.7, periodicity values between 300 and 900 seconds are compared. Again, the detection accuracy for all four algorithms is comparable to the previous experiments. This is unsurprising, as Pleiades and Phoenix do not take periodicity into account at all, and the other tested algorithms are resistant to deviations to periodicity on this small scale. For very large periodicities, the tested algorithms might fail to detect malicious traffic, but these periodicities, of longer than an hour, would be highly ineffective for normal botnet operations, as it would take a long time for infected hosts to find their central authority.

Finally, networks with varying periodicities are constructed and the detection results are compared in figure 4.8. Starting out with 600 seconds as periodicity for network activity, the periodicity is adjusted to follow a normal distribution with varying standard deviations. Compared to the previous experiments, this does not influence BotGAD, Pleiades and Phoenix. It does however have a significant effect on PsyBoG, which ceases to detect botnets when the standard deviation reaches values is larger than $100s$. Before that, it already fails to detect all hosts reliably. In practice, this means that a botmaster can easily avoid detection by randomizing the timing between requests of individual hosts.

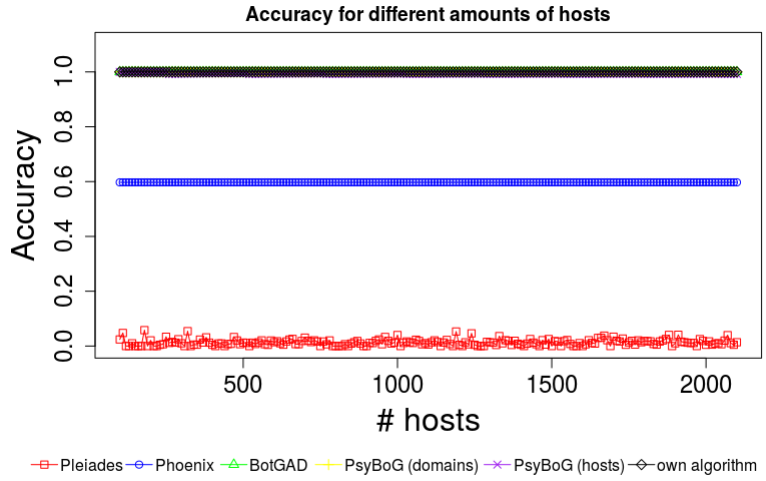

Figure 4.6: Detection accuracy of Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm with varying amounts of hosts active in the network. **BotGAD, PsyBoG and the proposed algorithm almost fully overlap**; PsyBoG is decreasing slightly with more hosts.

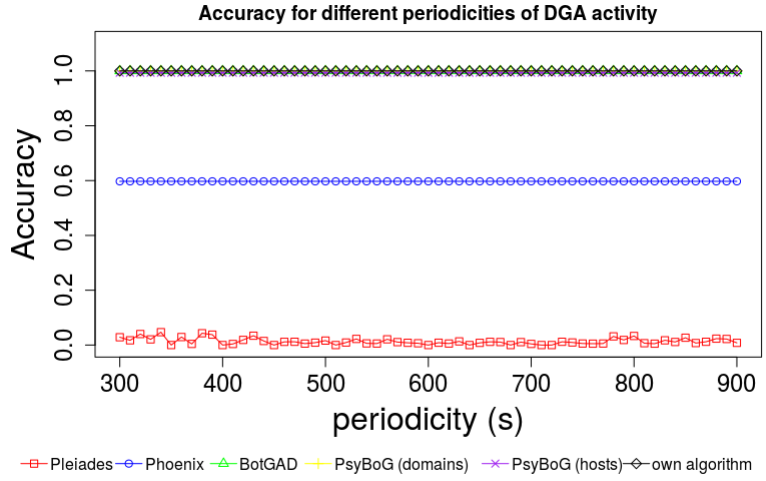

Figure 4.7: Detection accuracy of Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm with varying periodicities of DNS activity in the DGA. **BotGAD, PsyBoG and the proposed algorithm fully overlap**.

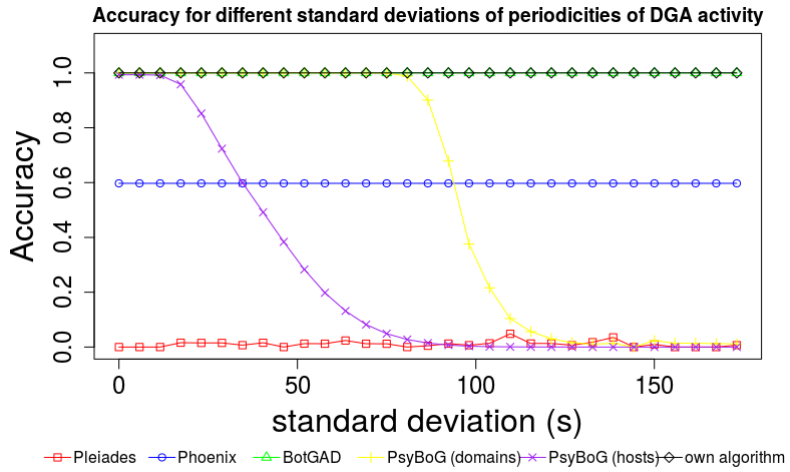

Figure 4.8: Detection accuracy of Pleiades, Phoenix, BotGAD, PsyBoG and our algorithm with varying standard deviations of the periodicities of DNS activity in the DGA. **BotGAD and the proposed algorithm fully overlap**.

