UNIVERSITY OF OSLO
Department of Informatics

Security Incident detection with passive DNS logs

Master thesis

Martin Boasson Nordby

May 2, 2014

Abstract

This thesis investigates the potential for using log data gathered by DNS sensors to automatically detect previously known and unknown malicious domains and possibly infected clients. Results show that this is possible by applying a set of techniques for analyzing the domains queried. In addition to the analytical features, whitelisting can be used to reduce the dataset size and blacklists can be used to match the domains against possible reputation hits.

The system created is stealthy by design, meaning that no outbound requests need to be made during the analysis. Results demonstrate that this method of analyzing DNS traffic has a high detection rate, which means that it can be very useful in practical settings. We see a potential for expanding and improving the system, which most likely would enhance the system's detection capabilities.

As far as creating a stealthy, multi-featured reputation based system for malicious and infected host detection – this has, to our knowledge not been done in a similar way before.

Implementation of this system in a computer network offers the ability to detect malicious traffic not detected by other mechanisms.

Acknowledgement

This Master's thesis marks the end of my Master studies at the University of Oslo. The last year has been extremely interesting, allowing me to study and get immersed in a field of computer science I have a great passion for; information security.

Great thanks to all my colleges at mnemonic. During the past months mnemonic has been my second home, welcoming me with an academic environment, conversations, smiles and a friendly atmosphere. A special thanks to Joakim Von Brandis for all the time spent, discussions and knowledge shared. You make me see things from different points of view, which inspires me to figure out the rest on my own. I would also like to thank Jan Henrik Schou Straumsheim for reading, comments and constructive criticism.

Furthermore, I would like to express my sincere gratitude to my supervisors Joakim Von Brandis and Audun Jøsang for the invaluable support, comments, remarks and engagement throughout the learning process of this thesis. Additionally, I would like to thank Tom Danielsen for useful help and discussions regarding various technologies. I would also like to thank my family and friends who have been extremely supportive and positive throughout the project.

The result would not have been the same without you - thank you all.

Table of Contents

List of Figures

List of Tables

1 Introduction

The world is globally interconnected. The Internet and the World Wide Web (WWW) has become a driver for innovation, economic and social development. The global economy is becoming increasingly reliant on these technologies, allowing unprecedented speed, collaboration and exchange of information. However, at the same time as our society embraces this development, we expose ourselves to new types of risk. Criminals have quickly adapted to take advantage of the systems we rely on to create, store and manage information that is of value to us. Cyberspace has emerged as a new battle space for conflicts both between nation-states, and against organized crime. Networks and clients, both public and private, are facing constant attacks and must be protected against opponents and attackers. As the Internet and the World Wide Web now likely serves as society's most critical infrastructure asset, we are forced to examine new avenues of approach to ensure these systems can be secured and trusted.

This thesis explores the potential for of using log data from DNS (Domain Name System) traffic to automatically detect malicious domains by using a combination of domain and IP reputation combined with other analytical metrics. Based on the specific findings, we are able to flag clients as possibly infected. Results show that this method has great potential for identifying malicious domains and infected clients, and indirectly for detecting security incidents.

2 Research Questions

Detecting malicious domains and clients infected by malicious software (malware) is a constant challenge for companies and people working with information security. Targeted attacks and cyber-espionage is a constantly growing threat for companies and governments around the world, especially companies that are leading within their fields of practice and that have valuable information they wish to keep confidential. Compromised clients in the network can lead to information leakage, further compromise of additional hosts, loss of trade secrets and economic data, and can result in considerable economic and reputational losses for the owners and stakeholders.

Various technical controls can be deployed in an active computer network to analyze traffic flow and trigger events and alarms upon certain traffic patterns. For example different kinds of Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs). This thesis looks at the possibility of using data captured by DNS sensors - both DNS queries and answers - apply an algorithm and automatically detect malicious domains, in addition to detecting clients having a certain probability of being compromised. Clients can be infected by a variety of vectors, and detecting these can be a challenge without having an extensive implementation of different sensors in the network. The infection vector is often not through DNS, although all callback traffic that is not done directly against an IP address avoiding DNS will use DNS to complete the traffic. The technical solution investigated in this thesis is to detect malicious domain names queried, providing an easy to integrate system, capable of detecting those clients by looking at DNS data.

Based on this approach, the research questions for this thesis is formulated as follows:

Q1) How to design a practical system for detecting malicious domains and infected clients based on monitoring DNS queries and answers?

Q2) What is a realistic detection rate of malicious domains detected with this system?

Q3) What is the nature of detected malicious domains?

3 Research Method

The research method used in this thesis was structured to be a design research[1], carried out in a cyclic research process consistent of the following elements: Awareness of the problem, suggestion to the acknowledged problem, artifact development, evaluation and finally conclusion. Figure 1 shows an illustration of the cyclic research method used.

In the awareness phase, the research problem was identified. During the suggestion phase, a technical solution to solve the problem was presented, thought out and discussed with colleagues and supervisors. In the development phase an artifact was developed. Next, an evaluation of the implemented solution was performed and tested. If the results were satisfactory the technical solution and implementation could be accepted. In the event of unsatisfactory results, it was possible to go back to the previous stages (awareness, suggestion and artifact development).

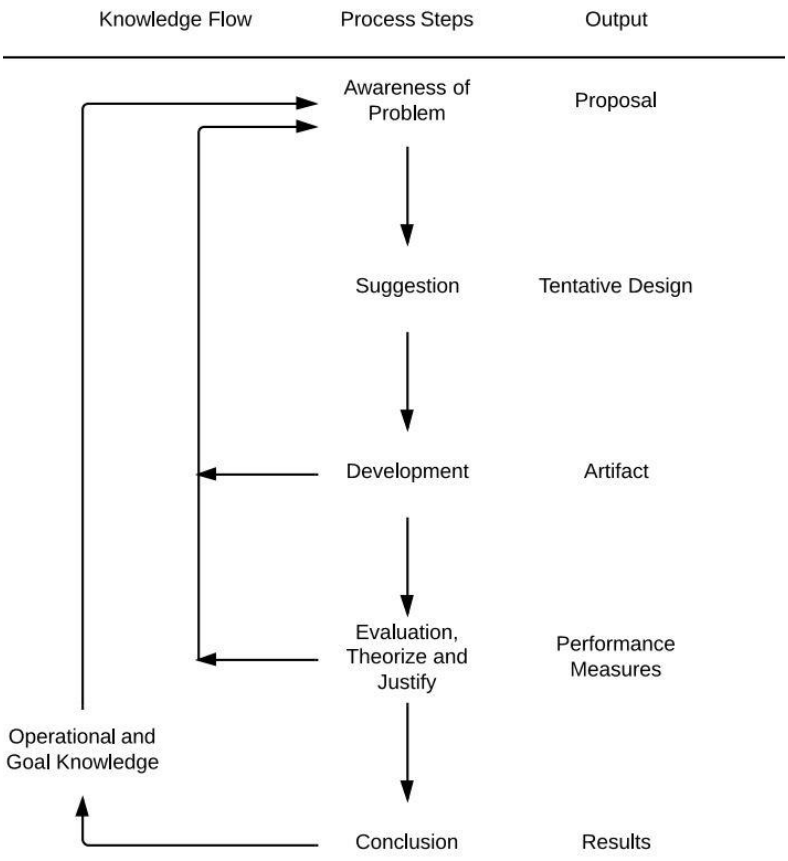

Figure 1 - Overview of Research Process[1]

4 Overview of the Chapters

Ch5. Background

In this chapter central background knowledge of DNS, passive DNS (PDNS), active DNS (ADNS) and reputation lists are presented. Differences between ADNS and PDNS, and how PDNS can used to detect security incidents are highlighted and introduced to have the necessary knowledge base for the rest of the thesis.

Ch6. Previous Work

This chapter presents previous work relevant for the thesis. Other systems analyzing PDNS data are described, along with key-features that are highlighted. The differences between the systems and the system created for this master thesis is sustainable, a technical foundation for the motivation of creating this system.

Ch7. Technical

This chapter describes the system created for this master thesis, ADomDec (Automatic Domain Detection). Motivation, technical choices, system design, detection metrics and program flow are presented.

Ch8. Results, Discussion and Conclusion

This chapter summarizes the results, discusses them and presents thoughts for future work. My experiences are deeply embedded throughout all topics, ending the chapter with a conclusion.

5 Background

5.1 DNS and Passive DNS

Fully understanding how passive DNS works requires a thorough understanding of the Domain Name System (DNS). This section will first briefly describe how DNS works before focusing on differences between passive DNS vs. active DNS, and how PDNS can be useful from a computer security standpoint.

5.1.1 DNS

In short, DNS can be described as a key-value stored dictionary placed in a hierarchal system for domain names and IP addresses (both ways), for the internet, running on port 53. It is structured as a tree-structure of domain names, with a root domain at the top[2], as shown in Figure 2 - Domain Name Space Tree[1].

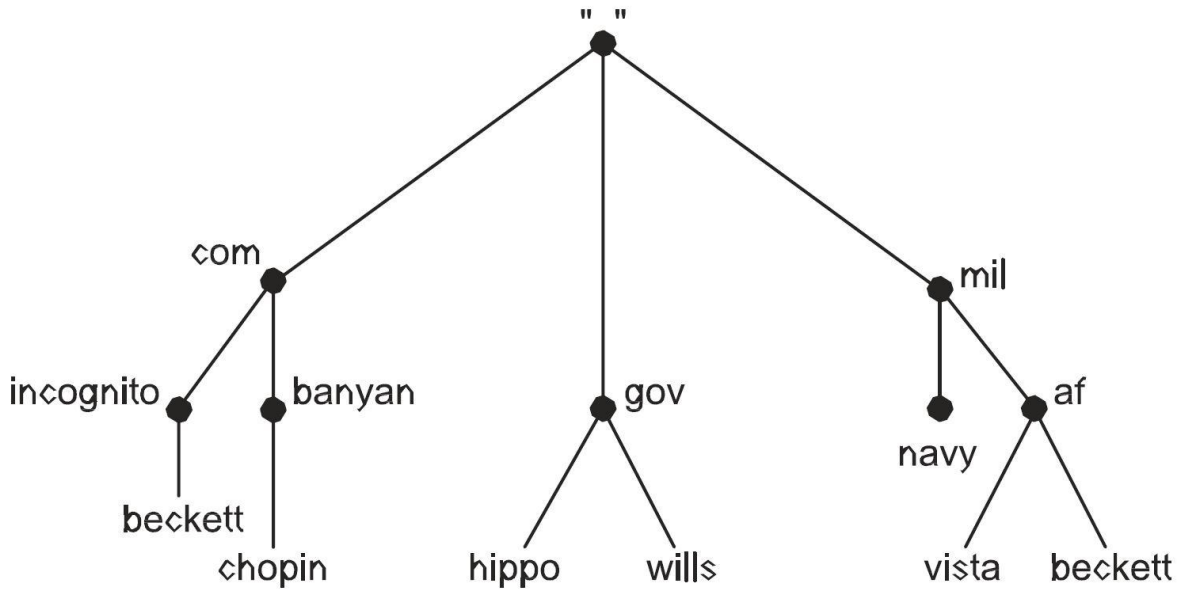

Figure 2 - Domain Name Space Tree[1]

There are many different types of classes and types[3] used in DNS queries, defining the kind of request being made. The most common ones are class “IN”, type “A” – requesting an IPv4

host address for internet service. Figure 3 - DNS query to answer", shows an illustration of a fictitious query for the domain "offer.store.ebay.com":

- The resolver sends the request to the DNS, which bounces the request to the root node.
- The node reads the request backwards and tells DNS which node to ask next.
- The "com" node continues where the node left off, and tells DNS to continue at "ebay".
- The "ebay" node does the same thing.
- At the final node the actual address is returned, there are no further nodes that need involvement to find the address.
- When the DNS gets the answer, it sends it to the resolver, which uses the answer for its intended purpose.

