Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Eirik Rismyhr

Graph Representation of DNS-related Data for Detecting Malicious Actions

Master's thesis in Information Security
Supervisor: Marios Anagnostopoulos
June 2020

NTNU
Norwegian University of Science and Technology

Eirik Rismyhr

Graph Representation of DNS-related Data for Detecting Malicious Actions

Master's thesis in Information Security
Supervisor: Marios Anagnostopoulos
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Acknowledgements

I would like to thank Dr. Håkon Gunleifsen for his help in collecting the anonymized Eidsiva DNS dataset. I would also like to thank my supervisor, Dr. Marios Anagnostopoulos, for providing guidance and feedback through each stage of the process.

Abstract

Malware is an increasing problem in the cyber security domain. Recent research indicates that almost all malwares exploit DNS to carry out their malicious purposes. The DNS protocol was not originally designed with security in mind, which has made it a natural choice for malware authors. Recent and notorious security incidents have shown that DNS is used for the coordination of botnets, specifically for locating command-and-control servers and disseminating commands from the botmaster, for data exfiltration through DNS tunnelling, and for redirecting network traffic to rogue servers by hijacking the user's DNS request.

The MSc thesis at hand utilizes the Neo4J graph database solution to represent DNS related data in a graph data model, and uses this to reveal historical relations between malicious domain names and IP addresses involved in security incidents that cannot be trivially extracted with other traditional methods. In addition, structuring the DNS traffic in a graph database presents a way to discover relations between domain names where data can be extracted faster and easier than in traditional relational databases. The outcome of this thesis is a mechanism that can operate at the level of the local DNS recursive resolver, e.g., ISP, in order to detect malicious domain names and block the related traffic.

Sammendrag

Skadevare er et økende problem innen cybersikkerhet. Forskning viser at en stor andel av skadevaren benytter seg av DNS-protokollen for å utføre ondsinnede handlinger. Da DNS-protokollen først ble utviklet var ikke sikkerhet et fokusområde. Derfor har DNS blitt et naturlig valg for skapere av skadevare. Nylige alvorlige sikkerhetshendelser har vist at DNS brukes i koordinering av botnets for å lokalisere sentrale styrende noder og sende kommandoer til klienter i botnettet. I tillegg brukes DNS for å eksfiltrere data fra infiserte klienter, og omdirigere nettverkstrafikk til usikre servere ved å kapre brukerens DNS-spørringer.

Denne masteroppgaven bruker grafdatabasen Neo4j for å representere data knyttet til DNS i en grafdatamodell. Modellen kan brukes for å å avdekke historiske relasjoner mellom ondsinnede domenenavn og IP-addresser involvert i IT-sikkerhetshendelser som ikke enkelt kan hentes ut med tradisjonelle metoder. Ved å strukturere DNS-trafikk i en grafdatabase skapes også en metode for å oppdage koblinger mellom domenenavn hvor data kan hentes ut på en raskere og enklere måte enn i tradisjonelle relasjonsdatabaser. Resultatet av dette arbeidet er et verktøy for innsamling og klassifisering av DNS-trafikk i lokale navnetjenere, som kan analysere logginformasjon og oppdage domenenavn knyttet til ondsinnet aktivitet.

Contents

Figures

Tables

Code Listings

Acronyms

DNS - Domain Name System
IP - Internet Protocol
C&C - Command and Control
IDS - Intrusion Detection System
UDP - User Datagram Protocol
TCP - Transmission Control Protocol
IETF - Internet Engineering Task Force
RFC - Request for Comments
NIC - Network Information Center
FTP - File Transfer Protocol
RR - Resource Record
TLD - Top-level Domain
FQDN - Fully Qualified Domain Name
CNAME - Canonical Name
DNSSEC - DNS Security Extensions
DoH - DNS over HTTPS
HTTPS - Hypertext Transfer Protocol Secure
TSIG - Secret Key Transaction Au-thentication for DNS
DDoS - Distributed Denial of Service
MDN - Malware Distribution Networks
DHCP - Dynamic Host Configuration Protocol
DDNS - Dynamic DNS
RAT - Remote Access Tool
APT - Advanced Persistent Threat
AS - Autonomous System

Chapter 1

Introduction

1.1 Topics covered by the Thesis

The main topic for this project is detection of malicious domains using DNS related data. The data is stored in a graph database with the use of the Neo4j graph platform. This is done to make it easier to reveal historical relations and known associations between the domains and IP addresses in the data set.

1.2 Keywords

DNS, malware domain name, botnet, malicious domain detection, network traffic analysis, data labeling, network monitoring, graph analysis

1.3 Problem Description

Computer malware is a big and increasingly important issue. The Cisco 2016 Annual security report indicates that 91.3% of malwares exploit DNS to carry out their malicious purposes [1]. The exploitation allows malwares to establish Command & Control (C&C) channels, to exfiltrate data and redirect traffic [2]. In recent and notorious security incidents, we have seen that DNS has been utilized to coordinate botnets, specifically for locating the C&C server and disseminating the botmaster's commands, as well as data exfiltration through DNS tunneling, as in the case of Equifax[1]. DNS was also used to redirect network traffic to rogue servers by hijacking the user's requests.

[1] https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html

1.4 Justification, Motivation and Benefits

It is challenging to monitor and detect the malicious DNS traffic using the traditional intrusion detection systems (IDS). Evasion methods utilized by malware, such as domain flux and IP flux, makes signature based detection methods insufficient. A data model based on graph database technology could provide improved performance and help IT administrators protect their systems. By analyzing not only the malicious nodes in the graph, but also their connected nodes, there is the possibility to find additional domain names and IP addresses that are related to malicious activities. In particular, this will be useful for detecting botnets and malware distribution networks that utilize the aformentioned evasion techniques to avoid network security monitoring tools such as IDSs. Graph databases such as Neo4j can be queried with easily formatted queries, and query results are returned faster than in relational databases.

1.5 Research Questions

In order to achieve the desired results, the following research questions were defined:

- **RQ1:** How can DNS data be represented in a graph database?
- **RQ2:** How can data from external sources be incorporated into the graph structure?
- **RQ3:** How can graph databases be used to detect malicious domains?

1.6 Contributions

The main contributions of this thesis project can be summarized as follows:

- Graph database representation of DNS related data from captured network traffic using the Neo4j Graph Platform [3].
- Algorithm that determines the maliciousness of a domain
- Monitoring tool that can operate at the level of the local DNS recursive resolver. The main goal of this tool is to detect internal devices trying to connect to malicious domain names, as well as identifying access requests from malicious servers.