In conclusion, after comparing differing amounts of domains, amounts of hosts, DNS timing periodicities and standard deviations of DNS timing periodicities, it can be seen that Pleiades never manages to detect any significant amount of domains used by a domain generation algorithm at all, while Phoenix has a steady 62% detection rate and tends to perform better when a DGA uses more different domains on a single day. BotGAD and the newly proposed algorithm perform perfectly in all of these instances and are not influenced by any parameter used to construct these artificial networks. PsyBoG is also almost perfect, with the difference that it fails to detect infected hosts when hosts no longer conform to fixed periodicities when trying to contact a DGA domain.

4.5.2. Runtime

In order to be able to determine whether or not an algorithm is feasible in practice, having an indication of the runtime required to execute it is helpful. For the same experiments as shown in section 4.5.1, the runtime required to calculate the results is presented. All sets of tests were run in sequence on the same hardware, which eliminates the effect of any unfair scheduling in the CPU. In figure 4.9, it can be seen that Pleiades takes more computational time when confronted with more different domains over the measured timespan. The other algorithms are not influenced by different amounts of domains. The reason for this is mainly due to Pleiades creating one large matrix with all requesting hosts and the corresponding DNS requests, and subsequently calculating the eigenvectors of this matrix. This calculation is at least $o\left(n^2\right)$ [27]. The other implemented algorithms rely more on either linear time operations or do not use information about all requests at once. The algorithm designed in this thesis is heavily influenced by the amount of domains used, as shown in section 3.3.5, but as it does not use information about all requests and only uses a small subset of the original data as input, it still requires less time than Pleiades.

Figure 4.10 shows that all algorithms, except for Phoenix require more time when the input includes more different hosts. As with the difference of timing with a change in the amount of domains used, these changes in the runtime can be expected.

When the periodicity is changed, shown in figure 4.11, it is evident how BotGAD requires less time, the higher the periodicity is. This can be attributed to the way BotGAD filters requests that do not follow a common pattern over time. As requests are stored in buckets of multiple minutes, choosing periodicities larger than the bucket size will result in many requests being filtered prior to determining the classification of a domain name.

In figure 4.12, the influence of the standard deviation of the DNS network activity periodicity is shown. No noticeable differences in the required runtime for any algorithm can be noticed. When confronted with more hosts, having a significant periodicity will reduce the required runtime for PsyBoG, compared to that same amount of hosts without a significant periodicity. This happens as it tries to detect a periodicity in small time windows, and if the first window would already contain a periodicity, the host would immediately be flagged

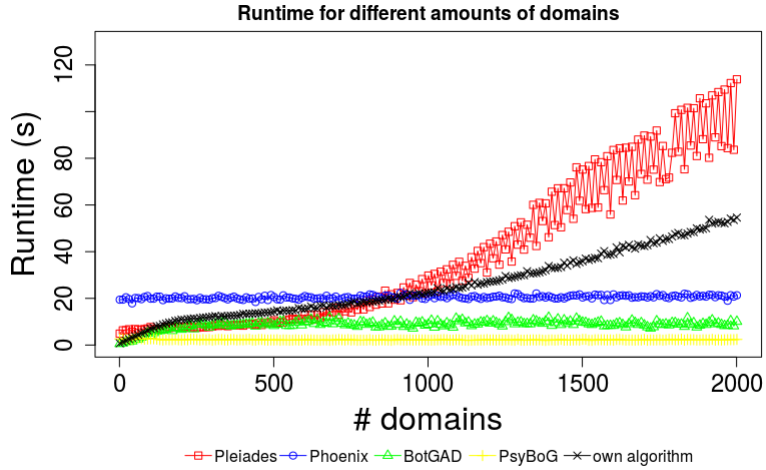

Figure 4.9: Runtime of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm with varying amounts of domains contacted per day by a DGA.

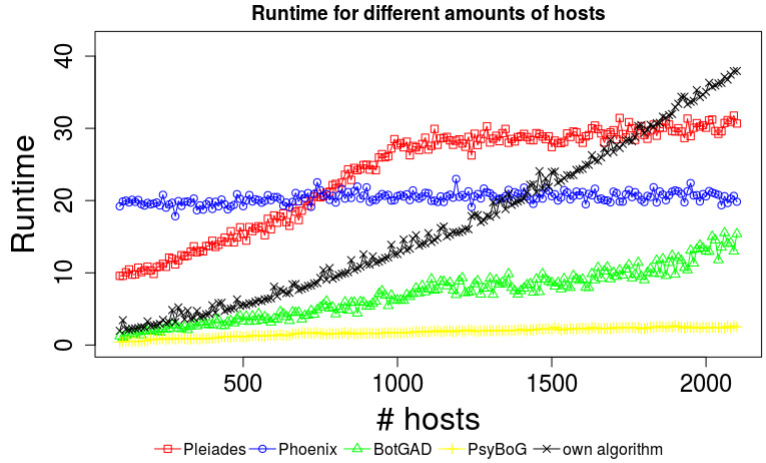

Figure 4.10: Runtime of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm with varying amounts of hosts active in the network.