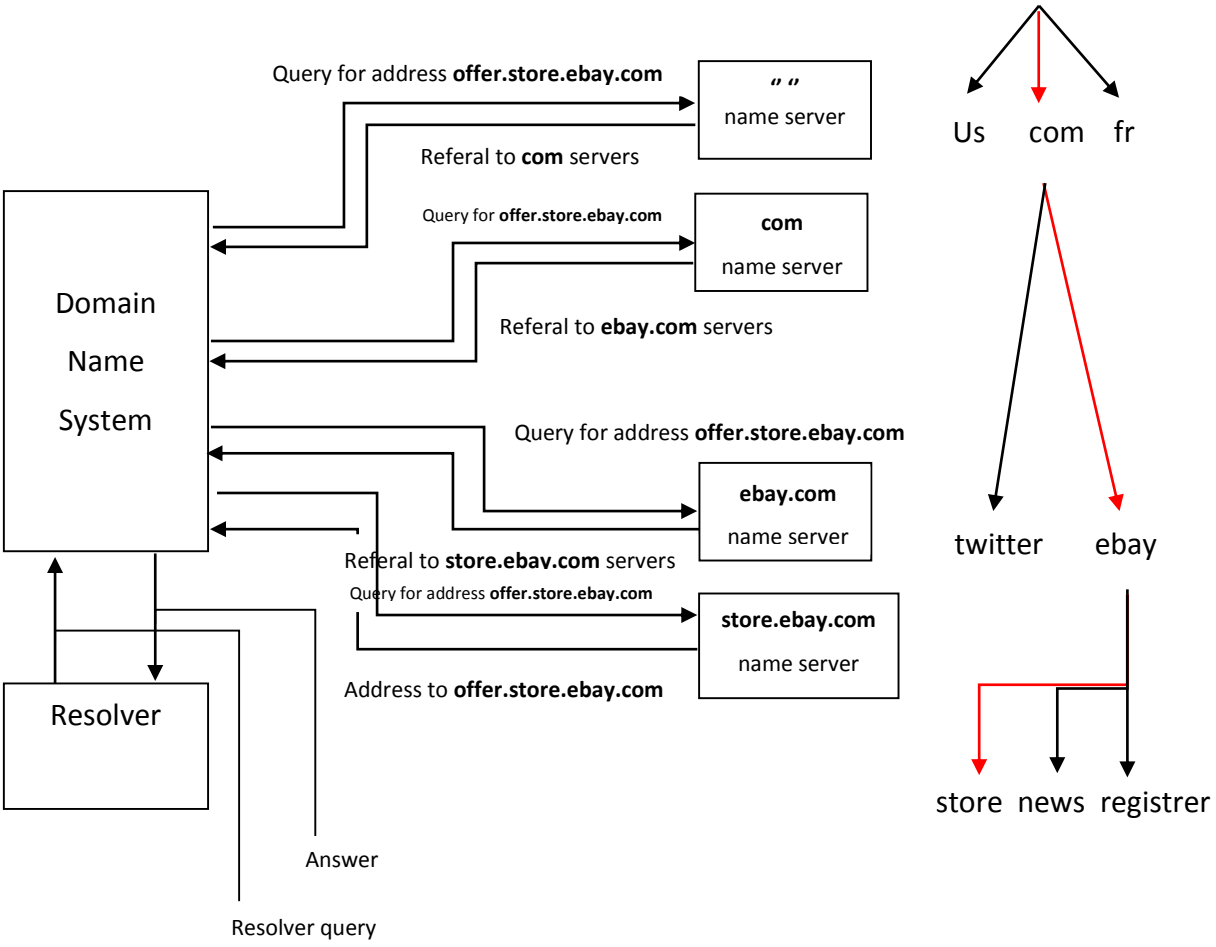

Figure 3 - DNS query to answer

5.1.1.1 Abuse of DNS

DNS cache poisoning

When the DNS query is not cached, the caching DNS server has to ask the authoritative DNS server for the correct answer. Next time the query arrives; the caching server will have the answer, and will be able to provide the answer directly. Attackers can exploit this lack of knowledge in the caching server and race the authoritative server to the update by providing an answer before the authoritative server does [4]. This attack is known as DNS cache poisoning, which is one method for performing DNS spoofing.

The effect will be that the record stored in the caching server points at somewhere the attackers want it to point. When a user requests the domain in record the attacker has inserted, the user will be directed there. The malicious redirect can lead to phishing of user credentials, exposure to malicious code, or other infection vectors.

Man-in-the-Middle Spoofing

Another way of performing DNS spoofing is to perform a Man-in-the-Middle (MitM) attack. An attacker can position himself between the client and the DNS server, intercepting traffic. When a DNS request is made, the attacker replies with the malicious answer. The result is the client using the malicious record, without knowledge of doing so.

Malicious DNS server

Attackers can also take over the authoritative DNS server itself. The effect of this is global. All caching servers asking the infected authoritative DNS server will pull malicious records. The amount of clients affected will depend upon the popularity, and amount of requests for the record.

Compromised Registrar

In the DNS spoofing category, taking over the registrar is by far the worst and most serious one. If attackers can take over the registrar, the instance responsible of issuing and registrate the domain itself, they could change the authoritative DNS servers in a greater

fashion than a direct poisoning. There would neither be a sign of any malicious activity taking place, since the changes come from a trusted source considered to perform benign operations. The attackers could change and insert records in the authoritative DNS server, reaching a high amount of clients, possibly causing enormous amounts of malicious redirects.

DNS Amplification Attack

DNS amplification attacks is a way to perform Distributed Denial of Service (DDoS) by directing DNS response traffic from public open DNS servers towards a targeted systems. The attacker sends a DNS query to the DNS server, but spoofing the source address to be the targeted address. The answer from the DNS server it is sent to the target – flooding it with traffic. The attackers often sets the record type to "ANY", to ensure the largest amount of traffic sent to the target. By initiating a botnet to send a considerable amount of spoofed DNS requests, attackers can achieve tremendous amounts of data with little effort. Since the data arrives from legitimate DNS servers, this type of attack is hard to detect and prevent.

5.1.2 Passive DNS

Passive DNS (PDNS) is a technique invented in 2004 by Florian Weimer. This technique is used to organize and reconstruct history from DNS by logging queries, answers and metadata into a database where the data can be organized and indexed. The data is collected before being forwarded to further analysis.

Usage of PDNS

PDNS can be used in different ways, serving different purposes. Often different fields of use can overlap and be integrated in the same system. Since DNS changes with time, a PDNS system can provide information regarding DNS history. This can be useful to get an overview of[5]:

1) Where the domain name pointed to in the past.
2) Which domain names that are hosted by a given nameserver.
3) Which domain names that point into a given IP network.

4) Which subdomains that exist below a certain domain name.

Different DNS servers can provide different answers to queries depending on the caches, and whether or not they have been compromised. If the PDNSDB shows i.e. 5million entries for two different IP addresses located in the US for “ebay.com”, and suddenly resolves to a new third IP address in the Far East – this would cause reason for suspicion and be worth looking into. By logging DNS queries and answers, data from many different DNS servers will be logged, giving a wide dataset. Having DNS history implicitly also means access to the geographical location the services resolved from at the queried time. An anomaly in geographical location can also be an indication of a compromised DNS server, infected with cache poisoning or the DNS request was intercepted and a Man-in-The-Middle attacked could have been performed. This is useful from a security point of view, since check-in (also known as Command-and-Control – CC) traffic patterns often are scattered over a wide geographical area. Having modules in the system to visualize geographical information, number of subdomains and other features can help attain a better picture of the nodes. This can be useful for both statistical purposes and for an analyzer looking at traffic.

5.1.2.1 Different kinds of PDNS systems

Simple logger

The system can be logging the data without further analysis. This is the minimum requirement for a system to function as a PDNS system. Without logging the data, the system loses the ability to draw a timeline of relationships between different domains and IPs, measure average of the time to live (TTL) value and other comparisons necessary in active use of data. Then it would simply be looking at DNS data without making further use of it, in other words a quite meaningless system. A simple logger would function as a database for manual lookups, or a backend for automatically getting DNS history from other systems, which could be useful.

Passive analysis of DNS

A system that covers the minimum requirement of logging the data in addition to performing further analysis can detect malicious domains previously unknown, fast flux networks and infected clients. Fast flux networks are networks administrated by cybercriminals with an ever-changing infrastructure to hide their primary nodes. By having a high number of IP addresses pointing to one qualified domain name, and changing the IP addresses with a high frequency, combined with using infected clients as proxies help to further disguise the traffic flow. This complicates the detection of the fast flux network and underlying structure. Previous work describing these features is located in Section 6: Previous Work.

Difference between PDNS and ADNS

A pure PDNS system, logs the data, makes passive analysis and outputs the result in a desirable form. Some makes active use of DNS queries in their analysis, making it an Active DNS system. This can be useful to gain a live overview of the cybercriminals network, but making active DNS queries to probe the network is not without risk. If the administrator of the illegitimate network detects a system actively probing their network, they could simply discard the requests, in effect making the system useless beyond that point.

Cybercriminals can for instance make use of access control lists (ACL), to provide different answers depending on who makes the request. During analysis, the system would receive an answer, but it is not given that the client observed from the logs received the same result earlier. By passively observing the data, without making active DNS queries during analysis, the system can remain stealthy and undetected by cybercriminals administrating the domains/networks which the system reveals as malicious.

PDNS analysis combined with reputation sources can be a powerful combination in the quest to detect malicious domains. PDNS provides a historical picture of DNS, not just a snapshot of the current picture of DNS (which also can vary depending on which DNS server you ask). Having historical information on DNS can be used to make a timeline of IP addresses or

domain names, where they have resolved in the past, their average TTL value, number of subdomains, etc. – information that normally is lost. Logging the information and making active use of it during analysis can help finding indicators of malicious activity.

The following section describes reputation as a whole, usage of reputation and different kinds of challenges that emerge while dealing with implementation of reputation in an active system.

5.2 Reputation

Reputation in the context of computer science is related to the trustworthiness of individuals, traffic patterns, domains, IP addresses and firms. Cybercrime has over the past decade become an increasing threat towards companies and end-users of the Internet. This makes the focus and awareness on information security increase as well.

Since computers have the force of automation – and therefore inhuman speed – criminals and companies have used this to their advantage. For example, this can be achieved either by sending out large amounts of e-mails containing a fraudulent offer (one that often does not exist), sending out specially crafted e-mails containing viruses that infects the recipient's computer, distributing malware and malicious code or trying to trick the recipient into giving away money.

The IP addresses and domains used in the different malicious attacks ends up in reputation lists, further discussed in 5.2.1: "Reputation Lists". Because of the enormous size of the Internet, allowing a high number of different IP addresses and domain names, reputation lists function as a way to remember the known-bad and known-good.

Reputation in this Master's is limited to reputation lists containing domains and IP addresses, described closer in the following sections.

5.2.1 Reputation Lists

After observing huge amounts of spam-mail and known bad content coming from the same IP addresses and domains, network administrators and information security vendors formed

a local picture of reputation[6-8]. To structure this in a more generic way, lists of known good and known bad IP's and domain names were made. These lists were naturally split into two main categories: Domain lists and IP lists.

5.2.1.1 Domain lists

The domain lists contains a list of domain names. These lists can either contain domain names known to be good (typically well-known vendors and companies such as Facebook, Ebay, and other companies who keep their sites clean). On the other hand, such lists can also contain domains known to be bad - domains known for distributing malicious code or serve as phishing sites. There are different ways of organizing these lists, and how much information the lists contain. The list can be limited to only containing domain names, or it can have additional meta-data fields such as when the domain triggered, the last incident, how long the domain the domain is kept in the list and when it first triggered a match.

5.2.1.2 IP-lists

IP-lists are similar to domain lists, only with IP addresses instead of domain names. Hackers and other actors that spread malware and code with malicious intent often do not have a specific domain name. The GET-request for the code can also be build using only the IP address. The IP-lists can also contain either just the IP-address or other meta-data information such as in the domain-lists.

5.2.1.3 Black-lists

Black-lists contains either IP-addresses or domain names of sites known for distributing malware, adware, IP addresses or domain names related to spam-emails, exploit-kits and other code of malicious content. Everything that the system should trigger on and classify as malicious will be gathered and placed in a black-list. New lists can be generated dynamically by looking at callback and check-in traffic made by malicious software and compromised clients. This will be described more in detail in section 5.2.3, Dynamic creation of new lists.

Black-lists are gathered from vendors publishing black-lists, as a part of an information exchange contract or as a part of creating your own lists. Since black-lists are based on reputation of the domain name or IP address, the trustworthiness will vary depending on the source generating the list. A single hit in an external list would have a lower confidence than hits from multiple lists from known security vendors, or hits from a list generated by your own system which you control yourself.

5.2.1.4 White-lists

White-lists are either domain or IP lists that contain known benign sites and IP addresses. This often includes domain names and IP addresses of large companies and well-known vendors and services. If one of these IP's or domain names get placed in one of the black-lists (lists containing known bad domain names and IP addresses), it could be a major source of false positives. One could classify them as benign either by either removing them from the black-list or adding them to the white list. If the domain or IP is located in the white-list the severity of the incident will be lowered and ignored.

Using white-lists can speed up the processing time because it reduces the dataset sent to further analysis. Using white-lists can also have a downside if one of the domains in the white-lists becomes compromised. If other mechanisms are not implemented, whitelisting it could allow the traffic to pass undetected.

5.2.2 Usage of reputation with DNS

Reputation can from an information security point of view be used to detect security incidents in a computer network when reputational information is used to augment logs from other sources. It can help to detect and identify callback traffic, landing pages, redirects, requests for exploits and payloads, and much more.

This section looks at the use of reputation matching against DNS and why it could be useful.

By matching DNS queries to reputation lists, the acting system can detect queries and

responses for IP addresses and domains that are considered malicious by the list. The system can either drop the request directly denying the user to fulfill the wanted request, or let it pass. The wanted action depends upon the architecture and function of the system. If an IP address is known for being a Control and Command server (C&C)[9-11] or a domain is known for hosting malicious content, it could be a reasonable idea to drop the request. At best, this can prevent an infection. If a request is matched, and still let through – a notification should be sent to an acting instance informing that the client behind the given IP address is likely compromised. Even though the outbound request is blocked, the client is still compromised and should be investigated further immediately. Cybercriminals strive to gather information about the network and client, further infect the network or other clients and gather sensitive information. Detecting and blocking communication could prevent this.