Chapter 2

Background

2.1 DNS Overview

Domain Name System (DNS) is a protocol implemented in all IP-based networks. It works by converting human readable domain names, such as `www.google.com`, into computer readable IP addresses, such as 8.8.8.8. This removes the need to know the IP address of the service that one wants to connect to. It is a hierarchical decentralized system, and a fundamental part of the Internet infrastructure. DNS requests and responses can be sent using both the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), but UDP is normally used because it has a lower overhead requirement than TCP. DNS is an open protocol, and the Internet Engineering Task Force[1] (IETF) oversees changes made to the standard. The initial version and all later additions are described in IETF Request for Comments (RFC) standard documents. The most important concepts are described in RFC 1034 [4] and RFC 1035[5]. Subsequent RFCs have added additional features, including more resource record types and security improvements.
In the early days of the Internet, mappings between host name and addresses were maintained by the Network Information Center (NIC) in a single file named HOSTS.TXT. This file was transferred to all hosts via the file transfer protocol (FTP). This worked well when the number of hosts was small. However, when the number of hosts increased, the size and update frequency of the HOST.TXT file was complicated to a level where it became hard to manage. One of the main motives for the development of the DNS protocol was to mitigate this problem by storing domain name mappings in a distributed hierarchical system.
The DNS consists of three major components:

- **Domain name space and resource records** specify the name space, which is a tree-structure where each node and leaf represents a set of information. Query operations are performed when specific types of information are to be retrieved from a set. Each query contains a domain name and the requested resource information.

[1] `https://www.ietf.org/`

- **Name servers** contain information about the domain name space and the information in each set. These servers generally contains complete information about a subset of the domain space, and can also contain cached structure or set information. In addition, each server stores pointers to other name servers that might contain the requested information, if it is not present in this particular server. If a server contains complete information about a part of the domain tree, it is an authority for this data set. This authoritative information is divided into zones, which can be automatically distributed to the name servers. Each name server has local copies of a number of zones.
- **Resolvers** respond to client request by retrieving the desired information from name servers.

Each node in the domain name space tree structure has a label and represents a set of resource records (RR). Domain names are built up by traversing the path from a node to the root of a tree. Each level in the tree is separated by a dot character while the root node is represented by a label of zero length. The domain name is built from left to right. Figure 2.1 is taken from RFC 1034 [4] and shows a part of the domain name space as it were at the time of writing in November 1987. The three subdomains of the root (MIL, EDU and ARPA) are known as top-level domains (TLD), and the domains in the next level below are known as second-level and third-level domains. A domain name can consist of a maximum of 127 levels or 253 characters. Many different domain names can be created by traversing this tree from the leaves to the root node. A complete domain name such as A.ISI.EDU is known as a fully qualified domain name (FQDN).

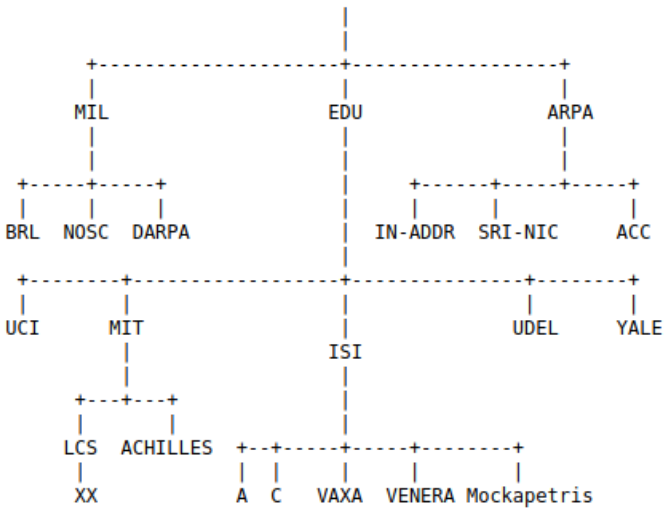


Figure 2.1: Domain name space example [4]

2.1.1 DNS Query Types

There are two types of queries:

Recursive Query

In recursive queries, a DNS client provides a domain name to the resolver, which performs the complete transaction. The recursive query process starts at the DNS root server and finishes when an authoritative name server that contains the requested information is found. The complete answer to the query must be returned.

Non-Recursive Query

A non-recursive (or iterative) query is a query where a DNS client provides a domain name and the resolver returns the requested information, if it is stored in its cache memory. If the resolver does not have the information, it provides a referral to authoritative name servers that might have it.

2.1.2 DNS Name servers

There are three types of DNS servers, all of which are involved when a domain name is resolved: stub resolvers (end-user), recursive resolvers, and authoritative nameservers.The root nameservers are the authoritative nameservers for the root zone, while the TLD nameservers are the authoritative nameservers for the TLDs, such as .com, .org or .no.
The following steps take place in the case when a domain name, for instance *example.com*, is resolved:

1. A program on the user's computer, such as the browser, sends a DNS request for a resource record (RR) to the stub resolver.
2. The stub resolver on the client sends the query to the DNS recursive resolver.
3. If the RR is not contained in the resolver's cache, the recursive resolver sends the query to one of the authoritative servers for the root zone.
4. If the queried authoritative name server is not authoritative for the requested information, it sends the query to the TLD name server that is authoritative. In this case it is sent to the .com TLD authoritative nameserver.
5. The original query is sent until it reaches the authoritative name server for the example.com zone. This server then finally provides the answer.

Figure 2.2 shows an overview of the process that occurs when a query is sent.

2.1.3 DNS Resource Records

Contents in DNS zone files are known as resource records (RR). Each DNS query contains a request for a specific RR type.

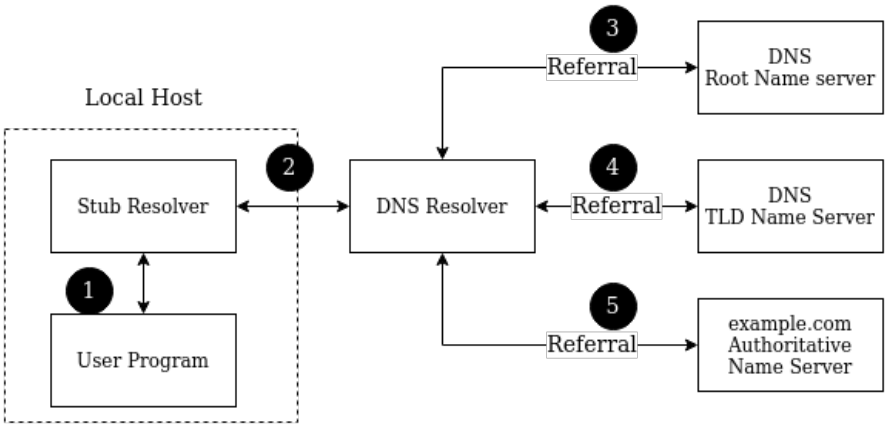


Figure 2.2: The process that takes place when a DNS query is sent

Record Content

Each RR has certain fields:

- **Owner:** Domain name that the RR belongs to.
- **Type:**A 16-bit values that specifies the type of this particular RR. Many types exist, as shown in section 2.1.3
- **Class:** Similar to the type field, but specifies a protocol family or instance of a protocol.
- **Time to live (TTL)** How long (in seconds) a RR can be cached.
- **RDATA:** Content of each type.