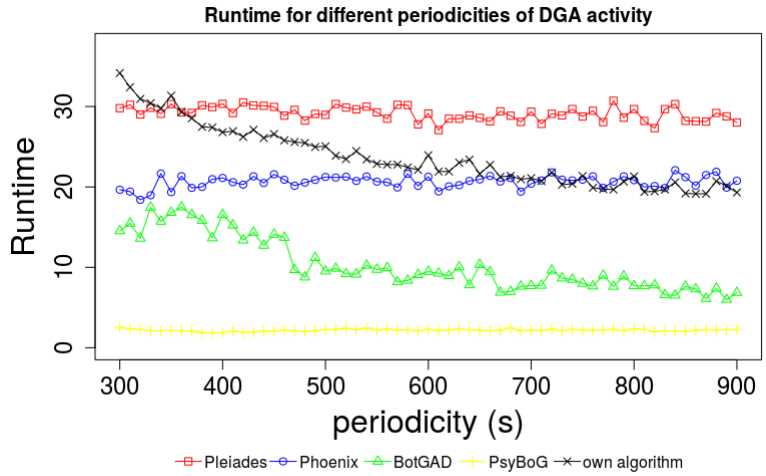

Figure 4.11: Runtime of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm with varying periodicities of DNS activity in the DGA.

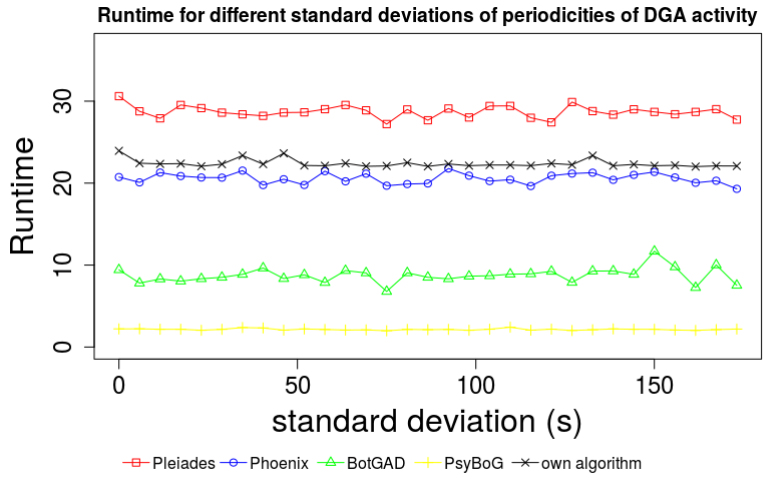

Figure 4.12: Runtime of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm with varying standard deviations of the periodicities of DNS activity in the DGA.

as suspicious and the algorithm could return early. But for only 1500 hosts, no such pattern can be seen.

Summarizing, it can be seen how PsyBoG is clearly the fastest algorithm, independent of the network setup. The only influence on its runtime is linearly correlated to the amount of hosts, which can be expected as PsyBoG tries to determine a common frequency in DNS traffic with the help of fast Fourier transformations on a per-host basis. BotGAD essentially creates a large matrix per domain marking all hosts accessing that domain on any given time window. After that, outliers are filtered out of these matrices. This explains, how BotGAD requires more time when it encounters more hosts related to a DGA and less time when the periodicity increases, as thresholds in the algorithm will see more columns (indicating time) as outliers and filters them out before applying a time-costly algorithm to determine whether or not a domain belongs to a DGA. A negative side effect in terms of accuracy can be seen in the section 4.4, where BotGAD is not capable of detecting large parts of domains used by a DGA. Pleiades is generally slow, as it calculates many Eigenvectors of matrices constructed with linguistic features of domains. That also explains the increased runtime requirement for more unique domains contacted within a network. The increase in required runtime when more hosts are active in the network can be attributed to a filtering operation within the algorithm actively filtering out domains with only a few unique requesting hosts. As eventually most domains in the constructed network have more requests than this threshold, the runtime ceases to increase. Phoenix only applies a statistical model to linguistic features of domain names. Therefore it scales linearly with regards to the amount of unique domains encountered in a network trace. As loading the statistical model into memory takes a relative long time, approximately 20 seconds, and testing a domain against this model is very fast, the plots do not indicate this linear correlation. Finally, the proposed algorithm from this thesis scales poorly, as described in section 3.3.5 and is heavily influenced by the amount of unique domains and hosts in the network. This theoretical problem can clearly be seen in the plots representing the runtime for different amounts of domains (figure 4.9) and hosts (figure 4.10).

4.6. Evaluation on synthetic data set with different infection rates

The previous test cases only differentiate over different sizes and periodicities of networks, but always assume a fully infected network. As this is a highly unlikely scenario, the effectiveness of the implemented algorithms for different rates of infection is also examined. To construct test cases to be used for this evaluation, a randomly generated network of 1000 hosts, of which a variable percentage of hosts is infected with a DGA performing DNS requests to domains generated by the Locky DGA, is created. Infected hosts would issue requests to 200 generated domains per day in a locally random order exactly every 10 minutes. All hosts would, not periodically, issue requests to legitimate domains found in the Alexa top list. As done in section 4.5, all hosts have a randomly assigned start and end time of activity on each day to mimic hosts being turned on in the morning and being shut down later that day. Because the network also contains emulated legitimate

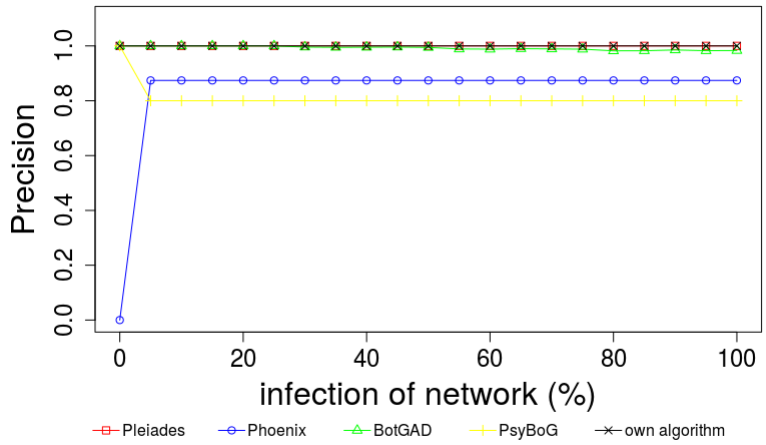

Figure 4.13: Precision of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm in a network.