If there is a false positive in the blacklist, blocking traffic based on reputation hits can deny legitimate traffic. This could be annoying for the user, create unnecessary business disruption, and could place an unwanted load on the system.

5.2.3 Dynamic creation of new lists

Domain blacklists can be generated in many different ways, giving different kinds of blacklists. Running an infected client in a lab-environment will provide solid and trustworthy data regarding the exploit-kit(s)/malware the client is infected with since the sensors can observe and capture live data. Carving spam e-mails, gathering data from spam-traps and filters will provide lists with domains and IP addresses used in phishing campaigns (such done as a part of “RB-Seeker”[12]), spear-phishing emails and instant-messenger worms[13].

Different lists will have independent trustworthiness, depending on the origin of the data. C&C domains carved from an infected computer in a lab-environment will have a high level of trustworthiness, while a five year old email-spam domain might not be as relevant. Using this information, the system in action can scale events up or down in severity giving the best reputation score possible.

While matching IP addresses and domain names against black-lists one can generate new lists based on the findings and the other traffic generated by the event. When observing a malicious file, running it inside a virtual machine (VM) in a lab-environment, one can dump all traffic made using full-capture on the outside of the machine[14]. By analyzing this traffic, IP addresses and domain names of C&C-servers, URI-structure of landing, redirects and payload pages specific for that exploit kit and the configuration files and updates sent by the C&C-server later can be observed. When IDSs parse traffic, traffic observed triggered by alarms with high trustworthiness can be placed in a reputation list. This provides a rather wide range, since the signature sets in use can cover a wide traffic field. The greatest challenge using this as a technique to generate lists is to select the signatures with an adequate level of certainty.

Using this information, it is possible to cross-reference the new IP addresses and domain names with the ones already existing in the blacklist. If there are any new entries, these can be added to the existing list, or they can be put in a new list – depending on the list hierarchy in use.

5.2.4 Cybercriminals' usage of black-lists

The people administrating and maintaining the servers hosting the malicious code (whether compromised or not), also use blacklists[15] to block vendors and systems wanting malware. The benefit of not serving code to companies within the Anti-Virus (AV) sector, or other sectors working on information security is of strictly administrative and economic reasons. If companies developing AV software or other companies that can benefit of knowing the internal structure of malware got the source code of the malicious software., they could reverse-engineer the code, see how it functions and make signatures to detect it. This would make the malicious software "harmless" in the sense that it either would not work, or it would be blocked somewhere before even reaching the client.

Knowing how the malware works and which modules it exploits to infect the client is a crucial part when attempting to prevent infections of new clients. The development time of the malicious software would be a waste, and they would have to make a new kit, unknown to the vendors. This game of cat and mouse is a continuous struggle in the war against

unwanted and malicious software made by cyber-criminals. In the recent years, the cyber-criminals have invested an increasing amount of work, time and money to develop malicious exploit-kits. These are either for sale on the black market as generic malware with support, or for own gain. “Zeus”[16, 17] is a good example of a Trojan package easily configured to steal money from online banking activity.

5.2.5 Challenges of reputation lists

Whitelists have the potential to significantly ease the workload of the system. If some of the traffic can be categorized as safe already at the very first module of the system, no further analysis is needed. On the other hand, if a domain that should be in a blacklist manages to sneak its way into the whitelist it can pass through undetected. This could potentially take time to find out and verify. If, on the other hand, it were the other way around, a whitelisted domain in the blacklist, the system would generate many false-positives FP’s.

A known domain such as “facebook.com” would for the most part be considered safe, but sites could be vulnerable for cross-site-scripting (XSS[18]) and some sites allow external plug-ins. Even legitimate websites, no matter how famous and big, always run a risk of becoming compromised. Systems should have a threshold for when to take a domain or IP out of the whitelist, and define a standard for how long it is considered malicious. Well supported sites will likely clean up their site as fast as possible, because they have both the skill and motivation to do so – spreading malicious content for a longer period of time will decrease the credibility of the company leading to a potential financial loss or damage to their brand and reputation.

Trustworthiness of reputation lists

Different lists can be shared and gathered, raising the question of trustworthiness. How was the data collected, when was the domain classified malicious, has the site cleaned up and should it be out of the list, was it manually made or automatically generated – these are all relevant questions when setting a trustworthiness score of the list. Lists generated automatically from an unknown external source should have a lower score than a list made

manually from domains and IP addresses seen involved in actual attacks. If a domain or IP address exists in several lists, the trustworthiness of the entry is higher and the severity of the event can be escalated.

6 Previous Work

6.1 Previous work

Systems for automatically detecting malicious domains through different approaches have been discussed in previous work. Some examples of the systems that have been developed are “EXPOSURE” and “Fluxbuster”. This chapter will describe these PDNS-related systems, how the systems work and what they accomplish. Together they highlight important features about DNS traffic that make the foundation for the system made for this master thesis.

6.1.1 EXPOSURE

EXPOSURE[19] is a system designed to automatically detect previously unknown malicious domains using a rather small dataset (in context of DNS answer/query amount), to first train the system offline before setting it inline to capture and process data real time. In their experiment EXPOSURE was deployed for two weeks with an ISP, capturing real-time data to prove scalability and the ability to detect and categorize malicious domains. In comparison to Notos[14] which needs more training because of an dynamic reputation score being set – EXPOSURE requires one week of offline training using a varied dataset before being deployed in a production environment.

6.1.1.1 How does EXPOSURE work?

EXPOSURE is divided into four feature categories: Time-based features, DNS answer-based features, TTL value based features and Domain name-based features. Counting 15 sub-features, nine of which not previously mentioned in other research. They define a malicious domain not only as a generic term aimed to cover all malicious activity, but also divided them in ten different categories:

"We divided the domains into ten groups: spam domains (Spam), black-listed domains (BlackList), malicious Fast-Flux domains (FastFlux), domains that are queried by malware that are analyzed by malware analysis tools (Malware), Conficker domains (Conficker), domains that have adult content, domains that are suspected to be risky by Norton Safe Web and McAfee Site Advisor (Risky), phishing domains (Phishing), domains about which we were not able to get any information either from Google or from other sources (No Info), and finally, benign domains that are detected to be malicious (False Positives)." [19]

The sub-features for the main categories are:

Time-based features:

- short life
- daily similarity
- repeating patterns
- access ratio

DNS-answer based features:

- Number of distinct IP adresses
- Number of distinct countries
- Number of domains share the IP with
- Reverse DNS query results

TTL-based features:

- Averege TTL
- Standard Deviation of TTL
- Number of distinct TTL values
- Number of TTL changes
- Percentage usage of specific TTL ranges

Domain name-based features:

- % of numerical characters
- % of the length of the LMS

There are five main components in EXPOSURE, which can be seen in the overview, Figure 4 - Overview of EXPOSURE[19].

I. **Data Collector**

The first component is the data collector. This component records the DNS traffic in the network where EXPOSURE is placed. Between the first and second component, the DNS records gets logged to the database.

II. **Feature Attribution**

The second component is the feature attribution. "This component is responsible for attributing the domains that are recorded to the database with the features that we are looking for in the DNS traffic." [19]

III. **Malicious / Benign Domains Collector**

The third component, the malicious and benign domains collector gathers and correlates white and blacklists of domains (for more information regarding reputation, see 5.2 Reputation). This information is used to label the domain after the Feature attribution is done. After the third component, the data is labeled; depending on the label, the data goes to either the fourth, the learning module, or the fifth component, the classifier.

IV. **Learning Module**

The fourth component is the learning module. It trains the labeled set of data to make models and detect malicious domains.

V. **Classifier**

The fifth and last component is the classifier. At this component the final decision is made whether the domain is to be considered malicious or not.

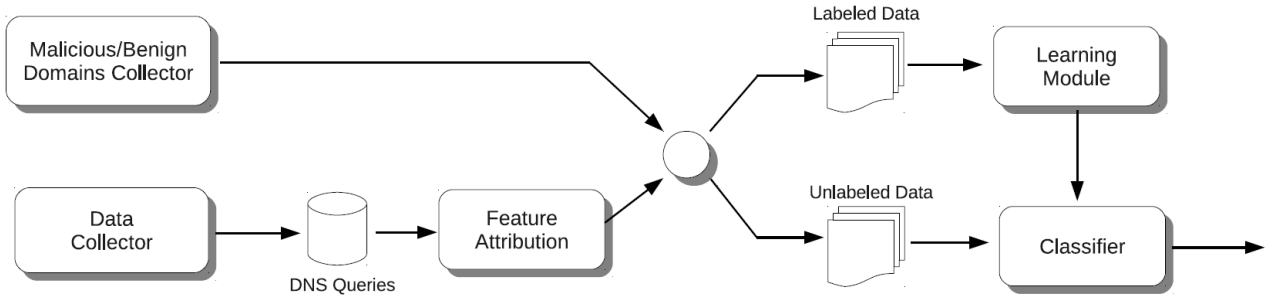

Figure 4 - Overview of EXPOSURE

Important Features

There is especially one feature that will be used by the system constructed for this master thesis; average TTL value. After the researchers of EXPOSURE had tracked domains used by the Conficker worm[20, 21] for a week, they observed that malicious domains had a higher number of changes in TTL values and the total number of different TTL values tend to vary more than for benign domains. They investigated the ranges [0, 1), [1, 10), [10, 100), [100, 300), [300, 900), [900, ∞), and the malicious domains had a significant peak in the range [0, 100).

6.1.2 FluxBuster

FluxBuster is "… a novel passive DNS traffic analysis system for detecting and tracking malicious flux networks"[22]. Cybercriminals are constantly looking for new ways to cover their malicious tracks. Flux networks is one their techniques. Detecting flux networks gives white hats an advantage in this continuous war. A flux network resembles an illegitimate content-delivery-network (CDN). Real legitimate CND's have existed for a while to provide a scalability, reliability and to provide good performance for high volume Internet services. The network often consists of multiple servers placed all over the world. When a user requests a service, the CDN sends the content from the logically placed closest node. In a legitimate network, the nodes are administrated, providing a reliable service for the users. In an illegitimate flux network, the nodes consists of flux-agents that are malware-compromised clients in a botnet. The flux-agents are controlled by a botmaster – commonly known to serve malicious content, phising websites, illegal adult content and other things used for malicious purposes[22].

A main difference between a legitimate CDN and an illegitimate flux network is that a legitimate CDN has stable, reliable servers making uptime and load balancing possible. Since the flux-agents consists of compromised clients, the botmaster has a difficult job of knowing which agents that is reachable and not, since the owner can turn the computer off at any time. Furthermore, the botmaster does not always have a good overview of the amount of traffic in the flux network, making load-balancing difficult.

In contrast to the majority of the previous work, FluxBuster does not make use of active DNS queries in the analysis. The previous systems heavily relied on spam emails as main information source. These systems carved out domains and IPs used in spam emails, and actively performed DNS queries to probe the malicious flux network. The dimension of the blacklists used were too small to make a significant matter during the analysis[22]. By only observing the DNS data, FluxBuster captures flux networks advertised through blog spam, social websites spam, search engine spam and instant messaging spam, in addition to e-mail spam and precompiled blacklists[22].

6.1.2.1 How does FluxBuster work?

FluxBuster consists of four main components: DNS message aggregator, message pre-filtering, domain clustering and the classifier.

I. **DNS Message Aggregator**

The DNS message aggregator module aggregates all DNS queries into higher-level DNS messages that contain all information regarding the queried domain with a given timespan (at least a few hours to ensure collection of enough information[22]). The high-level DNS messages contain metadata, such as amount of identical queries made, which IP addresses that maps to the same domain, first/last seen and average TTL value for the queries made within that time span.

II. **Message Pre-filtering**

The message pre-filtering module analyses the aggregated DNS messages from the previous module, and filters out everything that is not likely to be a flux network. In effect, the module functions as a data reduction module to save computation time by reducing the amount of data passing through to further analysis. This module is configured with a very conservative rule set, letting all the flux networks but also benign clusters through[22]. All clusters passed to the later components are possible flux network candidates.

III. **Domain Clustering**

The domain clustering module gathers the aggregated DNS messages that were let through the pre-filtering into clusters making the final flux network candidates.

IV. **Classifier**

The classifier is the last active instance of the FluxBuster. Here the final decision whether the cluster is considered to be a flux or non-flux network is made.

6.1.2.2 Characteristics and statistical features of flux networks

FluxBuster uses a series of features to determine whether a flux network is a regular legitimate CDN or an illegitimate flux network. They define the following variables used in their calculations: *C* is a generic domain cluster computed at the end of E_m, which is the epoch (timespan). *R* represents the set of all distinct resolved IP addresses during E_m that are related to the domains in *C* [22].