There are defined many DNS record types, but some of them are more commonly used. Some of these record types are used in the graph data model developed in this project. These were specifically chosen because of their prevalence in DNS traffic, and the usefulness of the information they contain. The most common record types are:

DNS Record Types

- **IP version 4 address (A):** The IPv4 Host address for the requested domain name.
- **IP version 6 address(AAAA):** The IPv6 Host address for the requested domain name.
- **Canonical name (CNAME):** Canonical name for a a domain alias. One domain can have several CNAME values, meaning that several domain names point to the same A or AAAA record.
- **Mail exchange (MX):** Name for the mail exchange server related to the domain.
- **Text (TXT):** It was originally intented to contain human-readable text, but often contains machine-readable code. It can be used for ownership verific-

ation or to determine the trustworthiness of the source using Sender Policy Framework (SPF) codes.
- **Name server (NS):** The authoritative name server for the domain.
- **Start of Authority (SOA)** It is located a the start of a DNS zone file. It contains information about the zone, such as the Authoritative Name Server, email address for the domain administrator, and how often this information should be updated.
- **Service location (SRV)** It specifies the host and port for different communication protocols.
- **Reverse-lookup Pointer (PTR):** It is used for reverse DNS lookup queries and shows the mapping between an IP address and a domain name.

2.2 DNS Security

When the DNS protocol was initially designed, security was not a primary concern. Therefore, DNS by itself is not a secure protocol. When a query is sent from a recursive resolver to an authoritative name server, there is no way to check the authenticity of the data in the response. The source of the response packet can be found by checking the IP address. However, IP addresses can be forged by attackers who can then pretend to be the authoritative server. This can be exploited to redirect users to malicious sites without their knowledge.
Another security issue is DNS cache poisoning [6], where an attacker sends forged DNS response packets to a recursive resolver. If the resolver accepts and caches the response, the cache is considered poisoned and other end-users that request the same domain name will receive DNS responses with forged data.
To improve the security of DNS, DNSSEC was introduced with RFC 4033 [7], RFC 4034 [8] and RFC 4035 [9] in 2005, which has later been updated with additional features such as stronger encryption algorithms. DNSSEC introduced the ability for DNS data to be digitally signed by the owner using public key cryptography. Each group of RRs with the same type is grouped into RRsets, which is then digitally signed.
Several DNS record types were added:

- **RRSIG:** Contains a cryptographic signature
- **DNSKEY:** Contains a public key
- **DS:** Contains the hash of a DNSKEY record
- **NSEC and NSEC3:** Used for denial-of-existence of a DNS record
- **CDNSKEY and CDS:** Used when a child DNS zone requests updates to DS records in the parent zone.

When combined, this data can be used to provide two important security features:

- **Data origin authentication** allows resolvers to verify the zone which the response data originated from.
- **Data integrity protection** allows resolvers to verify that the data received has not been modified in transit.

DNSSEC by itself has not been enough to mitigate all of the security issues in the DNS protocol. DNS over HTTPS (DoH) is a another security feature introduced in 2018 with RFC8484 [10] where DNS traffic is sent via Hypertext Transfer Protocol Secure (HTTPS). This has been implemented in recent years by popular web browsers such as Mozilla Firefox and Google Chrome, and is gradually being rolled out in more and more regions.
To secure the communication between DNS servers, Secret Key Transaction Authentication for DNS (TSIG) was introduced in 2000 with RFC2845 [11]. TSIG uses shared private keys to provide a secure method to perform zone updates. It is normally used to update dynamic DNS servers or secondary DNS servers.

2.3 DNS Vulnerabilities

Even after the introduction of DNSSEC, TSIG and DoH, there exists weaknesses in the DNS protocol. They are continually being discovered, and in May 2020 Shafir et. al. [12] discovered a flaw in the way that recursive resolvers handle NS referral responses that contain the domain name but not the corresponding IP address. This creates a potential for amplification attacks where a single malicious packet could be amplified up to 1620 times, causing the resolver to be overloaded.
In this thesis, we focus on attacks that use features in the DNS protocol to build more resilient botnets and malware distribution networks. These types of threats are serious and often challenging to detect and prevent. Domain names and IP addresses used in botnets are often changed rapidly to make the botnet more resilient against detection.

2.3.1 Botnets

Botnets [13] are networks of computers that have been compromised by malware and taken over by criminals. The botnets can reach sizes of up to millions of infected devices, where all of them are controlled by a botmaster. The size allows them to carry out large-scale attacks such as spam campaigns or Distributed Denial of Service (DDoS) attacks [14].
The bot members receive commands from the botmaster through command and control servers, also known as C2 or C&C servers, to receive orders and exfiltrate data. These servers rely on DNS to get the correct mappings between the domain names and IP address of each C&C server. Without this, the computers in the botnet cannot communicate with the servers.
Domain names and IP addresses used by botnets are continuously being blocked by security systems as they are discovered. To avoid detection, botnets often employ evasion techniques. If a C&C server goes down, another one takes its place in the network. This both makes the botnet traffic harder to detect and harder to stop. It used to be common for malware to use hardcoded IP addresses. This practice was abandoned, as it allowed the botnets to be easily shut down. The two main techniques used to achieve agile behaviour are Domain-Flux and IP-Flux,

also known as Fast-Flux [15]. The Domain-Flux strategy involves having several FQDNs associated with one IP address. New domain names are dynamically generated using Domain Generation Algorithms (DGA). To main goal is to generate a large number of domain names to make the botnet more resilient against attempts to take down the C&C servers or filter out the traffic.
Malware distribution networks (MDN) are another type of threat that has seen increased severity in the last years. A large amount of domains are used to trick users into installing malicious software. A main attack vector is drive-by download attacks where victims are lured into visiting malicious web pages that exploits weaknesses in the users' web browsers and their components. A number of articles investigate MDNs, including research by Wang et. al. [16] and Invernizzi et. al. [17], who analyze features of domains in MDNs.
IP addresses are usually assigned by ISPs dynamically using the Dynamic Host Configuration Protocol (DHCP). The addresses are typically assigned on a lease with a limited duration, meaning that a registered domain name will resolve to different IP addresses over time. Dynamic DNS (DDNS) services automatically update DNS records, and is provided by many registrars, for example Cloudflare[2]. This service makes dynamic DNS a useful evasion technique for bots, trojans and other Remote Access Tools (RAT) that depend on command and control servers. Zhao et. al. [18] investigate network traffic generated by advanced persistent threat (APT) malware and find that dynamic DNS is often used by attackers, and that several features found in the DDNS traffic could be used in detection algorithms. Some DDNS providers provide their services for free, which makes them a natural fit for malware.