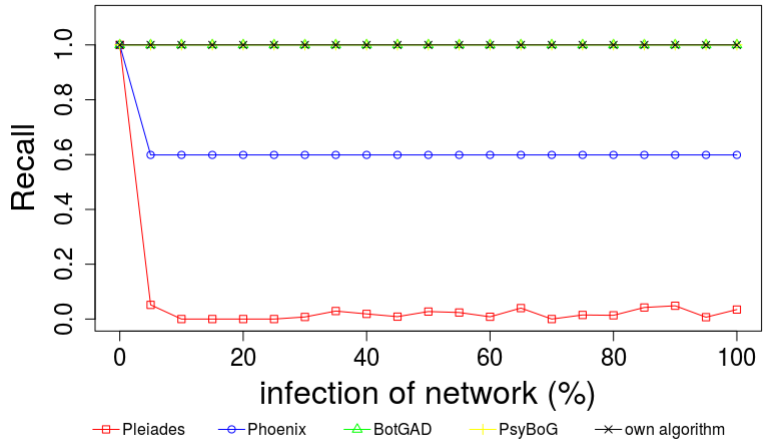

Figure 4.14: Recall of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm in a network.

traffic, all algorithms can be compared by their precision and recall.

As in the previous instances of synthetic data sets, the results are averaged over ten runs. The precision and recall for the instances with varying rate of infection can be seen in figure 4.13 and figure 4.14. BotGAD and the algorithm designed in this thesis score almost perfectly, except for a slight drop in precision for BotGAD when the infection rate reaches a high value, which is very unlikely in a diverse work network. Pleiades generally performs very bad; while it never detects a non-malicious domain as malicious, it hardly detects any actual malicious domains either. The recall of 100% with Pleiades for the 0% infection rate test case stems from Pleiades not marking any domain at all, which is the correct outcome.

Figure 4.15 and figure 4.16 show the runtime and memory consumption metrics for all algorithms while executing the test cases.Phoenix and PsyBoG are not influenced by the different amounts of infection rates. Other algorithms do use linearly more runtime, while the new algorithm uses non-linearly more runtime. The latter can be explained by the use of the Jaccard similarity metric to determine to which DGA cluster domains should be assigned. As the infection rate rises, the sets of IP addresses increase in size and calculating the intersection between these sets takes exponentially more time. This also explains the increase of memory consumption for the algorithm designed in this thesis, as during the entire calculation of clusters of domains, these large sets of IP addresses are stored in memory.

In the current implementation, the Jaccard similarity of these IP address sets with all other of such sets are compared. If there are n suspicious domains, this results in $\frac{n \cdot (n-1)}{2}$ calculations for the Jaccard similarity. This

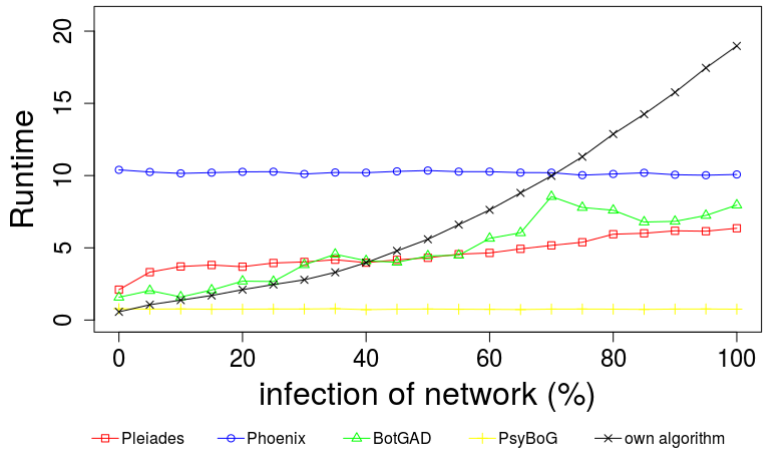

Figure 4.15: Runtime of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm in a network.

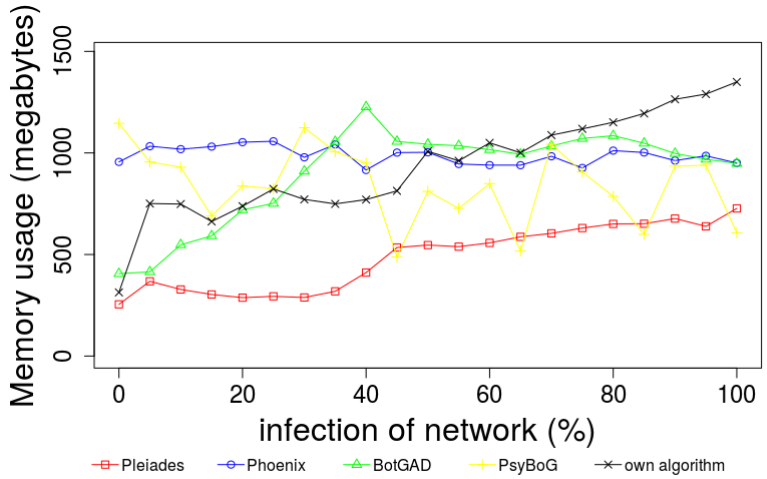

Figure 4.16: Memory usage of Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm in a network.

can be significantly reduced by only comparing these similarities for domains active around the same time, for example the same three days used for the detection of suspicious domains as described in section 3.3.2.