1) Number of resolved IP addresses to a domain in *R*.
2) Number of distinct domain names in *C*.
3) Average TTL value for the records within the timespan, E_m.
4) How many domains that share at least one resolved IP address from *C* in previous epochs.
5) The IP diversity based on a normalized entropy of the /16 network prefixes of the IPs in the set of all distinct resolved IP addresses collected during the epoch. FluxBuster computes it as follows:

$$\frac{-\sum x(p(x) \cdot log2p(x)}{log2(\varphi 1)}$$

Where φ1 is the results of nr1, and x is the network prefix. "Where the probability *p(x)* is given by the relative frequency of the network prefix *x*."[22]

6) The growth ratio of IP addresses, based on the number of new IP addresses discovered during the epoch, per each DNS query related to the domains in *C*.
7) The last growth ratio of IP addresses, finding the average number of new IP addresses per DNS query. This is computed in two versions; one value which is the average of independent domains in the domain cluster C, and one value by analyzing the last de-duplicated messages among all the messages related to the domains in the cluster.
8) The last growth ratio of IP prefixes. This feature is quite similar to nr7, but it computes the average number of /16 networks prefixes per DNS query discovered by analyzing the last de-duplicated DNS messages associated with each domain in the cluster during the time epoch. It focuses on new IP prefixes differentiating it from the previous feature. This feature also calculates two values.
9) The last feature computes the novelty of the difference between IPs seen through previous time epochs that have not been seen in the last, divided

by the number of time epochs taken into the equation. It calculates three values, used at thresholds of other parts of the system.

These features aim to define and catch the main characteristics of malicious flux networks. All clusters are considered potential flux networks. At the end of a time epoch, the features listed above are applied before letting a decision-tree classifier either classify a cluster as a malicious flux network, or a legitimate/non-flux network.

Setup and Training Set

When forming FluxBuster the authors used a labeled dataset, consisting of 10 months of data collection that was collected in the period from June 2010 to March 2011 through the ISC Security Information Exchange. They ended up using about four months of the dataset to build a labeled dataset (LDS). Their motivation for using a LDS had two reasons:

1. Estimating the accuracy of the Classifier module through cross-validation.
2. Train the Classifier module before deployment.

When making the LDS they used a semi-manual process to create a training set of clear-cut domain clusters either malicious or benign. In total, they classified 1337 domain clusters as flux, 5708 as non-flux and 313 as unknown. Clusters that were suspicious, but where insufficient information made it impossible to make a clear decision, were classified as unknown. The domain clusters categorized as unknown were not used. The process was partially automated by making use of prior information regarding flux networks, known malicious domains and legitimate popular domains.

Correctness and Data Evaluation of FluxBuster

Evaluating datasets classified by FluxBuster as a malicious flux network is a difficult task. Obtaining complete ground truth of a dataset is challenging because the domain cluster can fall into three categories:

1. The domain cluster may include domains or IPs that are malicious and known to be part of a flux network. In which case it would be a true positive.
2. The domain cluster is a pure false positive; it does not contain anything malicious. The domain cluster represents a legitimate service and an actual CDN.
3. There is no information available in the public, or even in private datasets containing information regarding the domain cluster in question. This could indicate a zero-day exploit, or previously unknown activity, but in practice, it is impossible to know and it will require manual analysis to determine the true nature of the dataset.

To determine the different outcomes they used different kinds of black/white lists publicly available.

7 Technical

7.1 What does the system do?

The system created for this master thesis is called ADomDec (Automatic Domain Detection); it automatically detects malicious domains based on parsing of passive DNS logs, and detects possible compromised clients using the information detected by the system. The system applies a series of checks to the data before matching it against a threshold.

This chapter first describes the layout and design of the system, and then describes in detail the different features and how they are applied in the system before the threshold match is done.

7.1.1 What makes ADomDec different from previous work?

Previous systems makes use of a set of features to detect clusters of malicious domains, but do not focus on who makes the request. The sole focus of previous systems is centralized on the external structure of botnets. ADomDec does not build a timeline and does not cluster requests in epochs. Every request is considered individual but the history of DNS through the PDNS DB is used (as an advantage to derive additional information) in the analysis. The timeline of ADomDec starts from the first request is logged, and continues as long as the system is running.

In contrast to previous work described above, ADomDec has two main focus areas when performing the analysis. The primary focus is to detect malicious domains, but based on the findings clients are flagged as possibly infected in incidents. Displaying this through a user-friendly webUI or combining the findings with other existing correlating systems could be a valuable asset in network detection. ADomDec could also complement existing IDSs providing a richer dataset for security analysts when monitoring network traffic.

7.2 System

ADomDec is created to automatically detect domains categorized as malicious, previously known and unknown. Beneath is a basic overview of the technical aspects of the system, before going into detail describing each of the components.

7.2.1 Basic overview, design

ADomDec consists of 13 components (Figure 5 - Overview of ADomDec for a complete overview):

#	Component name
I.	Logfiles
II.	Data Collector
III.	Logfile parser
IV.	DB logger
V.	PDNS database
VI.	Reputation sorter
VII.	Reputation sources
VIII.	Reputation gatherer
IX.	Fine-grained analyzer
X.	Harmless container
XI.	Malicious database
XII.	Client checker
XIII.	Infected clients database

Table 1 – List of components in ADomDec

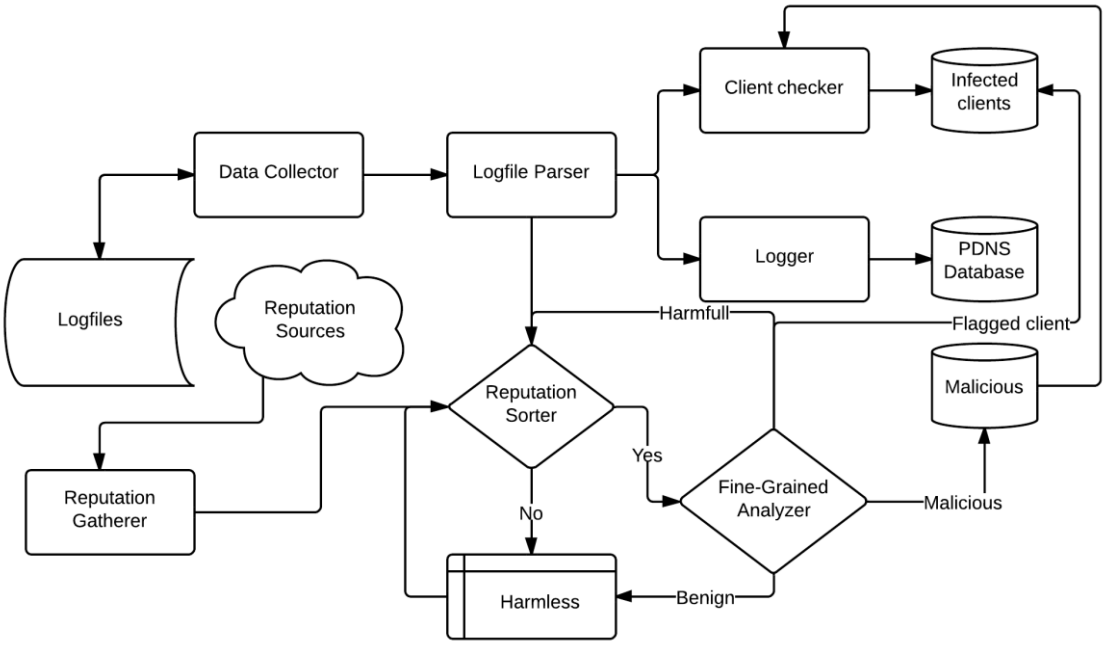

Figure 5 - Overview of ADomDec

Logfiles

Here lies the logfiles gathered from the sensor(s). For this Master's project the data is collected from a malware lab located locally at mnemonic AS[23] in Oslo. The data is generated by a mixture of malware infected clients and normal user behavior, ensuring a real dataset for analysis. It is non-sensitive data, and mnemonic AS has given consent for using the data in this Master's project. The logfiles are collected by the data collector component, and removed from the logfiles folder when the parsing is done.

Data Collector

The data collector component is a rather small component gathering the logfiles, and parsing them to the next component. After the "Logfile Parser" component has processed all the entries in the logfile, the data collector also removes the correct logfile from the logfiles folder.

Logfile parser

This component forwards different fields of data to other components. It parses the data from the logfiles, in a format that is simple to analyze later before forwarding data to the DB logger, client checker and the reputation sorter. It is a distribution component, placed to gain generativity in the system.

DB logger

This component receives data from the logfile parser. It checks if there exists an entry in the database for the dataset and either inserts or updates the data. This component is used to fill the database, and receives queries from a web interface.

PDNS database

This is the database containing all records from the logfiles. It receives data from the DB logger component, which either inserts or updates the entry depending on the existence of the data in the database. The database is also accessed from a web interface, were queries against the database can be made. The web interface makes it possible to view the data in the database in a user-friendly format.

Reputation sorter

This is where matching against reputation sources is done. If the domain name or IP address is matched against a whitelist, the request is discarded and no further analysis is made. Depending on the trustworthiness of the reputation list, a hit with a blacklist will give the event a certain score. If the domain name or IP address is triggered in several lists, the score value increases correspondingly.

Reputation sources

This module consists of available reputation sources that can be found online. Both blacklists and whitelists are included.

Reputation gatherer

This component maintains the reputation lists, download updates and has the possibility to manually whitelist and blacklist domains and IP's.

Fine-grained analyzer

This component analyses records that are not discarded as benign. This is done through a series of steps, each step adding a score to the domain. After the final step, the score is compared against a threshold. If the score is above the threshold, the domain is logged in the "Malicious" database, if not, it is discarded as benign. This component and its features are described in detail in the next section.

Harmless container

This component is a list made by the decisions from the previous component, the fine-grained analyzer. If the score is above a certain threshold, the domain name or IP address is considered malicious. It is appended to a blacklist if not already there – and a flag is raised for the client leading to a further analysis. If the score is below the threshold, the domain name or IP address is considered benign, so it is appended to the whitelist, the harmless container.

Malicious database

The malicious database contains domain names and IP addresses considered malicious by the system. It is filled with data from the fine-grained analyzer, and is queried by the client checker to see if the DB contains specific domains names or IP addresses.

Client checker

When a query is made, a check is done to see whether the client already is flagged. If it is,

the log entry is added to the client's traffic. This makes analysis faster, and helps providing a wider traffic picture when looking at the incident later.

Infected clients database

Once a domain or IP address is considered malicious, it is in addition to being added to the malicious database, added in a database containing possibly infected clients. The clients in question can be viewed through a web-interface or CLI, displaying the IP address of the client, and the traffic observed related to the client. A natural interaction would be to implement and enable the possibility to search in the traffic related to the client, display the information sorted by severity and scoring or by the indicators by flagging it as malicious.

7.2.2 Log files

The log files for this project was gathered in the malware lab located at mnemonic AS[23], and were used with consent. The malware lab includes clients known infected, as well as normal user activity, ensuring a true mixture of legitimate DNS traffic, as well as DNS traffic automatically generated by the variety of Trojan located at the infected clients.

The logfiles are generated by logging the DNS query and answer from DNS. Beneath is a sample of one log entry pulled from a random logfile, and a table defining the different fields.

```
1385456495||195.159.140.196||85.214.157.156||IN||www.facebook.com.||A||
69.171.247.29||60
```

Figure 6 - Sample from logfile

Field output	Definition
1385456495	Timestamp – when the query was made
195.159.140.196	Client IP address
85.214.157.156	DNS server IP address
IN	Class of query
www.facebook.com.	Queried domain
A	Type of query
69.171.247.29	Answer from DNS
60	TTL value

Table 2 - Definition of fields in logfile

7.2.3 Database

The different databases described in the Figure 5 was in ADomDec implemented as tables in one centralized database using MySQL. To achieve the wanted dataflow, the following tables was created (SQL create statements can be found in appendix B, chapter 10):

- Totaldataset
- Replists
- Malicious
- Incident
- Clients

Shown in Figure 7 below, the tables are neatly using foreign keys to reference IDs from other tables when creating incidents.

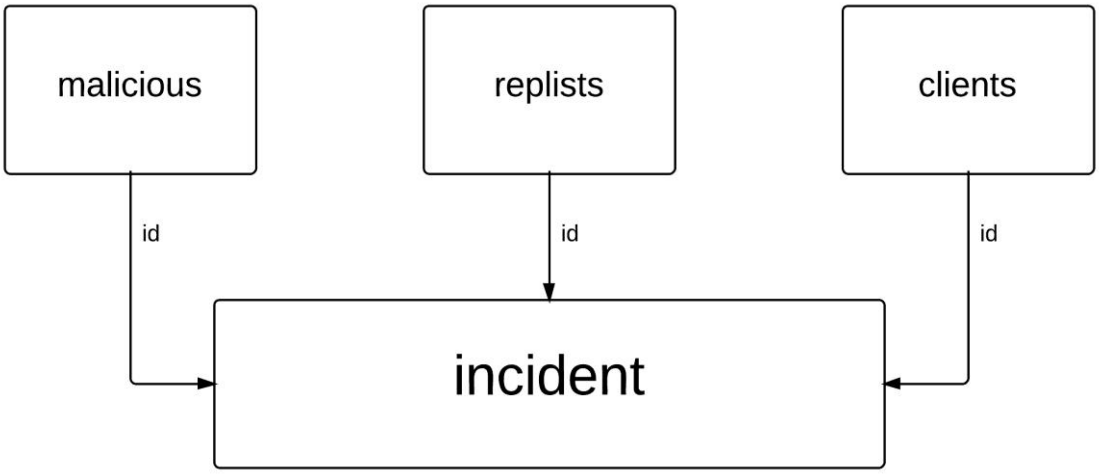

Figure 7 - Use of foreign keys to create incidents

By using the primary key from the malicious, clients and replists tables in addition to other fields of data generated from the analysis the incident can stay a clean and agile table with the flexibility to gather wanted data from the other tables without having to save the same multiple times.