2.4 Detecting Malicious Domains

As shown in the survey by Zhauniarovich et al. [19], there are many different approaches that attempt to detect malicious domain names by analyzing DNS data. A large share of the approaches in this survey use similar methodologies and are quantified using the following features:

- DNS data collection
- Data enrichment
- Detection methods
- Evaluation strategies

2.4.1 Data Sources

Sources of DNS-related data can be divided into two classes. Multiple locations in the DNS infrastructure can be used to collect DNS queries and replies.
The first type of data source is the resolver, where queries from end clients can be collected. One of the advantages with this is that it contains detailed information

[2] `https://www.cloudflare.com/learning/dns/glossary/dynamic-dns/`

about the queries and responses related to each client. This can be analyzed to find suspicious patterns in the traffic. However, only the traffic in a single network can be observed. This could be a challenge when analyzing malicious patterns. Also, privacy concerns often makes data from public resolvers difficult to access for researchers.
The second type of data source is traffic between DNS servers. Traffic can be collected at DNS servers such as authoritative name servers or TLD servers in order to see DNS requests from several organisations. However, there are several issues with this approach. Logs from these types of servers are usually not available to researchers, and often contain fewer features than data captured at resolvers.

2.4.2 Collection Methods

DNS data can be collected either actively or passively. Active data collection is done by sending DNS queries and monitoring the responses [20]. The queried domains are based on lists of popular domains such as the Alexa Top Sites[3] and The Majestic Million[4], as well as domains from blacklists and authoritative zone files. This methods works well when retrieving RRs, but do not reflect normal DNS traffic. There is also a potential for data bias due to factors such as the geo-location of the querying clients.
Passive data collection is done by monitoring the traffic at DNS servers or investigating server logs [21]. Sensor can be placed in several locations, which could lead to a more comprehensive set of features than in network logs that have been actively collected. However, publicly available datasets of this type usually only contain aggregated information about the traffic due to privacy concerns.

Data Enrichment

External data sources can be used to improve the accuracy when detecting malicious domains. The most widely used information types are [19]:

- **Blacklists/whitelist** for IPs and domain names are often used. As described in section 2.4.2, a variety of blacklists exist.
- **Registration Records** contain information about domain registrars, and temporal information such as creation/expiration time. They can be used to find relations between malicious domain names, as they are often registered by the same registrar in the same time period. Registration records are accessed through the WHOIS protocol [22], but parsing of the data can be challenging due to the lack of a standard format.
- **Autonomous System Numbers (ASN)** provide information about the distribution of IPs. An autonomous system is a single network or a group of networks that is controlled by a common network administrator. Malicious

[3] https://www.alexa.com/topsites
[4] https://majestic.com/reports/majestic-million

domains often change ASNs to evade detection, while legitimate domains remain mostly static.
- **Additional Network Data** such as HTTPS logs can be used to gain a better understanding of the domains.
- **Geo-location** can be retrieved from sources such as the Maxmind GeoLite2 database[5]. Features such as the reputation or geographical distances between hosting countries can be used to improve classifier performance [23].

Ground Truth

Most of the approaches described in this thesis and in the survey [19] utilize machine learning algorithms. Supervised and semi-supervised detection algorithms require a set of trustworthy ground truth for training and validation. The result of the evaluation phase depends on how the ground truth is processed and applied. The most popular way to gather a ground truth for the maliciousness of domains is to extract information from public blacklists. There exists a large amount of blacklists both for domain names and IP addresses. Some are based on specific types of activities, e.g. phishing (PhishTank[6], OpenPhish[7]) or spam (Spamhaus[8]), and others are more general (Malwaredomains[9], Malware Domain List[10]). There also exists proprietary reputation systems developed by security companies such as Symantec. However, these systems are often not available for research usage.
Most of the algorithms used to detect malicious domain are data-driven, and use machine learning to improve accuracy. These algorithms require a ground truth of malicious and domain traffic that can be used for training and evaluation of the machine learning methods. A simple semi-manual labeling for agile DNS domains is presented in Stevanovic et. al. [23]. DNSMap is used to provide mapping of agile domains names. Automated analysis as well as cluster analysis is performed before a human operator performs manual validation. A case study confirms that the semi-manual approach achieves better coverage than approaches relying solely on domain black/whitelists as it can discover malicious domains based on their association with other malicious domains and IP addresses. The proposed method is also time efficient.
DFBotKiller [24] is an online negative reputation system that detects botnets using domain-flux. This is done by analyzing traffic logs to find suspicious domain group activities and suspicious domain failures. This solution has a good detection rate, and a low false positive rate when provided with the history of suspicious domain activities.

[5] `https://dev.maxmind.com/geoip/geoip2/geolite2/`
[6] `https://www.phishtank.com/`
[7] `https://openphish.com/`
[8] `https://www.spamhaus.org/`
[9] `https://www.malwaredomains.com/`
[10] `https://www.malwaredomainlist.com/mdl.php`

2.5 Graph Representation of DNS Data

Berger et al. [25] show that the mappings between FQDNs and IP addresses can be used to establish a DNS activity profile. These profiles can be used for the detection of abnormal activities. DNS mappings not conforming with the normal profiles could be considered suspicious and should be further analyzed. Following, DNSMap [26] is used to detect agile DNS mappings. This is a methodology used to track observed mappings between FQDNs and IP addresses. The resulting set of mappings are used to create a bipartite graph which shows an overview of the observed mappings. After filtering, the final graph contains a set of agile groups. These groups are classified using the following features:

- Number of FQDNs per agile groups
- Number of IP addresses per agile group
- Number of different Autonomous Systems (ASs) per agile group

Peng et al. [27] explore the use of DNS CNAME resource records to construct an alias-canonical graph. This is used to determine if each domain in the dataset is malicious or benign. This approach can process large amounts of DNS traffic and identify malicious domains in near real time. This approach uncovered a set of malicious domains that other approaches were unable to find. The process of identifying malicious domains consist of three main steps:

- Removing CNAME RRs from public domains (web hosting, CDNs etc.) and building an alias-canonical graph G from the remaining RRs.
- Classifying domain nodes in G as malicious, benign or unknown with the help of blacklists and whitelists.
- Using a Belief Propagation (BP) algorithm to compute the marginal probabilities for each node based on the association with other nodes.