In conclusion, Pleiades performs poorly in all instances, while quickly requiring absurd amounts of time and memory to finish, making the algorithm unusable in an actual network of the size of the Delft University of Technology. Phoenix does not suffer from these time and memory requirement issues and achieves a high precision, but unfortunately the recall is not very high, resulting in many false positives being detected. Both BotGAD and PsyBoG perform extremely well, both in precision and recall, and the time and memory requirements. The proposed algorithm, while having perfect results with regards to precision and recall in all instances, does not scale as well as the other algorithms.

4.7. Evaluation on synthetic data set over multiple days

So far, all experiments comparing different algorithms were executed based on data of a single day. As the algorithm described in this thesis specifically uses information from DNS traffic over a longer timespan, detection in such a test case also needs to be tested. Given the algorithmic complexity of the Pleiades approach, the constructed network used for this is siginificantly smaller than the data set retrieved from the research network. A network of 1,000 hosts, of which 200 are infected with a botnet performing periodic DNS requests

		Linguistic-based		Periodicity-based	
	own algorithm	Pleiades	Phoenix	BotGAD	PsyBoG
Locky - precision	1.00	1.00	0.86	1.00	0.80
Locky - recall	1.00	0.02	0.51	1.00	1.00
natural language - precision	1.00	0.00	0.73	1.00	0.80
natural language - recall	1.00	0.00	0.23	1.00	1.00

Table 4.3: Precision and recall for DGA detection in a randomly constructed network for Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm. Domains were generated with a seeded random number generator and results were averaged over ten runs.

every 10 minutes to DGA domains (constructed as done by the Locky ransomware and domains generated by concatenating dictionary words), was constructed. This scenario results in an Eigenvalue decomposition that is still computationally feasible, and is therefore testable with Pleiades. For four days in a row, all hosts had a predetermined start and shutdown time, mimicking hosts being turned on in the morning and being shut down in the afternoon and evening. Additionally, all 1,000 hosts would randomly contact domains of the Alexa top 200. The precision and recall for Pleiades, Phoenix, BotGAD, PsyBoG and the new algorithm on this constructed network is shown in table 4.3. As can be seen, the algorithms most heavily relying on linguistics of domain names, Pleiades and Phoenix, do not perform well with the DGA domains consisting of dictionary words. As expected, request-based algorithms such as BotGAD, PsyBoG and the new algorithm do not see any change in this synthetic network.

5

Conclusion

In this chapter, the work of this thesis will be summarized by answering the research questions proposed in chapter 1.

RQ1: How does the current state-of-the-art of DGA detection work, and how can a DGA circumvent detection?

The state-of-the-art of DGA detection specifically stems from detection of botnets utilizing domain generation algorithms. Current detection methods either focus on detecting domain names constructed by randomly concatenating valid characters with a top level domain appended or by examining periodicities in DNS traffic on a per-host basis for short times. As current DGAs abide by these patterns, current techniques work very well and can accurately detect and classify botnet infections in networks. Specifically because DGAs are currently detected by showing anomalies in the construction of their domain names, consisting of fully randomized domain names, or by having very predictable periodicities used for their communication, e.g. showing traffic to their command and control server exactly every ten minutes, the assumption is made that botmasters will change their operations to utilize domains which are not fully randomized on a per character basis. Also randomizing the periodicity of traffic is an option botmasters have at their disposal. This thesis shows, that creating a DGA utilizing concatenated English dictionary words is possible and yields a botmaster many domains to work with. Randomizing the timing of hosts requesting domain names is also easily achievable by botmasters. Applying these two techniques defeats most of the current detection methodologies in place.

RQ2: How can a new DGA detection system be designed, that does not rely on linguistics or short term periodicities?

The approach of this thesis, for detecting DGAs in a network relies on detecting sudden increases, starting with no traffic, and decreases, going down to no traffic, in popularities of single domain names. This only makes use of the assumption that domains used by a DGA will only ever be used by for a short period of time. In practice most DGAs only use a set of domain names for a single day, or even less, but in practice issues with local clocks on infected hosts show that sometimes traffic towards a DGA domain already starts earlier and ends later than intended. Therefore, thresholds for this detection are set to allow for three days of traffic, with no traffic before or after. Legitimate domains, even from small, personal websites, will, over time, be accessed outside this three day scope and will be whitelisted by the detection system. If this assumption is correct, and dissection of several DGAs suggests it is, the system proposed in this thesis will accurately flag all domains used by any active DGA in the network. A problem experienced during the evaluation of the detection system on the real data set did result in benign domains also being flagged, as they were only accessed on a single day in the provided dataset. These benign domains included anything from personal websites to foreign advertising networks. The assumption is, in a network of the size the algorithm was tested on, over time most of these benign domains will be accessed another time and will then effectively be whitelisted. As only two weeks of real data

were used, there were still some benign domains left and the algorithm clustered approximately 30% of benign domains into clusters containing malicious domains.

A drawback of the approach of counting days of any activity on the network for any domain is, that it takes three days until any new DGA can be detected in the system. This threshold could be reduced to a shorter time period, but as DGAs sometimes suffer from desynchronization, doing so would possibly lower the detection accuracy. Thomas et al. [42] (see figure 2.2) noticed this unintended desynchronization in practice for the Conficker malware. Compared to currently used detection mechanism, this is a downgrade as they allow for detection within at most two hours, as they either don't use information about the timing of DNS requests, or only look at DNS traffic on much shorter time windows.