Totaldataset has all the fields found in the log file, in addition to a few other fields containing metadata:

MIN_TTL	The minimal TTL value observed for the recorded set
MAX_TTL	The maximal TTL value observed for the recorded set
AVG_TLL	Average TTL value, based on the two values above and count
FIRST_SEEN	Timestamp first observed
LAST_SEEN	Timestamp last observed
COUNT	Numbers of times the recoded set is observed

Table 3 - Metadata in totaldataset table

During insertion by the use of “duplicate key update”, all fields are updated. Average TTL is calculated on the fly, count added by one, max/min updated if a new value is valid and finally last seen is always updated.

7.2.4 Processing stages

This section describes the analysis stages of ADomDec: how the different features work, their motivation and the basis for the scoring system. How testing was performed, the results, technical challenges along the way and future work is described in Section 8.

The fine-grained analyzer gets the data remaining after the whitelists have sorted away benign domains. The whitelist matching is done using pre-processing to reduce the dataset and workload of the system.

To determine whether the domain name or IP address is malicious or not, the fine-grained analyzer component goes through a series of steps assigning a score along the way. After all the features have been calculated, the score is matched against a threshold. If the score is equal or above, that domain name or IP address will be flagged as malicious, if not, it is considered benign/harmless.

Beneath is a table showing the processing stages from whitelisting to the decision is made. If a domain is whitelisted none of the processing stages are done. Because I had no prior experience with the different features, the scoring system is a result of trial-and-error.

All log data goes through these steps after they have been inserted into the PDNS database.

Feature #	Feature	Score assigned
1	Whitelist	0
2	Blacklist	[1]*
3	Length of word	10
4	Numerical to letter relation	5
5	Only numbers	10
6	Levenshtein distance	15
7	Average TTL value	7
8	Invalid bigram	40
9	Connecting domains	[2]**

Table 4 - List of features in ADomDec

1) Whitelist

The first process is to check whether the domain is whitelisted or not. If the domain is whitelisted, there is no point performing the other operations. This reduces the dataset sent to parsing, speeding up the processing time. The whitelist is collected at alexa.com, containing 1,000,000 domains in descending order of trustworthiness. ADomDec uses the first 500,000 domains in its whitelist.

2) Blacklists

ADomDec has several blacklists, all of which gathered publicly and considered to have a high level of trustworthiness. Some of the lists were used as test data to determine system performance. The lists used as test data are marked in the list below. They are lists of domains with a high certainty of being malicious. Using them as reference when testing gave a conclusive result regarding the coverage rate for each of the fined-grained features were.

[1]* The score varies depending on the trustworthiness of the list.

[2]** The score varies depending on the domains found.

#	List name	Part of test dataset
1	Isc_suspiciousdomains.txt	No
2	Zeus_domains.txt	No
3	Malware_domainlist.txt	No
4	Malware_domains.txt	No
5	Cybercrime_zbox.txt	Yes
6	Virustracker.txt	Yes
7	Cryptolocker.txt	Yes
8	Cybercrime.txt	Yes

Table 5 - List of blacklists used in ADomDec

After the data from the reputation lists had served their purpose as test data, it was implemented as a part of the blacklists.
The findings and results of the system are described in detail in a later section.

All of the domains names and IP addresses both from the blacklist and the whitelist was implemented as a hashmap in ADomDec, having a String as a key and a reputation object as value. When putting a whitelisted domain into the hashmap a Boolean whitelisted is set to true, and opposite for blacklisted domains. When checking if a domain is whitelisted, it checks if the domain exists at all in the hashmap, if it does, and the whitelisted flag is set to true the rest of the analysis is discarded. If the domain exists in the hashmap when checking if it is blacklisted, the score belonging to the list is added to the total score value. The reputation object contains an array of objects allowing several lists to be attached to the same domain. When finding a match, the array is parsed, adding the score of the correct list.

3) Length of word

The longest word found in the Oxford dictionary consists of 45 letters[24, 25], this word is a medical term, and it is safe to assume that this is not plausible as a domain name. Any domain name with a length of >=45 will be assigned a score.

4) **Numerical-to-letter relation**

It is not uncommon for a domain or a subdomain to contain numbers. Google amongst many other has a series of name servers called: ns1.google.com, ns2.google.com, ns3.google.com, etc. – or another structure of name servers indexed with a number. Another example is the well-known online yellow pages/phonebook information service in Norway called, 1881.no. ADomDec calculates the relation between numerical characters and letters, if there are a predominance of numbers to letters a score is added to the total.

This feature is motived from my personal experience working as a security analyst at a security operations center (SOC), having observed thousands of callback/check-in domains and their structure. This traffic often consists of what appears to be a randomized mixture of letters and numbers, with a majority of numbers to letters. The bigram feature described at "8) Bigram" catches the domains randomized without numbers, and the next feature "5) Only numbers" describes detection of domains and subdomains containing exclusively numbers.

5) **Only numbers**

As discussed in the previous feature, domains and subdomains can contain numbers in benign settings. However, many fast flux domains also have subdomains purely consisting of numbers. This feature detects domains or subdomains containing exclusively numbers, but in isolation does not provide sufficient reason to flag the domain in question as malicious, and must be seen in correlation with the other features to provide a useful contribution to the total score. The motivation for this feature is also based upon my personal experience from working at the SOC.

6) **Levenshtein distance**

An edit distance is also calculated, with the use of Levenshtein distance technique[26] with the goal of detecting domain names generated by DGA's (Domain Generating Algorithm) [27, 28]. In 1965 Vladimir Levenshtein considered this technique as the fewest substitutions necessary to morph one with to another, i.e:

Wanting to change from the word "*k*itten" to "*m*itten", the Levenshtein distance would be 1, replacing the 'k' with an 'm'. The Levenshtein distance from "*m*itt*e*n" to "*s*itt*i*ng" would require changing the 'm' and the 'e', in addition to adding a 'g' at the end. Since there are no shorter way than three edits to perform this change, the Levenshtein value is three.

The edit distance will have a lower value for benign domain names than malicious, giving a good indication of possible algorithmically generated domains. Benign services typically name their name servers (NS) ns1.* and ns2.*, giving a small edit distance. Domains generated by a DGA will normally have a greater value.

In effect, this looks for substrings with a higher edit distance value than a given threshold. i.e.:

ns1.google.com → ns2.google.com has a low value, making it benign. While as:

xyzxxcjv.cybercrook.org → uqhfgvnb.cybercrook.org has a high value, making it suspicious.

7) Average TTL value

The average TTL value of the query is calculated, and if the value is within the range [0, 100), a score is added to the total. The value of what to be considered suspicious in the feature is based on research from EXPOSURE[19].

8) Bigram

There are certain letters that never occur in sequence for most western languages. Through research I have not found the sequence "zq" occurring in any language other than in the use as an abbreviation. For ADomDec I modified a list[29] of impossible bigrams to occur in the English language. During the parsing ADomDec splits the domain into substrings for the corresponding domain and subdomains (skipping "www", top level domain (TLD) and double TLDs), checks for illegal/impossible combinations of bigrams and adds a score if it is found.

The bigrams are implemented as a hashmap to make quick lookups of existence. When more than one bigram is found the following code sets the score to be 20 up to two occurrences, after that a score of 10 is added in addition pr hit:

```
if(counter==1) add_to_total_score(20);
else {
      int score=(counter*20/2);
      add_to_total_score(score);
}
```

Figure 8 - Java code for setting score in bigram feature

9) Connecting domains

This feature uses the history of the PDNS DB to gather information for further processing. It targets groups of domains based on the answer from DNS when making the query. Beneath is a fictional example of several domains mapping to the same IP address:

$$\frac{yehajskfor.info}{\frac{uqjoqjrdas.com}{pqluzxyi.org}} \rightarrow 1.2.3.4$$

Figure 9 - Domains related to an IP address

ADomDec uses the answer field from the DNS record to query the PDNS DB for all domains mapping to the exact same IP. These domains are checked if blacklisted, and if they are, half of the initial score of the list is added to the total score in the analysis. This is done to prevent an enormous escalation of events based on related reputation. Answers may have a large set of domains attached, generating false positives if the true score of the list is added.

8 Results, Discussion and Conclusion

8.1 Technical challenges

When developing ADomDec, there were a few technical challenges that had to be dealt with for the system and metrics to function as intended. In the following sections, the different technical challenges and the applied solution are discussed.

8.1.1 TLD and double TLD

The main point of the Levenshtein feature is to calculate the edit distance of two subdomains. However, the TLD can legally consists of two entries, i.e. ".co.uk". Simply splitting the queried domain on '.' will generate queries to the PDNS DB for "*.co".

i.e.) If you keep the second last entry of the query after splitting on '.', this will work for every domain where the length of the split domain array is greater or equal to three – given that the first element is not "www" (since the query have to contain a subdomain as well).

1. "google.com" – second last element: google
2. "something.google.com" – second last element: google

However, the first entry in the list above would return the method, without performing any further analysis since there are not any subdomain beneath google. In the second entry, a query for ".com.google.*" would be performed against the PDNS DB to get other subdomains beneath "google.com" (the query is reversed to make the search quicker in the DB).

When the TLD consists of two elements in the split array this logic will not return the wanted result:

1. "something.google.co.uk" – second last element: co
2. "google.co.uk" – second last element: co

Resulting in queries for ".uk.*". Hardcoding in if-statements to cover all double TLDs would be an inflexible way of solving the problem. In addition, the probability of making a mistake and miss one would be non-negligible. Mozilla has published a list[30] containing all double TLDs possible to use. This list is loaded into a hashmap at startup of ADomDec. If there is a match against the hashmap, ADomDec uses the third last element to get the actual domain (with the same but adjusted check of total length of the query regarding the added length of TLD and "www").

This challenge was relevant both for the Levenshtein and bigram feature. Both of which the system needed to use the domain name in question and not the TLD. As for the bigram feature, the TLD can be considered irrelevant.

8.1.2 Database layout

The layout of the database sets both possibilities and limitations regarding implementation of new features. When starting the development, the system model displayed in Figure 5 - Overview of ADomDec" was followed. This is a system created for a Master's project to get results to prove whether or not it is possible to detect malicious domains using the set of features described, and to flag clients as possibly infected. For this purpose, the system gave very good results.

It would also have been desirable to analyze in detail how well the different features detect malicious domains, on what form of domains and how well they work all together. All main components are there, but they are tighter connected than wanted in a full-scale operational system. Thoughts of how the system can be extended to the full and several new features are discussed and described in Section: 8.4 "Future work, conclusion and recommendations".

Reversed Query

At the very start, the query was logged in pure form. When querying the total dataset table with the Levenshtein distance feature, it started as “*.uggabugga.ru.”. Doing so, makes the DB search through the entire table, taking a considerable amount of time rather than reversing the domain before executing the query. By setting the index on the reversed query, the efficiency gain was significant. This enables querying “.ru.uggabugga.*”, letting the DB get much deeper into the search-tree before having to look for everything, thereby speeding up the search even more.

8.2 Results

To measure whether ADomDec is able to detect malicious domains it is tested against a testing set containing exclusively malicious domains of various types. The test set was obviously not loaded into the system as blacklist entries before testing.

8.2.1 Preparation

PDNS DB

In order to have a rich dataset in the PDNS DB, log data was used to fill the database. During the preparation phase, the data used as test set was kept aside. A total amount of 14 683 250 logfile entries were inserted into the DB (see Figure 10). Out of these, there were 4 906 423 unique entries (see Figure 11) (criteria for uniqueness can be found in the SQL create statement in Section 10: “Appendix B: SQL create statements”, “Figure 15 – Appendix B: SQL create statement for creation of totaldataset”). Out of almost 5 million unique entries, 2 725 549 were without collision (see Figure 12) and the remaining 2 180 874 had one or two counts or more (see Figure 13). This provides the foundation of the PDNS DB, and functions as DNS history when the test set was used.

```
mysql> select sum(count) from
totaldataset;
+------------+
| sum(count) |
+------------+
|   14683250 |
+------------+
```

Figure 10 - Total amount of log data in PDNS DB

```
mysql> select count(id) from
totaldataset;
+-----------+
| count(id) |
+-----------+
|   4906423 |
+-----------+
```

Figure 11 - Number of unique entries in PDNS DB

```
mysql> select count(id) from
totaldataset where count=1;
+-----------+
| count(id) |
+-----------+
|   2725549 |
+-----------+
```

Figure 12 – Number of true unique entries (only seen once)

```
mysql> select count(id) from
totaldataset where count>=2;
+-----------+
| count(id) |
+-----------+
|   2180874 |
+-----------+
```

Figure 13 - Number of entries with count>=2

Test set

ADomDec was tested with three test sets. This section describes how the test sets were constructed and what purpose they serve.