The dataset used to train and test the model consists of passive DNS traffic collected at 217 DNS server distributed in 14 large Chinese ISP networks. It contained over 2.5 billion DNS A records and 1.1 billion DNS CNAME records, and was collected over a period of 1530 days from February 2012 to June 2016. Testing of this classifying technique on the real-world dataset yielded a true positive rate of 97.25% and a false positive rate of just 0.027%.
Several approaches for DNS monitoring use spatial and temporal attributes to determine the maliciousness of domains. Lee et. al. [28] propose the use of sequential correlation, namely the correlation between domains queried before or after each other. The degree of the sequential correlation is determined with a client sharing ratio (CSR). The CSR is estimated using the Jaccard similarity of the IP addresses belonging to querying clients. The main advantage of using sequential correlation is the sensitivity of the detection method. This detection method gathers temporally scattered traffic from each client and unsynchronized traffic between clients on a graph structure called Domain Name Travel Graph (DNTG). Another advantage is the accuracy of the detection, which is achieved by filtering out the noise created by traffic to legitimate domain names. When the DNTG has

been constructed, related domain names are grouped together in clusters to detect which malware domains work together. Malicious domains are then detected using a domain blacklist. The DNS data used was captured from DNS servers in large ISP networks in the U.S and South Korea, containing between 1713K and 8661K queries. Experiments with GMAD resulted in a detection accuracy and sensitivity superior to other detection methods available at the time.
A recent attempt at using graph databases to analyze network log files is described in Diederichsen et. al. [29]. Zeek[11] is used to capture traffic from various protocols and generate log files in real time. The log files are then entered into a Neo4j graph database in order to facilitate efficient analysis of relations within the network traffic. Figure 2.3 shows the graph data model used when creating the DNS database.

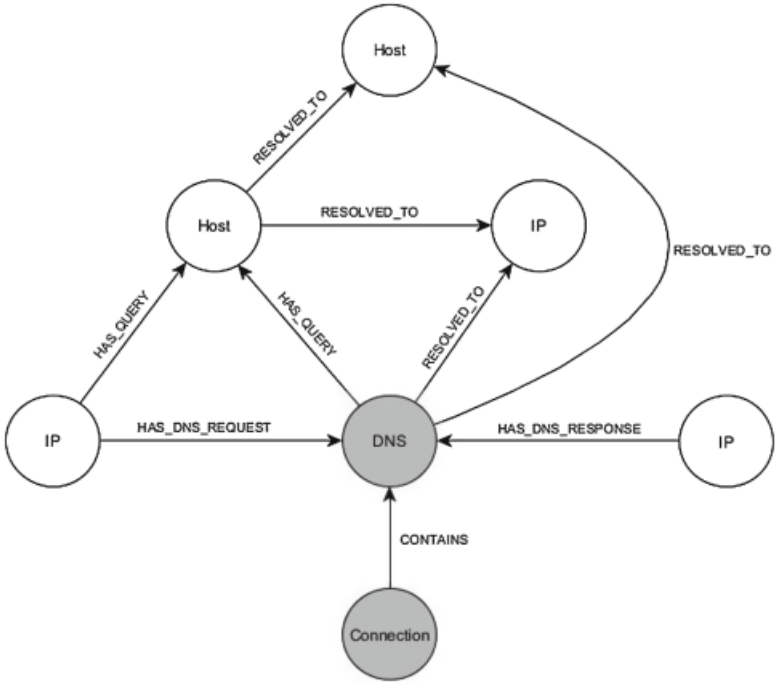


Figure 2.3: An example of a graph data model for DNS logs [29].

2.6 Domain Features

This section describes the features used by the different approaches to classify and define relations between domain names. The features used vary greatly, and most of the approaches use data from external sources such as the ones described in section 2.4.2.

[11] `https://zeek.org/`

The automated analysis performed by Stevanovic et al. [23] is used for characterizing the graph components that remain after the filtering is performed. A number of features are extracted for each of the graph components, based on theoretical knowledge and empirical evidence. They can be grouped into 6 categories:

1. Graph analysis
2. FQDN analysis
3. IP analysis
4. FQDN whitelist analysis
5. FQDN blacklist analysis
6. IP blacklist analysis

Zou et al. use both a DNS Query Response Graph (DQRG) and a Passive DNS Graph (PDG) in order to detect malicious traffic. [30]. The PDG uses A and CNAME records. Prior knowledge of both domains and hosts is used to determine the maliciousness of domains. Domain prior knowledge consists of well-known legitimate domains, known malicious domains, and domain reputation gathered from Alexa Top Sites[12] and freely registerable subdomains. The sources of known malicious domain includes several well-known domain blacklists. Domain suffixes for DDNS providers are also gathered. Host prior knowledge is gathered in a similar way.
Khalil et al. [31] use passive DNS replication to capture inter-server DNS messages. It focuses on A records where each record contains information about the domain name, IP address, first observation, last observation and the number of observations.
Yadav et al. [32] analyse IP-addresses and determine if they belong to a botnet. Benign addresses are filtered out using the following measures:

- Degree of each IP-address: The number of domains that map to this IP.
- Correlation metric: Analyses the correlation between DNS successes and failures in a given time window. It is computed as the probability of observing at least one failed DNS query in a time window, given that the IP was returned as an answer to a successful DNS query in the same window.
- Succeeding Domain Set Entropy: Measures the edit distance between domain names in successful DNS queries
- Failing Domain Set Entropy: Measures the edit distance between domain names in DNS failures. To compute the entropy of failed domain names, failing queries that occur in the vicinity of a successful DNS query are analysed.

Jiang et al. [33] create a graph based on failed DNS queries and use this to discover traffic to malicious domains. This work focuses on DNS type A request, which contain the IPv4 address of the domains. All queries that contain other response codes than "NOERROR" are considered failed queries and could be indicators of malicious traffic.

[12] `https://www.alexa.com/topsites`

Chapter 3

Choice of Methods

This section describes the technical design of our implementation, the design process and the graph data model used to import log files into Neo4j.

3.1 Requirements

The main part of the workload in this project involved creating a program that could extract the DNS resource records needed to create the graph data model. In order to have a clear understanding of what needed to be done, a certain set of functionality was required:

- Parse DNS RRs from log files in PCAP, TXT and CSV file formats.
- Load external data into Cypher queries.
- Collect and parse WHOIS data.
- Collect and parse ASN and ISP data.
- Check blacklists and whitelists.
- Ability to run Cypher queries in Python.