RQ3: How can hosts, infected with the same software utilizing a DGA, be clustered together with this new system?

In order to cluster together domains marked as malicious by the algorithm designed in this thesis, sets of hosts related to traffic to any suspicious domain are composed, as described in section 3.3.3. For these sets, the Jaccard similarity with all other sets is calculated. When this similarity is above a fixed threshold, the algorithm assumes two domains to belong to the same DGA. From tests, a threshold of 70% results in a optimal state where domains from the same DGA are still detected as a single cluster with only a small amount of unrelated domains included. A higher threshold resulted in less unrelated domains being detected, but the DGA domains would be split over multiple clusters. A lower value would result in multiple DGAs being clustered together as a single DGA and more benign domains would be included.

In conclusion, this thesis presents a novel approach in order to detect DGAs commonly used by botnets. As current detection methods rely on detecting used domain names by linguistic features or by detecting suspicious patterns over short time intervals, a proof of concept DGA was designed attempting to circumvent these detection methods. After evaluating and comparing the new method presented in this thesis, to approaches proposed by literature, it can be seen how the new methodology is more resilient to variations in DGA behavior than previous work. One compared algorithm, BotGAD [8], was able to reliably detect at least one used DGA domain in varying test cases. In practice, only detecting a single malicious domain per DGA is still hard to find for network administrators, as it will disappear in the noise of false positives. The final algorithm used in comparisons, PsyBoG [17], does not detect clusters of domains used for DGAs but instead detects clusters of infected hosts. While achieving a high precision, it would not detect all infected hosts in the performed experiments. The new approach did correctly detect and cluster DGA domains together in a single group, which would stand out from this type of noise. Other algorithms would either be to time and/or memory intensive for use in practice, or they would be defeatable by using a DGA specifically designed to evade detection based linguistics or short term periodicities. The cost of this detection accuracy is an increase in required time and memory to execute the new algorithm. In practice, the experiments on the full data set of the research network still executed faster than real time on a cheap desktop computer, making it still usable in practice.

6

Discussion and future work

In the previous chapters, a novel approach towards detecting hosts contacting DGA domains in a network is presented. There is, however, still room for improvement in this new design. For one, unique hosts within the network are identified by their IP address. In practice, the tested network uses the Dynamic Host Configuration Protocol (DHCP) to provide IP addresses to clients in the network. By its configuration, physical devices can regularly receive new logical addresses, which the proposed algorithm would then interpret as a different host. This is detrimental to the clustering of domains, as the algorithm can possibly not find a link between the old and new addresses. This problem can be solved, when instead of the IP address, the physical media access control (MAC) address can be used for host identification. In the data set available while working on this thesis, this information was not correctly stored in the used packet captures. In a network setup in this way, it can still be possible to retrieve this physical address by looking up the detected IP address in the DHCP lease table, maintaining all associations of physical to logical addresses for that time.

Another problem is introduced by the time complexity of calculating the Jaccard similarities between all sets of hosts. In the current implementation, all these similarities for all suspicious domains detected for the entirety of the input data are calculated. In practice, the amount of calculations can be reduced to only determine the similarity between domains accessed on the three day time window also used to detect whether or not a domain is suspicious, allowing again for desynchronization issues of misbehaving DGAs.

As the detection algorithm on its own can be reduced to extracting features from discrete signals from sensors (DNS traffic from hosts) and finding correlations in these signals (graph traversal over a mesh of hosts and domain names), this algorithm is not only applicable to DGA detection and might be used to find other correlations between outliers in large amounts of data.

Bibliography

[1] Jasper Abbink and Christian Doerr. Popularity-based Detection of Domain Generation Algorithms. In *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*, pages 1–8, New York, New York, USA, 2017. ACM Press. ISBN 9781450352574. doi: 10.1145/3098954.3107008. URL http://dl.acm.org/citation.cfm?doid=3098954.3107008.

[2] 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 on Internet measurement - IMC '06*, page 41, New York, New York, USA, 2006. ACM Press. ISBN 1595935614. doi: 10.1145/1177080.1177086. URL http://portal.acm.org/citation.cfm?doid=1177080.1177086.

[3] Manos Antonakakis and Roberto Perdisci. From throw-away traffic to bots: detecting the rise of DGA-based malware. In *Proceedings of the 21st USENIX Security Symposium*, page 16, 2012. ISBN 978-931971-95-9. URL https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final127.pdf.

[4] Basil AsSadhan, José M F Moura, David Lapsley, Christine Jones, and W. Timothy Strayer. Detecting botnets using command and control traffic. In *Proceedings - 2009 8th IEEE International Symposium on Network Computing and Applications, NCA 2009*, pages 156–162. IEEE, jul 2009. ISBN 9780769536989. doi: 10.1109/NCA.2009.56. URL http://ieeexplore.ieee.org/document/5190367/.

[5] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection, 2006. URL https://www.usenix.org/legacy/event/sruti06/tech/full{_}papers{_}old/binkley/binkley{_}html/srutibot.html.

[6] Jim Bound and Yakov Rekhter. Dynamic Updates in the Domain Name System (DNS UPDATE). URL https://tools.ietf.org/html/rfc2136.