The first test set was made to measure the baseline and effectiveness of the system; how much is it possible to detect? The test was constructed by concatenating several blacklists containing known malicious domains. As listed in Table 5 - List of blacklists used in ADomDec" blacklist 5, 6, 7 and 8 were used. The domains were unknown to ADomDec before generating the test file, ensuring that the domains were not selected to perform well during testing. The test set was made by a python script that generated a logfile on the same format as described in 7.2.2 Log files. The python script can be found in chapter 11 Appendix C: Python script for test data, Figure 20 - Python code to generate logfile for test data. The test set contains 10918 unique logfile entries carrying exclusively malicious domains.

The second test set was a mixture of 7500 benign domains, in addition to the same malicious domains from the first test set. This test set was made to measure the amount of false positives and false negatives from the first test set.

In addition, a third test set containing 1000 entries was sent through ADomDec. All domains were manually analyzed before the test, ensuring a complete white-box test. The file was a randomly selected logfile used to assess the false-positive (FP) and false-negative (FN) rate on a mixed test set.

This was then launched into ADomDec with the history of PDNS described earlier. How each feature was tested and the statistics gathered are described in the next section, 8.2.2 Testing.

8.2.2 Testing & Findings

The basis for the test has briefly been described in earlier sections. 14 683 250 logfile entries were inserted into the PDNS DB making a small set of DNS history. Note that this is a subset of data, recorded over a period of approximately two months. The first test set consists of

10918 malicious log entries considered to have an extremely high probability of being malicious. The testing of ADomDec was performed in three stages:

Nr. 1) Discover the maximum coverage, the highest amount of entries possible to detect with the given features.

Nr. 2) Assess the false positive (FP) rate when adding benign data into the test set.

Nr. 3) 1000 entries pulled from a random logfile - manually analyzed to discover a real false negative rate (although on a very small data set).

This section first describes the findings for test set nr. 1, how well the different features scored, presented in a statistical manner – sorted by feature name, ending with a statistical summarizing table. Then the same will be done for test set nr. 2, including the FP-rate after testing with a combined test set of the same malicious domains used in test set nr. 1, concatenated with a series of benign domains. Finally, the findings of test set nr. 3 is presented. The discussion of the results is presented in 8.3 Discussion.

8.2.2.1 Total detection (test set nr. 1)

Total results

ADomDec detected 8450 domains, out of the total 10918 domains. This gives a detection rate of 77,39% under ideal circumstances.

This result was produced with a combination of all features. Single features and multiple features contributed to the score. This result sets the baseline for the potential of the system.

Whitelist

Of the 10918 domains entering the system, 8 were whitelisted giving 0,07%, showing that either the test set is a good representation of malicious domains, or that the whitelist is weak. Given the combination of trustworthiness between the whitelist and blacklists making the foundation of the test set. This result proves that the test set was representative.

Blacklists / Reputation

ADomDec discovered 509 domains based on reputation, a quite small percentage of 4,66% of the total data (out of the 10918 domains), and 6,02% of the detected domains. This is tightly related to the quality and quantity of the reputation lists used by the system. Although the malicious domains used for the test had their origin from blacklists, the domains were not previously known or based upon already implemented blacklists. Had the test set data been present in the blacklists, the detection rate should be close to 100% based on reputation alone. This will be discussed further in chapter 8.3 Discussion.

Invalid bigrams

The illegal bigram feature detected 7906 domains, a total of 72,4% of the total parsed dataset and 93,56% of the total amount domains detected. This makes a clear majority of the detection, and could be related to the structure of the domains in the test set. If the domains in the test set were related to Trojans making use of domain generating algorithms, one could expect to see many occurrences of illegal bigrams. However, the structure of the domains in the test set were not known prior to the testing.

Length of word

This feature did not generate one single hit from the test set. The low trigger rate could be a consequence of the test set used, and does not necessarily indicate the feature is completely irrelevant. Measuring the false positives (FP) is difficult without having any hits. Either this feature has a low FP rate, or it does not provide a sufficient contribution to the analysis and can be removed.

Most of the legitimate domain names used are far from 45 letters long. The scoring for this feature could be adjusted to trigger at an earlier stage and increasingly add score as the length of the word increases.

Numerical-to-letter relation

This feature triggered in 2 of the 8450 detected domains. In the entire test set containing

10918 domains, 257 of them contained one or more numbers. Out of the 8450 domains detected only 111 domains contained numbers. Out of the total test set (10918) the percentage detection was 0,018%, and in the domains detected (8450) it was 0,023%. Out of the 111 domains that contained one or more digits it detected 1,8%.

This feature alone does not provide enough score to flag the domain as malicious. Hence, the domains detected with this feature were also triggered by other features allowing the score to get high enough for the domain to be flagged.

Only numbers

This feature detected exactly the same amount as the numerical relation feature. Hence, the calculations of percentage is the same.

Levenshtein distance

Out of the total test set, the Levenshtein distance feature 48 domains, giving a total of 0,25% and 0,56% in the set of detected domains. However, for the Levenshtein distance to even perform a calculation of the query it must contain a subdomain. There were 225 domains containing subdomains, giving this feature a detection rate of 21,3%.

Connecting domains

This feature did not generate one single hit from the test set.

When generating the test set, the resolved IP to the queried domain was sat to a static IP address that does not exist in the PDNS DB. When this feature looked in the DB for connecting domains, the result came up empty every time.

The python script that generated the logfile set fictitious data for timestamp, client IP (generated by random numbers, ensuring only different clients), DNS IP, class, type, resolved IP and TTL.

Some of the domains used does not resolve to an actual IP anymore, attempting to map up the existing domains to IP addresses could have been done. However, this would reduce the

test set radically. I chose not to do this because the importance does not lie in whether or not the domain actually lives; it is whether or not ADomDec will detect the structure of the domain classifying it as malicious or benign.

Average TTL

This was unmeasurable.

The measure of this feature was ruined when generating the test set. Since the test data is generated based on blacklists, there are no true metadata for the queries. By setting the TTL to 3600 in the python script, there is no way for this feature to measure real data. The motivation of setting the TTL to a static value is the same as described for the previous feature, “Connecting domains”. This is discussed further in 8.3.3 “Other features”.

Multiple triggers

Some domains were detected by more than one feature. In total 104 of the 8450 domains were detected simultaneously of several features, giving a detection rate of 1,2%.

Summarizing table of features

Feature name	# of detected domains	% of 10918	% of 8450	Trigger base	% of the trigger base
Whitelist	8	0,07	-	-	-
Blacklist	509	4,66	6,02	-	-
Invalid Bigram	7906	72,4	93,56	-	-
Length of Word	0	0	0	-	-
Numerical to Letter Relation	2	0,018	0,023	111	1,8
Only Numbers	2	0,018	0,023	111	1,8
Levenshtein Distance	48	0,25	0,56	225	21,3
Connecting Domains	0	0	0	-	-
Multiple Triggers	104	-	1,2	8450	1,2

Table 6 - Performance summarization of features

8.2.2.2 Detection including benign domains (test set nr. 2)

ADomDec was also tested with a test set consisting of all the domains from the previous test set, but extended with 7500 benign domains. This was performed to detect the false positive (FP) rate, and false negative (FN) rate. Previous results gave a baseline; how many domains is it possible to detect under ideal conditions; how much *can* be seen? Tuning the different features correctly has a significant impact on the results presented below. Tuning and scoring will be discusses in a wider aspect in the next chapter. The extended test set was created with the first 7500 domains from the Alexa whitelist. When the analysis was performed, the entire whitelisting feature was deactivated. Otherwise, all the domains different from the first test set would be whitelisted and no analysis would have been performed.

Results from Features

Since whitelisting was turned off, no domains were whitelisted. It is safe to assume that the same eight domains that were whitelisted when parsing the first test set would retrigger when running the extended test set. Since the extended data was pulled from a whitelist, having whitelisting enabled would undermine all other results and purpose for running the extended test set.

As for the rest of the features: Blacklist/reputation, invalid bigrams, length of word, numerical to letter relation, only numbers, Levenshtein distance, connecting domains, average TTL and multiple triggers – a total of 90 ‘new’ domains were detected with the extended test set, making the total FP rate with this dataset at 0,48%.

8.2.2.3 “Random” test data (test set nr. 3)

To measure the FP and FN rate on a mixed test set of true data, a random logfile was selected and 1000 entries were sent through the system. All the domains were manually analyzed before the testing started to see how many of the 1000 domains that were malicious. The expected result was one malicious domain.

After analyzing the domains, 17 domains were considered malicious, giving a percentage of 1,7%. The distribution of the domains found over the different features are listed below.

Feature name	# of domains
Whitelist	**776**
Blacklist / Reputation	**0**
Invalid bigrams	**13**
Length of word	**0**
Numerical to letter relation	**2**
Only numbers	**3**
Levenshtein distance	**7**
Connecting domains	**3**
Average TTL	**0**
Multiple triggers	**10**

Table 7 - Features and number of found domains

The domains detected was mainly detected by bigrams, which is to some degree related to the differences between the English language and the Nordic. For instance the bigram “kv” is according to the list illegal – but there are words in the Norwegian language with the sequence “kv” that are perfectly legal. The remaining domains detected was flagged with numerical features, which appears to be automatically generated DNS queries related to streaming services.

8.3 Discussion

In this section, the findings presented above are discussed. The interesting points are the correctness of the system, whether or not the data was tested in a fashion that reveals appropriate results, and why/how the features achieved according to the results previously shown.

8.3.1 Test sets

Total Detection

The total detection test set was based on blacklists. The nature of the blacklists has an impact regarding the structure of the domains. If the blacklists have domains all belonging to one Trojan family, one could expect a vast majority of the domains to have the exact same structure. This leads to a weakness in this way of measuring. If the system detects one of these domains, it is likely it will detect the rest – and opposite. The blacklists were however a mixture of domains belonging to several different Trojans, proven by the fact that out of the 10918 domains analyzed, 8450 of them were detected by ADomDec.

Extended Test Set

The extended test set was a combination of the domains from the total detection set, and the 7500 first domains from the Alexa whitelist. This measure makes sense as long as the whitelisting feature is deactivated. All the domains detected from the first test set was rediscovered when analyzing the extended logfile, in addition 90 other domains were detected. This indicates an FP rate at 0,488%.

The other blacklists used during the testing contains known malicious domains. When parsing through domains from a whitelist, naturally none of them triggers unless there is a FP in one of the lists. During this test, none of the 90 domains was flagged with reputation. This result is only an indicator of the correctness in the reputation lists.

Mixed Dataset

1000 logfile entries from a randomly selected logfile were analyzed by ADomDec to detect the FP and FN rates. I manually analyzed all of the 1000 domains in order to establish a certain baseline. Out of the domains, a clear majority was benign, containing many duplicate entries. I detected manually one entry, which was malicious and should be flagged by ADomDec. Since the logfile was selected at random it is impossible to know what kind of data the file contains. Depending on which list is selected, it is impossible to know whether it is a file containing one or 25 (example numbers to illustrate the difference of occurrences regarding malicious domains) malicious domains and how many of the 1000 domains that are whitelisted, without knowing the content of the file before choosing.

8.3.2 Reputation lists

Whitelist

When the cybercriminals develop their exploit kits (EKs), it is reasonable to assume that some of the more advanced organizations build their command and control (C&C) domains regarding the existing whitelists available online. The fast fluxing EKs seem to neglect this as a part of their development routines, which is natural regarding the nature of fast fluxing. By changing the domain names and IP addresses with a high frequency – the domain is active for such a short period of time, that it does not really matter. By the time white-hat organizations have blacklisted the domain, they are no longer using it as a part of their network. The cybercriminals could of course use Markovs chain to generate English looking words to avoid statistical features such as illegal bigrams and to some degree make sure to achieve a plausible Levenshtein distance score.

Regarding the detection coverage of domains being whitelisted by ADomDec, this is dependent on the whitelist implemented. For ADomDec the Alexa whitelist was used. The list is generated in a sorted order of the most used domains on the Internet with google.com, facebook.com and youtube.com placed as top three. The list contains 1 000 000 domains, but ADomDec only uses the first 500 000. This was done because the more domains implemented, the confidence level of the domain actually being benign is lowered.

Blacklist / Reputation

Many of the same discussion topics are valid when discussing blacklists and whitelists. The danger of having too many domains in the whitelist, is to whitelist and assume a domain is benign based on external analysis. Regarding blacklists generated by an external source, one has little knowledge of the reason for the domain or IP to be blacklisted. Detecting domains based on blacklist therefor depends on the information exchange agreement with other vendors, and the credibility of the lists gathered from public sources.

In ADomDec, a rather small set of domains (relative to the amount which is possible to gather) is used by the system. A total of 76800 domains and IP addresses are loaded into ADomDec at startup, with a clear majority of domains. This makes the number of detected domains by blacklist an example and proof of concept (POC) of the impact of having reliable blacklists implemented in the system, and what they are capable of detecting. The reputation hits found when parsing the test set is determined by the quality of the reputation lists used in the blacklist feature section of the system.

Collisions with blacklists and whitelists

When automatically updating black- and whitelists a new challenge arises. If there are collisions between the lists, so a domain exists in both lists, a malicious domain could be whitelisted generating a FN or a benign domain could be blacklisted generating a FN. This is a general problem when having both white and blacklists, but when leaving the updating of the lists to the system there is a greater danger of losing control.