3.2 Technical Design

Figure 3.1 shows the general design of the software that creates the graph data model used in this project. The Python program *import_pcap.py* extract information from log files (either in PCAP or TXT format) and combines this with data from external sources, including WHOIS, RIPE and several domain filtering lists. This data is then stored in a Python dictionary, which is passed to the official Neo4j Python driver. The driver links the Python program with the local Neo4j server, and creates the data model based on several Cypher queries. A complete list of the Python functions can be found in appendix A.

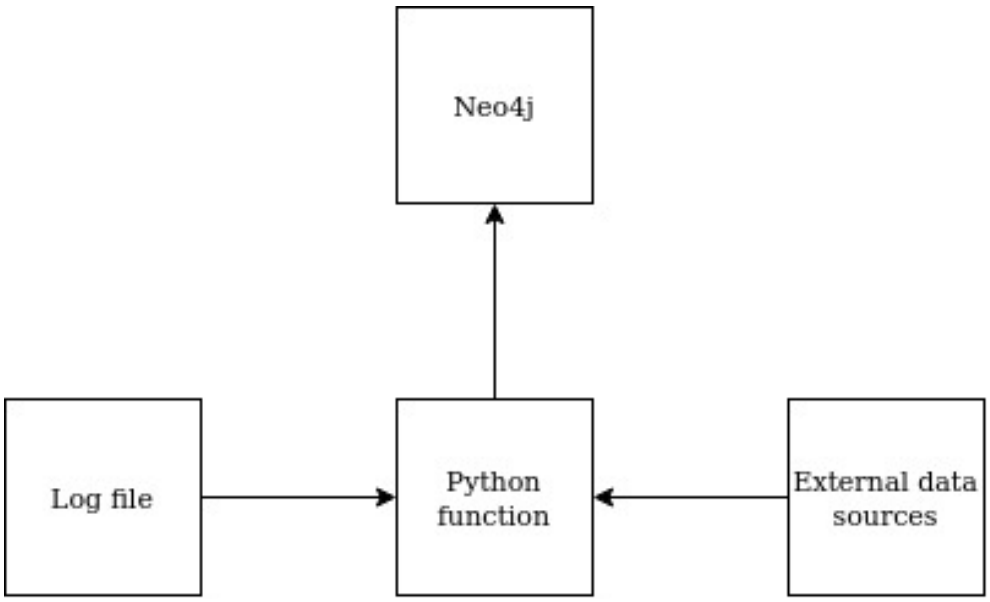


Figure 3.1: Overview of the log import process

3.3 Implementation

The software for importing DNS log files was developed in Python 3 using Pycharm[1]. Python was chosen for several reasons. Its simple syntax allowed for more time to focus on developing the graph data model. There also exists an official Neo4j Python driver, and several libraries for analyzing network traffic and gathering the necessary enrichment data to be used in the database.

3.3.1 Python Libraries

Several Python libraries were used to extract and process log data:

- **PyShark**[2] is a Python wrapper for tshark that allows networks packets to be parsed using wireshark dissectors.
- **whois**[3] is a wrapper for the Linux whois command. In this implementation, the whois command directly queries the WHOIS server for each domain name and retrieves information about the registrar and creation date for each domain name.
- **Neo4j Python Driver**[4] provides a connection between Python programs and local Neo4j databases.

[1] https://www.jetbrains.com/pycharm/
[2] https://github.com/KimiNewt/pyshark
[3] https://github.com/DannyCork/python-whois/
[4] https://neo4j.com/docs/driver-manual/current/

3.3.2 Deployment

The entire source code and database dumps are available online[5]. The database dump files can be loaded[6] into Neo4j in order to reproduce the results of this project. The following software is required to run and reproduce the results in this thesis:

- Ubuntu 18.04 or later
- Python 3.6 or later with packages:
    - Neo4j Python Driver 4.0 or later
    - PyShark 0.4.2.9 or later
    - whois 0.9.7 or later
- Neo4j 4.0 or later

3.4 Data Analysis

An exploratory study was performed in order to find or collect the data needed for this project [34]. The dataset needed to contain relevant features for domain classification and be extensive enough to be used for both testing and evaluating a machine learning algorithm. We considered collecting the data manually from a DNS-resolver, but this was found to be too challenging for several reasons. Getting access to a resolver is difficult due privacy concerns. Active data collection was also ruled out because it would likely lead to a sparse or biased datasat that would not reflect normal DNS traffic. Instead, datasets were collected online at DNS database repositories. Eidsiva also kindly provided a dataset of DNS traffic captured at their local resolver.

3.5 Experiments

The implementation follows the scheme presented in [19] closely. Since the outcome was a graph data model, and not a machine learning classifier, the evaluation strategy was different.

1. Data collection
    - Collect and select appropriate DNS datasets that can be stored in a graph database and evaluated.
    - Enrich the DNS data using data from external sources.
    - Establish a ground truth of data that can be used to train and test the classifier.
2. Detection

[5] `https://github.com/eirikrismyhr/MIS4900`

[6] `https://neo4j.com/docs/operations-manual/current/tools/dump-load/`

- Select features in the dataset that are suitable for classification algorithms.
- Develop a graph data model based on the selected features.
- Import dataset along with enrichment data into Neo4j.

3. Evaluation
    - Evaluate various use cases for the graph data model.

3.6 Graph Database Models

Graph databases present an alternative to relational database management systems (RDBMS). There are several advantages with graph databases that have motivated an increasing number of companies to start using them. Some of the advantages include performance, flexibility and agility [35]. This provides a database solution that can be used to store and retrieve data quickly.

3.6.1 Neo4j

Neo4j[7] is a transactional, ACID-compliant database. ACID represents four goals that many database management systems strive to accomplish [36]:

- Atomicity: Transactions only happen when all parts of the transaction complete successfully.
- Consistency: Only valid data can be entered into the database, meaning that it has to follow to database schema. However, Neo4j implements an optional schema, meaning that the consistency rules are looser than in relational databases.
- Isolation: If multiple transactions are executed on the database at the same time, they cannot impact each other. For example, if one write transaction is writing to the database, read transactions must wait until the write operation is complete. This is to ensure that the data stays in the correct state.
- Durability: Committed transactions cannot be lost. This is ensured with persisted storage and transaction commit logs.

There are several reasons why Neo4j was chosen as the Database Managements System (DBMS) to be used in this project. One of the main reason is its scalability. More specifically, this includes

- Capacity
- Latency
- Throughput

Another main reason for choosing Neo4j is that it is a graph database. This makes it easier to find relations between entries in the database than in traditional relational databases which only return query results in tables. In addition, graph

[7] `https://neo4j.com/`

databases such as Neo4j can be queried with easily formatted queries, and query results are returned faster than in relational databases.
Nodes and relations in the graph database were created using the official Python driver, which allows for easy data import. Neo4j also includes a desktop client known as Neo4j Desktop[8] that was used to manage the local instances of the databases in this project. Included in Neo4j Desktop is the Neo4j Browser that can be used to run Cypher queries, see the results both in graph and text format, and gain an overview of all node label, relationship types and property keys in the database that is currently running.