[7] Ping Chen, Lieven Desmet, Christophe Huygens, Audrey Dorofee Christopher Alberts, Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, Wenke Lee, Keith Jarvis, Jason Milletary, Ari. Ting-Fang Yen Juels, Chris Poulin, V. Sekar, Y. Xie, M.K. Reiter, H. Zhang, Aditya K. Sood, Richard J. Enbody, Stuart Staniford-Chen, Steven Cheung, Richard Crawford, Mark Dilger, Jeremy Frank, James Hoagland, Karl Levitt, Christopher Wee, Raymond Yip, Dan Zerkle, D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, T. Bowen, Rohan Mahesh Amin, Julie J C H Ryan, Johan Rene Van Dorp, Charles Smutz, Angelos Stavrou, Nikos Virvilis, Dimitris Gritzalis, Parth Bhatt, Edgar Toshiro Yano, and Per Gustavsson. BotHunter: detecting malware infection through IDS-driven dialog correlation. *IEEE Security and Privacy*, 10(3):12, 2014. ISSN 15407993. doi: 10.1.1.135.8028. URL http://dl.acm.org/citation.cfm?doid=2420950.2420987{%}5Cnhttp://csrc.nist.gov/nissc/1996/papers/NISSC96/paper065/GRIDS.PDF{%}5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.7882{&}rep=rep1{&}type=pdf{%}5Cnhttps://securityintelligence.com/targ.

[8] Hyunsang Choi and Heejo Lee. Identifying botnets by capturing group activities in DNS traffic. *Computer Networks*, 56(1):20–33, 2012. ISSN 13891286. doi: 10.1016/j.comnet.2011.07.018. URL http://dx.doi.org/10.1016/j.comnet.2011.07.018.

[9] Graham Cluley. 17,000 Macs hit by malware botnet, with help from Reddit, 2014. URL https://www.grahamcluley.com/mac-malware-botnet-reddit/.

[10] John (JD) Douceur. The Sybil Attack, jan 2002. URL https://www.microsoft.com/en-us/research/publication/the-sybil-attack/?from=http{%}3A{%}2F{%}2Fresearch.microsoft.com{%}2Fpubs{%}2F74220{%}2Fiptps2002.pdf.

[11] FireEye Inc. APT29: HAMMERTOSS - Stealthy Tactics Define a Russian Cyber Threat Group, 2014. URL https://www2.fireeye.com/APT29-HAMMERTOSS-WEB-2015-RPT.html.

[12] Jan Goebel and Thorsten Holz. Rishi: identify bot contaminated hosts by IRC nickname evaluation, 2007.

[13] Siobhan Gorman. Annual U.S. Cybercrime Costs Estimated at $100 Billion - WSJ, 2013. URL http://www.wsj.com/articles/SB10001424127887324328904578621880966242990.

[14] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *SS'08: Proceedings of the 17th conference on Security symposium*, volume 5, pages 139–154. USENIX Association, 2008. ISBN 978-1-931971-60-7. doi: 10.1.1.144.5167. URL http://static.usenix.org/events/sec08/tech/full{_}papers/gu/gu{_}html/https://www.researchgate.net/publication/221260587{_}BotMiner{_}Clustering{_}Analysis{_}of{_}Network{_}Traffic{_}for{_}Protocol-{_}and

[15] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix Freiling. Detection and Mitigation of Fast-Flux Service Networks. *Ndss*, 2008. URL http://www.isoc.org/isoc/conferences/ndss/08/papers/16{_}measuring{_}and{_}detecting.pdf.

[16] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. pages 7–7, 2007.

[17] Jonghoon Kwon, Jehyun Lee, Heejo Lee, and Adrian Perrig. PsyBoG: A scalable botnet detection method for large-scale DNS traffic. *Computer Networks*, 97:48–73, 2016. ISSN 13891286. doi: 10.1016/j.comnet.2015.12.008.

[18] Wen-hwa Liao and Chia-Ching Chang. Peer to Peer Botnet Detection Using Data Mining Scheme. In *2010 International Conference on Internet Technology and Applications*, pages 1–4. IEEE, aug 2010. ISBN 978-1-4244-5142-5. doi: 10.1109/ITAPP.2010.5566407. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5566407.

[19] Carl Livadas, Robert Walsh, David Lapsley, and W. Strayer. Usilng Machine Learning Technliques to Identify Botnet Traffic. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 967–974. IEEE, nov 2006. ISBN 1-4244-0418-5. doi: 10.1109/LCN.2006.322210. URL http://ieeexplore.ieee.org/document/4116687/.

[20] Mahalanobis. On the generalised distance in statistics. 2(1), 1936.

[21] Steve Mansfield-Devine. DDoS goes mainstream: how headline-grabbing attacks could make this threat an organisation's biggest nightmare. *Network Security*, 2016(11):7–13, 2016. ISSN 13534858. doi: 10.1016/S1353-4858(16)30104-0.

[22] Mohammad M Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Peer to peer botnet detection for cyber-security: a data mining approach. In *CSIIRW '08: Proceedings of the 4th annual workshop on Cyber security and information intelligence research*, pages 1–2, New York, New York, USA, 2008. ACM Press. ISBN 978-1-60558-098-2. doi: http://doi.acm.org/10.1145/1413140.1413185. URL http://portal.acm.org/citation.cfm?doid=1413140.1413185.

[23] Yin Minn, Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. IoTPOT : Analysing the Rise of IoT Compromises. *Woot*, 2015. ISSN 09226389. doi: 10.1002/2014GB005021.

[24] Abedelaziz Mohaisen and Omar Alrawi. Unveiling Zeus Automated Classification of Malware Samples. URL http://delivery.acm.org/10.1145/2490000/2488056/p829-mohaisen.pdf?ip=145.94.198.185{&}id=2488056{&}acc=ACTIVESERVICE{&}key=0C390721DC3021FF.512956D6C5F075DE.4D4702B0C3E38B35.4D4702B0C3E38B35{&}CFID=953721188{&}CFTOKEN=30868296{&}{_}{_}acm{_}{_}=1498653416{_}4e8777131c107c012d.