ADomDec is created to trust the whitelist, giving the whitelist precedence. If a domain is whitelisted, analysis will never be performed. The domains that exist in the whitelist are less likely to change compared to domains in the blacklist. New, malicious domains are often discovered when existing exploit kits are updated, or new ones emerge. Therefore, implementing an automated mechanism to handle black- and whitelists would be more beneficial.

8.3.3 Other features

Invalid bigrams

During ideal circumstances this feature detected the majority among the domains considered malicious. This shows that either the domains were poorly balanced, or this is a good feature. If the domains used in the total detection test set all had their origin from the same exploit kit (EK), they would follow the same structure. If ADomDec detects the general form, a vast majority will be detected – if the structure is unknown to the system it will pass through undetected.

The test set was, however, constructed of domains from different EKs and different sources making the findings a plausible representation of the total detection rate of the system. Detecting invalid bigrams proves to be a sufficient feature for detecting domains generated by a domain generating algorithms (DGA). As mentioned earlier, cybercriminals could make use of Markovs chain or similar algorithms to generate domain names that have the same structure as a valid English word without actually being one. This could bypass this feature, leaving the detection to other features; mainly blacklists and possibly Levenshtein distance.

Length of word

ADomDec detects domain names with a length above 45 letters. This is based on the longest existing word in the Oxford dictionary – a medical term. This feature will mainly detect domain names made by a weak DGA. The testing revealed this feature as rather irrelevant, based on a low trigger rate and little detection in all the tests. Having features that trigger seldom does not necessarily mean that the feature is a total waste. However, if the number of features with a low trigger rate is high it could affect the total time to perform analysis in a negative way if the computational cost is greater than the profit from the analysis. As mentioned earlier, this feature could be adjusted to trigger at an earlier stage and progressively increase the score based upon the numbers of letters above the placed limit.

Numerical to letter relation and only numbers

These features prove to have a low trigger rate, but without generating FPs. As discussed in

the previous section, having features that do not trigger often is acceptable with today's equipment and hardware prices. There are benign domains containing numbers, but because of the score set for these features, simply having a majority of numbers or exclusively numbers in the domain name is not enough to flag the domain as malicious. These features plays a positive role in correlation with other features, pulling the score up – making the multiple trigger feature work and supporting the foundation of the scoring system (these features are discussed in detail at the end of this section).

Levenshtein distance

The Levenshtein distance feature detected a minority of the domains, but seen in relation to what is has the ability to detect the overall value of the feature is considerable. The Levenshtein distance is only calculated for domains having a subdomain, so it cannot be calculated for all domains. As seen in the total test set, only 225 of the 10918 domains contained a subdomain triggering analysis. Out of these 225, 48 had an edit distance (ED) above the selected threshold.

Since the ED is computed for all existing subdomains ever existed for the TLD, this brings a source to FPs with the current design of the feature. If a TLD once in history ever got hijacked and hosted malicious code or was a part of a CC infrastructure, this is present in the totaldataset. When ADomDec searches the totaldataset, this domain is returned. When the ED is calculated, a high score arises since the rest of the domains are benign. Unless the scoring is neatly tuned, this could easily generate a FP.

Connecting domains

During analysis historical domains mapping to the same IP address are gathered from the totaldataset. In the tested version of ADomDec, only reputation against blacklists are performed for the returned domains. If any of the domains are found in the blacklists, half of the initial score is added to the total score. IP addresses can have a considerable amount of domain names attached, if a full score is added in addition to escalation of score depending on the number of lists it is found in, this could be a serious source of FPs.

Average TTL

During the testing of ADomDec the generation of the test set logfile ruined the basis for the average TTL component. The TTL was hardcoded in the python script because only domain names were available. Many of the domains in the test set were no longer active, so generating true DNS requests would reduce the dataset considerably. The nature of the test was the structure of the domain, and giving precedence of keeping quantity of the test set, this decision was made. Similarly to the numerical to letters relation and the only numbers feature, this feature has an extremely low score (compared to features such as invalid bigrams), so this feature is mostly interesting when all features have been processed and ADomDec looks at the number of features triggered and the total accumulated score. This feature alone is never enough to flag a domain as malicious.

When tested with the other test sets, this feature gives a rather low trigger rate, although the domains triggered showed a low FP rate. The feature was implemented based on the research performed in EXPOSURE[19].

When testing this and the previously described feature (“Connecting domains”) with the first test set the resolved IP address and TTL value was hardcoded. This motivation for this was to prevent a reduction of the test set because many of the domains are no longer active. By using existing public passive DNS databases one could possibly find resolved IP addresses and TTL values. This would make it possible to test this feature in a better way.

Multiple triggers

Before checking the total score against the threshold, ADomDec looks at the number of features triggered and adds a small escalation score. Simultaneous triggering by multiple features indicates higher certainty of maliciousness. Features having a small score, are therefore valuable in correlation with the other features, but can also generate FPs if only the minor features triggers and the domains are flagged on a weak basis.

Tuning, scoring and thresholds

The scoring was initially based on guessing and my personal experience from working as a security analyst at a security operation center (SOC). When performing the tests, there were minor changes in the scoring. More testing with different test sets and increased experience would make it possible to tune the scoring system towards an ideal combination based on the feature set.

Motivation and scoring

Some of the features were inspired from previous work, other were inspired by my personal experience from working as a security analyst at a security operations center. The representation of trigger rates amongst the features have a tight connection with the scoring system. If a feature has a low score, it has to trigger along with at least one other feature. This can lead to a low trigger rate based on score, not detection. However, to keep the FP rate as low as possible, not all features can have the same score.

8.4 Future work, conclusion and recommendations

8.4.1 Future work

This section describes different features that could have a positive contribution in the analysis, technical challenges that could arise and why the suggested feature would benefit ADomDec.

Jaccard index

When comparing two strings, ADomDec uses Levenshtein distance. This could be expanded to also calculating the Jaccard index [27, 31]. The Jaccard index is calculated from the raw differences between two strings based on the letters as sets. Two identical strings give a Jaccard index of 1, which is the same result as two sets containing the same letters. Whereas Levenshtein Distance generates a score based on the placement of the letters, the Jaccard index generates a score based on the existence of letters only.

Examples:

String x = “abcde”
string y = “ecdab “

The Jaccard index is 1 between both ‘x’ and ‘y’ and between ‘x’ and ‘x’, i.e. Jaccard(x,y) = Jaccard(x,x) = 1.

The Levenshtein distance is 4 between string ‘x’ and ‘y’, and 0 between ‘x’ and ‘x’, i.e. Levenshtein(x,y) = 4 and Levenshtein(x,x) = 0.

Invalid bigrams for different languages

The system could have different illegal bigrams lists for different languages, and match the domain against the correct list based upon TLD. This would eliminate the linguistically differences, and provide a more accurate result.

Length of word

This existing feature could be altered to trigger at a short length of word, however with a lower score. The score should increase progressively as the length increases.

Testing of average TTL and connecting domains

To test this features and the effectiveness, test set nr1 should be matched against existing public passive DNS databases, in addition to reverse DNS lookups to find the resolved IP address and TTL value of the domains.

Caching

To optimize the system, caching should be implemented. By caching the 100 000 most used searches using for instance Ehcache[32], the system would not have to query the database for each domain regarding the connecting domains and Levenshtein distance feature. This could shorten the analysis time by a great deal, if the system were to be implemented in a larger scale network.

Client checker

In the tested version data is not added to a client unless it is considered malicious. This is not like initially intended where once a client is considered possibly compromised, all later traffic is associated with the client. This was done because the data can be extracted from the total dataset table when querying from the webUI. This will raise the technical question regarding the aspect of time, and amount of associated traffic. Not all traffic related to the client may be of interest. However, having good queries sorting on time, and relevance in addition to a user-friendly interface displaying the information, this problem is avoidable.

WebUI

An interface displaying the infected clients and the traffic related was not included in the implementation, making the system a pure proof-of-concept system. For the system to have the wanted function, a webUI or another instance of interface interacting with the user/admin of the system must be implemented. Nevertheless, accessing the DB directly via the mysqladmin shell was adequate to extract the data proving the concept of the ADomDec system.

The webUI should have functionality such as:

- view currently infected clients
- query previous infected clients and the traffic related
- search for specific traffic patterns within the traffic of the client (domains, part of domains and IP addresses)
- display infected clients based on domains or IP addresses
- Flag clients as handled, removing them from the view
- Add comments to the traffic presented
- Get notifications if a new associated event related to the client occurs

Ecosystem with other systems

By setting other systems such as EXPOSURE and FluxBuster in parallel using the decision made from the external systems as input into the analysis in ADomDec, an ecosystem can be

created. This would create a better data foundation for making a decision whether or not a domain is considered malicious or not.

Connecting domains and performing more analysis

The current implementation of ADomDec only performs blacklist checking against related domains to the resolved IP address. More analysis could be performed to decide if the connected domains are malicious or not. This again relates to the general weakness of using related domains actively in the analysis – if a benign domain once hosted a malicious domain it would trigger this feature. Using this as a feature will require good tuning. The system could gather a total score for all the connecting domains and see the score in correlation with the amount of domains before matching it against a threshold.

Whois information

Malicious domains often have a recently new creation date, since they tend to be changed with a high frequency. This could be revealed by looking at whois information. However, doing this removes ADomDec from being a system in the totally stealth category to a semi-stealth system because the queries would generate outbound traffic that could be observed.

Increased modularity

If the system should be expanded further and placed in a system for live analysis, an increased form of modularity should be implemented. By making the different parts of the systems as modules and placing a message queue system to handle communication between the modules, it will be easier to insert new components and monitor the traffic between the components. This would make it easier to get an overview if parts of the systems stopped, to monitor and make graphs over the amount of traffic passing through the system, and would function as an extra insurance against data loss (if a component stops the messages would be in the queue when the components re-starts).

Automatically collect and update white/blacklists

Having an automated process to gather and update the white- and blacklists is not implemented in the tested version of ADomDec, however it would be beneficial. Automatically updating the black- and whitelists would reduce FPs and ensure an even better data basis for the analysis. Having a crontab taking care of the updating of the lists would also relieve the administrator of workload, although it brings the challenges discussed in the previous chapter.

Detecting compromised DNS servers

Since ADomDec silently looks at DNS data over time, applying additional algorithms and statistical features to detect abuse of DNS could be done. This would not have a direct effect on the system – but notifying those responsible for the infected DNS server would be beneficial for everybody.

8.4.2 Conclusion

In the previous chapters, ADomDec has been presented as a complete system to automatically detect malicious domains based on different features targeting the characteristics of the domain names. This chapter discusses and concludes whether ADomDec was successful implemented and answered the research questions from chapter 2.

False Positives and False Negatives

In terms of automatically detecting malicious domains, ADomDec proved that it possible to detect only the majority of malicious domains without generating enormous amounts of FP's. As presented in 8.2 Results, the detection rate and FP rate varied depending on the test set used as input. However, the detection rate was 75%+ and the FP rate was 2%-. Further research and development will tell whether it is possible to increase the detection rate while keeping the FP rate stable.

The graph below shows the relationship between detection of data in correlation with FP and FN rates.

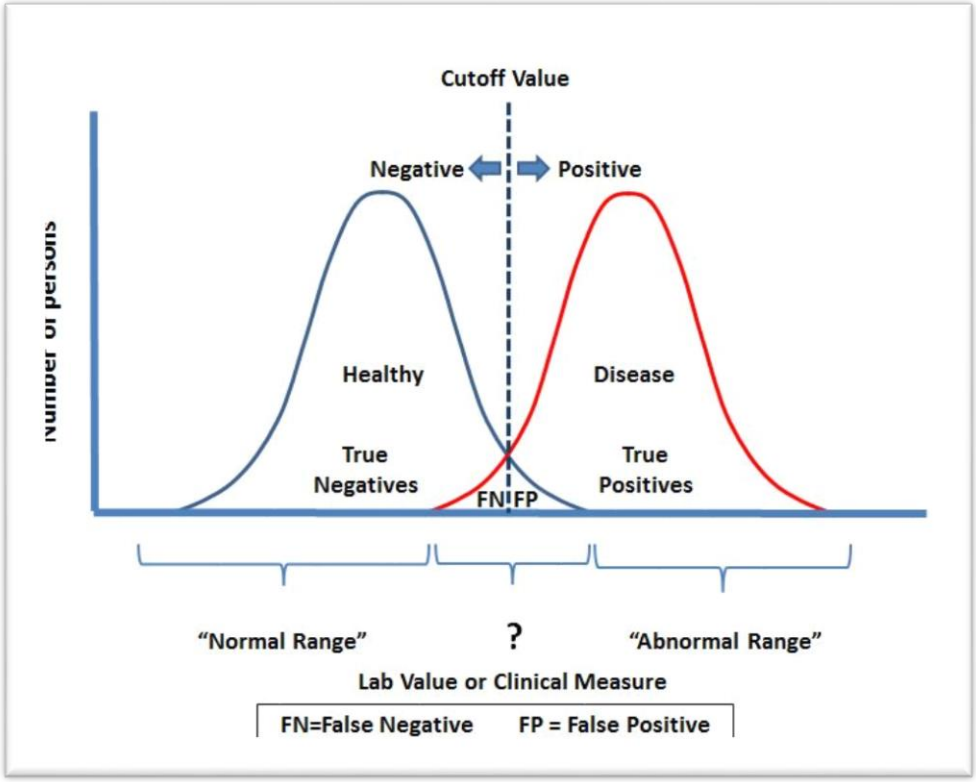

Figure 14 - FP/FN rate[33]

The graph is an illustration of the amount of detected domains, and the amount of false positives (FP) and false negatives (FN). Having a low FP will imply a higher FN, and having a low FN will increase the FN. Setting ADomDec against this data model the cutoff value will change according to tuning of the scoring system.