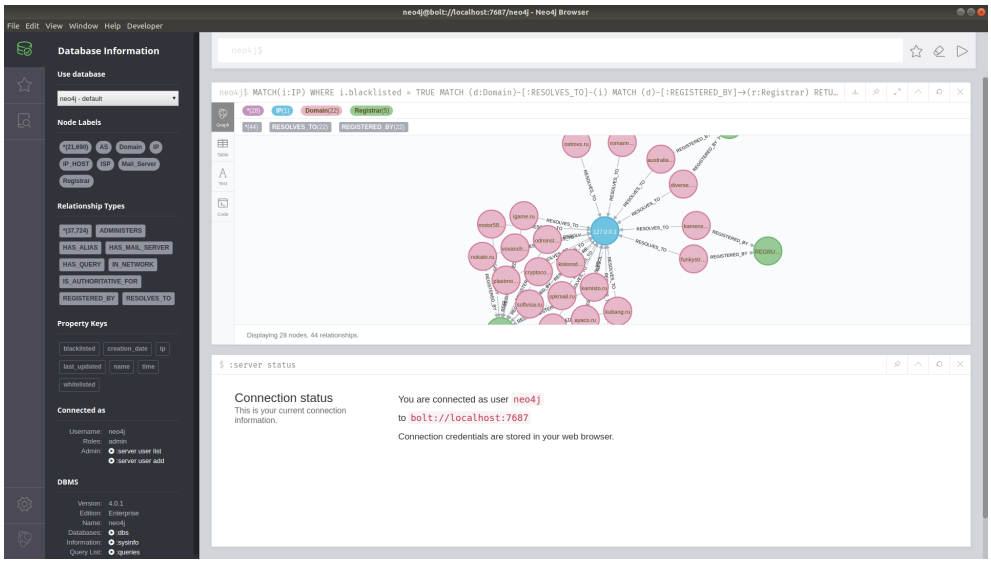


Figure 3.2: Neo4j Browser user interface

Nodes

Each node in the graph represents one data type in the database. Nodes have unique labels to differentiate them from other nodes. They also have attributes, for instance, Domain nodes have a *name* attribute.

Relationships

Nodes are usually connected to other nodes through relationships, in the same way that vertices are connected by edges in graph theory. The relationships also have unique labels and can contain properties such as timestamps. Each relationship is either directed or undirected.

[8] `https://neo4j.com/developer/neo4j-desktop/`

Cypher

Cypher [37] is the query language used to store and retrieve data from the Neo4j graph database. Its syntax is designed to be simple and human-readable, and is inspired by ASCII art. The queries in Cypher constitute the CRUD operations (Create, Read, Update, Delete), which are the basic functions Cypher supports. Cypher is also used in other property graph databases and is made open source through the openCypher[9] project.
The query structure is similar to the one used in SQL, where queries are built using several clauses. The most used clauses are the following:

- MATCH: Selects a graph pattern based on a certain pattern. Similar to SELECT in SQL.
- WHERE: Adds constraints to the pattern used in MATCH. Similar to WHERE in SQL.
- RETURN: Defines what data is returned.
- SET: Updates node labels and properties on nodes and relationships.
- CREATE: Creates nodes and relationships.
- MERGE: Creates a pattern in the graph if it does not already exist. Can be used to create both nodes and relationships in the same way as CREATE.

The simple query shown in 3.1 creates a subset of the data model described in section 3.8. A computer with the IP address 192.168.2.1 sends DNS requests for the domain names *google.no* and *nrk.no*, which resolves to 172.217.21.131 and 91.135.34.18 ,respectively.

Code listing 3.1: Cypher Query Example

```
CREATE (src:IP_HOST {ip: '192.168.2.1'})
CREATE (g:Domain {name: 'google.no'})-[:RESOLVES_TO]->(:IP {ip:'172.217.21.131'})
CREATE (nrk:Domain {name: 'nrk.no'})-[:RESOLVES_TO]->(:IP {ip:'91.135.34.18'})
CREATE (src)-[:HAS_QUERY]->(g)
CREATE (src)-[:HAS_QUERY]->(nrk)
```

3.7 Data Collection

The log files used in this thesis were in different file formats, and therefore required different data parsing methods. Figure 3.4 shows an example of a DNS query from the CTU-13 dataset [38], captured in PCAP format. The dataset provided by Eidsiva was in the format shown in code listing 3.2.

3.7.1 Datasets

CTU-13 Dataset

The CTU-13 dataset consists of 13 network traffic captures from different botnet samples in the CTU University, Czech Republic, in 2011 [38]. Each capture