[25] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2009. URL www.bitcoin.org.

[26] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan. PeerShark: Detecting Peer-to-Peer Botnets by Tracking Conversations. In *2014 IEEE Security and Privacy Workshops*, pages 108–115. IEEE, may 2014. ISBN 978-1-4799-5103-1. doi: 10.1109/SPW.2014.25. URL http://ieeexplore.ieee.org/document/6957293/.

[27] Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing - STOC '99*, pages 507–516, New York, New York, USA, 1999. ACM Press. ISBN 1581130678. doi: 10.1145/301250.301389. URL http://portal.acm.org/citation.cfm?doid=301250.301389.

[28] D. Pelleg, D. Pelleg, A.W. Moore, and A.W. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning table of contents*, pages 727–734, San Francisco, 2000. Morgan Kaufmann. ISBN 1558607072. doi: 10.1007/3-540-44491-2_3. URL http://portal.acm.org/citation.cfm?id=657808.

[29] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. Technical report, 2009. URL https://www.usenix.org/legacy/event/leet09/tech/full{_}papers/porras/porras{_}html/leet{_}paper.html.

[30] Niels Provos, Dean Mcnamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The Ghost In The Browser Analysis of Web-based Malware. URL https://www.usenix.org/legacy/events/hotbots07/tech/full{_}papers/provos/provos.pdf?{%}3Fid=3695.

[31] Douglas Roffel and Christopher Garret. A Novel Approach for Computer Worm Control Using Decentralized Data Structures | Blockchain (Database) | Bitcoin, 2014. URL https://www.scribd.com/document/250009335/A-Novel-Approach-for-Computer-Worm-Control-Using-Decentralized-Data-Structures.

[32] Somitra Sanadhya and Palash Sarkar. Information Security and Privacy. In *Acisp*, volume 4586, pages 274–292, Berlin, Heidelberg, 2008. Springer, Springer Berlin Heidelberg. ISBN 9783540699712. doi: 10.1007/978-3-540-70500-0. URL http://www.springerlink.com/content/hu2578414x215472.

[33] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: DGA-based botnet tracking and intelligence. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8550 LNCS, pages 192–211. Springer, 2014. ISBN 9783319085081. doi: 10.1007/978-3-319-08509-8_11.

[34] Kamaldeep Singh, Sharath Chandra Guntuku, Abhishek Thakur, and Chittaranjan Hota. Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests. *Information Sciences*, 278:488–497, sep 2014. ISSN 00200255. doi: 10.1016/j.ins.2014.03.066. URL http://linkinghub.elsevier.com/retrieve/pii/S0020025514003570.

[35] Ed Skoudis and Lenny Zeltser. *Malware, Fighting Malicious Code*, volume 18. 2003. ISBN 0131014056. doi: 10.1109/MNET.2004.1301015. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1301015.

[36] Sam Smith. Cybercrime will Cost Businesses Over $2 Trillion by 2019 - Juniper Research, 2015. URL https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillionhttp://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion.

[37] Matija Stevanovic and Jens Myrup Pedersen. An efficient flow-based botnet detection using supervised machine learning. *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 797–801, 2014. doi: 10.1109/ICCNC.2014.6785439. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6785439.

[38] Petre Stoica and Randolph L Moses. *Introduction to spectral analysis*, volume 1. Prentice hall Upper Saddle River, 1997.

[39] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet. In *Proceedings of the 16th ACM conference on Computer and communications security CCS 09*, volume 97, page 635. ACM, 2009. ISBN 9781605588940. doi: 10.1145/1653662.1653738. URL http://portal.acm.org/citation.cfm?doid=1653662.1653738.

[40] W. Timothy Strayer, Robert Walsh, Carl Livadas, and David Lapsley. Detecting botnets with tight command and control. In *Proceedings - Conference on Local Computer Networks, LCN*, pages 195–202. Springer, 2006. ISBN 1424404185. doi: 10.1109/LCN.2006.322100.

[41] Peter Szor. *The Art of Computer Virus Research and Defense*, volume 43. Addison Wesley, 2005. ISBN 978-0-387-30236-2. doi: 10.5860/CHOICE.43-1613. URL http://vxheaven.org/lib/pdf/TheArtofComputerVirusResearchandDefense.pdf.

[42] Matthew Thomas and Aziz Mohaisen. Kindred domains. In *Proceedings of the 23rd International Conference on World Wide Web - WWW '14 Companion*, WWW '14 Companion, pages 707–712, New York, NY, USA, 2014. ACM. ISBN 9781450327459. doi: 10.1145/2567948.2579359. URL http://dl.acm.org/citation.cfm?doid=2567948.2579359.

[43] Michail Vlachos, Philip Yu, and Vittorio Castelli. On Periodicity Detection and Structural Periodic Similarity. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 449–460. Society for Industrial and Applied Mathematics, Philadelphia, PA, apr 2005. doi: 10.1137/1.9781611972757.40. URL http://epubs.siam.org/doi/abs/10.1137/1.9781611972757.40.

[44] Scott Wen-tau Yih and Chris Meek. Learning Vector Representations for Similarity Measures, 2010.

[45] David Zhao, Issa Traore, Ali Ghorbani, and Bassam Sayed. Peer to peer botnet detection based on flow intervals. In *Information Security and ...*, volume 376 AICT, pages 87–102, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-30436-1{_}8.