Scoring

The scoring system was developed with a trial-and-error method because no data existed with regard to the different features, the trigger rates and their effectiveness. During development and testing, the scoring was adjusted towards the ideal phase of the graph illustrated above. Whether or not the scoring is optimal, and the improvement gained along

the way is hard to know. Further development, testing and adjusting is needed to determine the optimal settings for the scoring system in order to minimize FP and FN rates.

Traffic detection

The origin of the flagged traffic made by the clients is hard to identify. Since DNS does not include URI, only host – it is impossible to know for sure if the traffic observed is related to the early stages of an infection, or if the client is compromised. The reputation lists are often sorted in categories, defining domain names as callback/checkin domains or related to downloading of exploit kits, etc. When a client is flagged based upon the traffic analyzed by ADomDec, the client should be investigated further. For future work, an additional expansion of the flagging system can be implemented. It could be useful to assess the range of certainty if a client has been compromised or not.

Q1) How to design a practical system for detecting malicious domains and infected clients based on monitoring DNS queries and answers?

A1) ADomDec proved it possible to detect malicious domains and the clients responsible for the traffic by monitoring DNS queries and answers. By applying a set of technical metrics analyzing the domain name, combined with reputation ADomDec proved to be a qualified system design.

Q2) What is a realistic detection rate of malicious domains detected with this system?

A2) The detection rate is plausible and realistic regarding the function of the system. As discussed previously in the chapter, the detection rate was 77,39%. For a prototype system, this is satisfactory. As further development will increase the detection rate even higher, ADomDec can become a valuable asset regarding malicious domain detection.

Q3) What is the nature of detected malicious domains?

A3) The majority of the detected domains contained illegal bigrams, implying they were generated by a DGA. Some of the malicious domain names could consist of legal words, appearing to be benign, however they can also be malicious because the domain is compromised or because the cybercriminals try to appear legit. Such domains will (if not found in a blacklist), go undetected through ADomDec.

When I started working on this Master's project, I had initial ideas that to some degree had been implemented. Throughout the research phase some of the ideas proved wrong and others out of scope. Some remained, although they did not detect as much as intended. The illegal bigram feature proved to detect a vast majority. In retrospect, lists of illegal bigrams for different languages should have been implemented, and as the feature triggers the language list should be displayed. I believe that with some further development, ADomDec can be a valuable asset if placed in a computer network.

In conclusion, ADomDec demonstrates the capability to detect malicious domains, and to find the clients responsible for the traffic. The outcome of the Master's project is very promising. Implementing ADomDec in a computer network provides an additional element for security incident detection, and for identifying infected clients. It provides valuable data, and strengthens and supplements existing of security detection mechanisms.

9 Appendix A: Glossary of Terms and Acronyms

- DNS: Domain name system – see [34] for a complete list of RFC related to DNS.
- RFC: Request of comments – A defined internet standard[35].
- PDNS: Passive domain name system
- ADNS: Active domain name system
- DB: Database
- CC: Checkin
- FP: False positive
- IDS: Intrusion detection system
- IPS: Intrusion prevention system
- DGA: Domain generating algorithm
- LDS: Labeled dataset
- CDN: Content delivery network
- FW: Firewall
- CLI: Command line interface
- SOC: Security operation center
- EK: Exploit kit
- POC: Proof of concept
- ED: Edit distance
- TLP: Traffic light protocol
- NM: Name server
- TLD: Top level domain

10 Appendix B: SQL create statements

```
use pdns;
CREATE TABLE IF NOT EXISTS totaldataset (
id                      BIGINT(64)      NOT NULL AUTO_INCREMENT,
Time                    VARCHAR(32)     NOT NULL,
Source_IP               VARCHAR(21)     NOT NULL,
DNS_IP                  VARCHAR(21)     NOT NULL,
Class                   VARCHAR(10)     NOT NULL,
Query                   VARCHAR(256)    NOT NULL,
Reversed_Query          VARCHAR(256)    NOT NULL,
Type                    VARCHAR(10)     NOT NULL,
Answer                  VARCHAR(256)    NOT NULL,
Min_TTL                 INT(10)         NOT NULL,
Max_TTL                 INT(10)         NOT NULL,
Avg_TTL                 DOUBLE          NOT NULL,
First_seen              VARCHAR(128),
Last_seen               VARCHAR(128),
Count                   BIGINT(64),
UNIQUE (Class, Type, Query, Answer),
INDEX totaldataset_query (Query),
INDEX totaldataset_Reversed_query (Reversed_Query),
INDEX totaldataset_answer (Answer),
PRIMARY KEY (id)
);
```

Figure 15 – Appendix B: SQL create statement for creation of totaldataset

```
use pdns;
CREATE TABLE IF NOT EXISTS clients (
id              BIGINT(22)      NOT NULL AUTO_INCREMENT,
ip              VARCHAR(21)     NOT NULL,
First_seen      VARCHAR(128)    NOT NULL,
Last_seen       VARCHAR(128)    NOT NULL,
UNIQUE(ip),
PRIMARY KEY (id)
);
```

Figure 16 - Appendix B: SQL create statement for creation of clients

```
use pdns;
CREATE TABLE IF NOT EXISTS malicious (
id                  BIGINT(22)          NOT NULL AUTO_INCREMENT,
DNS_IP              VARCHAR(21)         NOT NULL,
Class               VARCHAR(10)         NOT NULL,
Query               VARCHAR(256)        NOT NULL,
Type                VARCHAR(10)         NOT NULL,
Answer              VARCHAR(256)         NOT NULL,
Min_TTL             INT(10)             NOT NULL,
Max_TTL             INT(10)             NOT NULL,
Avg_TTL             DOUBLE              NOT NULL,
First_seen          VARCHAR(128),
Last_seen           VARCHAR(128),
Count               BIGINT(22),
INDEX malicious_query (Query),
UNIQUE (Class, Type, Query, Answer),
PRIMARY KEY (id)
);
```

Figure 17 - Appendix B: SQL create statement for creation of malicious

```
use pdns;
CREATE TABLE IF NOT EXISTS replists (
id                  BIGINT(22)          NOT NULL,
list_name           VARCHAR(128)        NOT NULL,
last_updated        VARCHAR(32)         NOT NULL,
UNIQUE (id),
PRIMARY KEY (id)
);
```

Figure 18 - Appendix B: SQL create statement for creation of replists

```
use pdns;
CREATE TABLE IF NOT EXISTS incident (
id                  BIGINT(64)          NOT NULL AUTO_INCREMENT,
mal_id              BIGINT(64)          NOT NULL,
cli_id              BIGINT(64)          NOT NULL,
replist_id          BIGINT(64)          ,
score               BIGINT(64)          NOT NULL,
reason              VARCHAR(500)        NOT NULL,
created             VARCHAR(32)         NOT NULL,
last_updated        VARCHAR(32)         NOT NULL,
status              INT(2)              NOT NULL,
count               BIGINT(64)          NOT NULL,
PRIMARY KEY (id),
foreign key (mal_id) references malicious (id),
foreign key (cli_id) references clients(id)
);
```

Figure 19 - Appendix B: SQL create statement for creation of incident

11 Appendix C: Python script for test data

```
import random
input_file = open("testset_raw.txt", 'r')
output_file = open("generated_loggdata.txt", 'w')
all_domains = []
clients = []

for line in input_file:
    if line not in all_domains:
        client_ip = ""
        while 1:
            client_ip = "%d.%d.%d.%d" % (random.randint(10,200), random.randint(10,200),
random.randint(10,200), random.randint(10,200))
            if client_ip not in clients:
                clients.append(client_ip)
                break
        loggentry = "1400328000||%s||123.45.67.89||IN||%s.||A||666.66.666.66||3600\n"
% (client_ip, line[:-2])
        output_file.write(loggentry)
        all_domains.append(line)
```

Figure 20 - Python code to generate logfile for test data

12 References

[1] B. K. Vijay Vaishnavi, "Design Science Research in Information Systems," p. 45, 2013.

[2] I. Software, "Understanding DNS (the Domain Name System)," 2007.

[3] D. N. S. D. Parameters. (2013.11.18). Available: http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml

[4] B. Hally. (2008, 03.11). *How DNS cache poisoning works*. Available: http://www.networkworld.com/news/tech/2008/102008-tech-update.html?page=1

[5] M. Kaeo, "Passive DNS. A Tool that is Making a Difference in Tracking Down Criminal Activity On The Internet," p. 8.

[6] A. Lev. (2010). *DNS blocklists and reputation services, part1* [Scientific]. Available: http://blogs.computerworld.com/15264/dns_blocklists_and_reputation_services_part_1

[7] A. Lev. (2010). *DNS blocklists and reputation services, part2* [Scientific]. Available: http://blogs.computerworld.com/15272/dns_blocklists_and_reputation_services_part_2

[8] A. Lev. (2010). *DNS blocklists and reputation services, part3* [Scientific]. Available: http://blogs.computerworld.com/15273/dns_blocklists_and_reputation_services_part_3

[9] (2014, 01.21). *Botnet* [Encyclopdia]. Available: http://en.wikipedia.org/wiki/Botnet

[10] VirusBulletin. (2014, 01.21). *Gloassary; Command and Control*. Available: https://www.virusbtn.com/resources/glossary/command_and_control.xml

[11] Radware. (2013, 01.21). *DDoSPedia; Command and Control Server*. Available: http://security.radware.com/knowledge-center/DDoSPedia/command-and-control-server/

[12] M. K. Xin Hu, Kang G. Shin "RB-Seeker: Auto-detection of Redirection Botnets," 2009.

[13] C. Janssen. (12.11). *Instant Messaging Worm (IM Worm)* [Educational infmation dictionary]. Available: http://www.techopedia.com/definition/61/instant-messaging-worm-im-worm

[14] R. P. Manos Antonakakis, David Dagon, Wenke Lee, and Nick Feamster, "Building a Dynamic Reputation System for DNS," p. 17.

[15] Anonymous. (2013, 05.26). *Anti ZS spyeyes Tracker .htaccess*. Available: http://www.exposedbotnets.com/2012/05/anti-zs-spyeyes-tracker-htaccess.html

[16] James Wyke, "What is Zeus?," p. 17, May 2011.

[17] E. C. Nicolas Falliere, "Zeus: King of the Bots," p. 14.

[18] A. E. Mohamed, "Complete Cross-site Scripting Walkthrough," p. 23.

[19] E. K. Leyla Bilge, Christopher Kruegel, Marco Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," p. 17.

[20] (2014.05.01). *Conficker*. Available: http://en.wikipedia.org/wiki/Conficker

[21] SANS. *The Conficker Worm*. Available: http://www.sans.org/security-resources/malwarefaq/conficker-worm.php

[22] I. C. Roberto Perdisci, Giorgio Gaicinto, "Early Detection of Malicious *Flux* Networks via Large-Scale Passive DNS Traffic Analysis," p. 14, 2012.

[23] (2013.12.18). *mnemonic homepage*. Available: http://www.mnemonic.no

[24] O. Dictionary. (01.10). *Oxford Dictionary lookup for word: pneumonoultramicroscopicsilicovolcanoconiosis*. Available: http://www.oxforddictionaries.com/definition/english/pneumonoultramicroscopicsilicovolcanoconiosis

[25] O. Dictionary. (01.10). *Oxford Dictionary lookup for word: antidisestablishmentarianism*. Available: http://www.oxforddictionaries.com/definition/english/antidisestablishmentarianism

[26] (01.04). *Levenshtein Distance* [Technical information. Encyclopedia]. Available: http://en.wikipedia.org/wiki/Levenshtein_distance

[27] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis," *Networking, IEEE/ACM Transactions on,* vol. 20, pp. 1663-1677, 2012.

[28] S. Yadav, "Detecting Algorithmically Generated Malicious Domain Names," 2011.

[29] N. Pipitone. (2013, 02.13). *Impossible Bigrams In The English Language*. Available: http://linguistics.stackexchange.com/questions/4082/impossible-bigrams-in-the-english-language

[30] Mozilla. (04.10). *public suffix list*. Available: https://publicsuffix.org/list/

[31] (2014, 04.01). *Jaccard Index* [Technical information. Encyclopedia]. Available: http://en.wikipedia.org/wiki/Jaccard_index

[32] Ehcache. (2014, 03.03). *Ehcache homepage*. Available: http://ehcache.org/

[33] *FP rate*. Available: http://www.drcoplan.com/wp-content/uploads/2013/08/chart-two.png

[34] (02.19). Available: http://www.zoneedit.com/doc/rfc/

[35] (2014, 02.19). *RFC*. Available: http://www.webopedia.com/TERM/R/RFC.html