[9] https://www.opencypher.org/

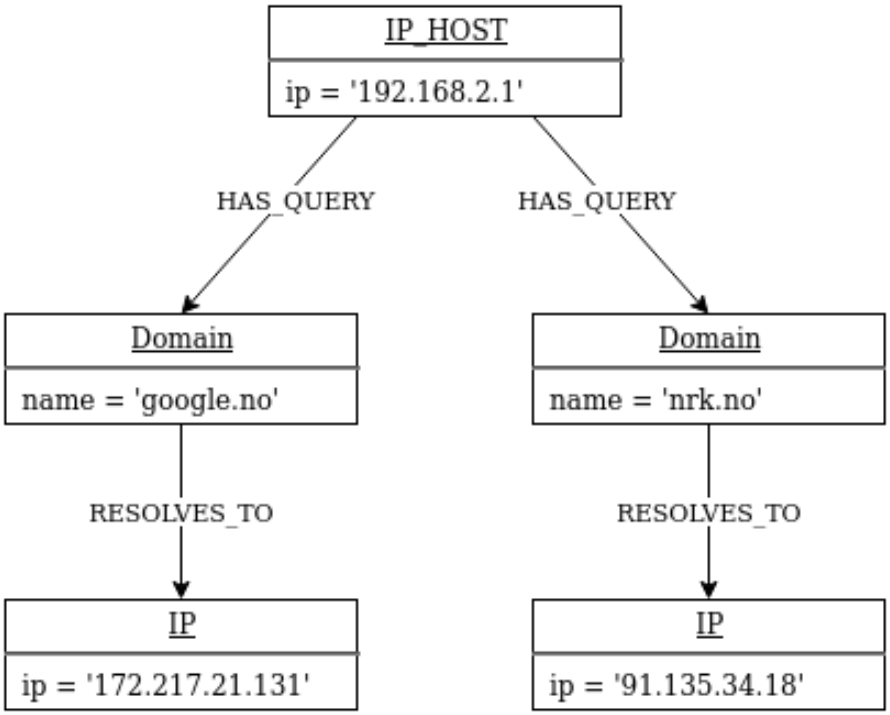


Figure 3.3: The resulting graph of the query in code listing 3.1

```
Transaction ID: 0x91e3
Flags: 0x0100 Standard query
0... .... .... .... = Response: Message is a query
.000 0... .... .... = Opcode: Standard query (0)
.... ..0. .... .... = Truncated: Message is not truncated
.... ...1 .... .... = Recursion desired: Do query recursively
.... .... .0.. .... = Z: reserved (0)
.... .... ...0 .... = Non-authenticated data: Unacceptable
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
Name: cm.g.doubleclick.net
Name Length: 20
Label Count: 4
Type: A (Host Address) (1)
Class: IN (0x0001)
cm.g.doubleclick.net: type A, class IN
```

Figure 3.4: DNS query example from the CTU-13 dataset

contains botnet traffic mixed with normal traffic and background traffic.
Due to the large size of the complete dataset, we chose to use only scenario 2 when testing the model. This scenario[10] contains traffic from a malware sample known as Neris. The capture file used in this thesis project contains only the botnet traffic from the infected machine. The background and normal traffic contains private information and has not been made public. The botnet traffic does not reflect normal DNS traffic, but it contained sufficient data to properly test our model. It is also a traffic volume that could be imported to our model in an acceptable time.

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	2,824,637	52GB	Neris	1
2	4.21	71,851,300	1,808,123	60GB	Neris	1
3	66.85	167,730,395	4,710,639	121GB	Rbot	1
4	4.21	62,089,135	1,121,077	53GB	Rbot	1
5	11.63	4,481,167	129,833	37.6GB	Virut	1
6	2.18	38,764,357	558,920	30GB	Menti	1
7	0.38	7,467,139	114,078	5.8GB	Sogou	1
8	19.5	155,207,799	2,954,231	123GB	Murlo	1
9	5.18	115,415,321	2,753,885	94GB	Neris	10
10	4.75	90,389,782	1,309,792	73GB	Rbot	10
11	0.26	6,337,202	107,252	5.2GB	Rbot	3
12	1.21	13,212,268	325,472	8.3GB	NSIS.ay	3
13	16.36	50,888,256	1,925,150	34GB	Virut	1

Figure 3.5: Data in each botnet scenario in the CTU-13 dataset [38]

Eidsiva Dataset

The second dataset used in this project was provided by Eidsiva and is in a different format than the CTU-13 dataset. It is in the TXT file format where the following fields will be used from each record: Date, timestamp, anonymized client IP address and queried domain name. It contains 5,457,344 DNS requests captured over a period of 52 minutes. The content of the DNS responses are not included so the model only contains the domain name, date and timestamp for each DNS query. IP addresses for the queried domain names are not available because this log file only contains the traffic between the client and resolver.
In the first version of the import program, the values necessary for creating the nodes and relationships were retrieved directly from the TXT file. When running the program on the first 10000 lines of the file, 13,241 nodes and 9,790 relationships were created in the database. The processes was completed in 879.56 seconds (14.65 minutes). This runtime was deemed unsatisfactory, and a new

[10] `https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-43/`

import method which used the LOAD CSV Cypher command was created. In addition, a function named txt_to_csv.py was developed to convert the dataset into the necessary comma-separated values (CSV) file format. The complete function in shown in code listing A.8 Importing the first 10000 lines using LOAD CSV resulted in a large improvement of the runtime, down to 74.93 seconds. However, this method was not used in the final version due to difficulties in importing external data.

Code listing 3.2: Example record from the Eidsiva dataset

```
04-May-2020 10:01:37.943 client 2bfc07a5afe8fc8d4242763c3fb55b761e0115c4
(tenor.googleapis.com): view ntp-stealth: query: tenor.googleapis.com IN A
+ (82.147.40.12)
```

3.7.2 Enrichment Data

Blacklists

The blacklists contain domain names or IP addresses that have been marked as malicious based on a reputation system where a reputation score is calculated based on a set of knowledge. These lists are the main method of determining if an Domain or IP node in the graph is malicious. The following blacklists are utilized in this project:

- Cybercrime Tracker[11]
- Phishtank[12]
- Malwaredomainlist[13]
- Urlhaus[14]
- Firehol[15]
- CINS Score[16]

Whitelists

As described in section 2.4.2, the Alexa Top Domain list is often used to get a ground truth of legitimate domains. The list is no longer available to download for free, but there exists alternatives. We decided to use The Majestic Million[17], a list of the million domains with the most referring subnets. It contained a largely similar set of features as the Alexa Top Domain list. The list is available to download for free in CSV format. To reduce the likelihood of whitelisting malicious domains, only the top 1000 domain names are included in our implementation.

[11] https://cybercrime-tracker.net/
[12] https://www.phishtank.com/
[13] https://www.malwaredomainlist.com/
[14] https://urlhaus.abuse.ch/
[15] http://iplists.firehol.org/
[16] http://www.cinsscore.com/
[17] https://majestic.com/reports/majestic-million

IP Geolocation

Information about the corresponding AS number and ISP for each IP address was gathered from the Maxmind GeoLite2 free downloadable database[18]. Each record in the database contains a subnet, its AS number and the ISP that controls the AS.

Code listing 3.3: Record example from the Maxmind GeoLite2 database

```
1.1.1.0/24,13335,CLOUDFLARENET
```

3.8 Graph Data Model

The graph data model used for the databases in this thesis project is shown in figure 3.7. The nodes and relationships in the model were chosen based on how useful the information would be when trying to detect malicious network traffic. Table 3.1 shows all nodes and their properties, while table 3.2 shows the relationships between the nodes.

Table 3.1: Nodes and properties in the graph data model

Node	Property
IP	ip, blacklisted
Domain	name, blacklisted, whitelisted
Registrar	name
AS	number
ISP	name
Text	content
IP_HOST	ip
Mail_server	name, blacklisted
NXDOMAIN	

Figure 3.6 shows a subset of the nodes and relationships created from the CTU-13 dataset. An IP_HOST (yellow) has sent DNS request for several Domain nodes (red) which in turn have an assigned IP address (light blue) and a Registrar (green). Each IP node is connected to an AS (dark blue) which is administered by an ISP (grey).

3.8.1 Nodes

Table 3.1 shows the nodes that are created for each analyzed DNS query or response. This sections describe the information provided by each node type.

[18] https://dev.maxmind.com/geoip/geoip2/geolite2/

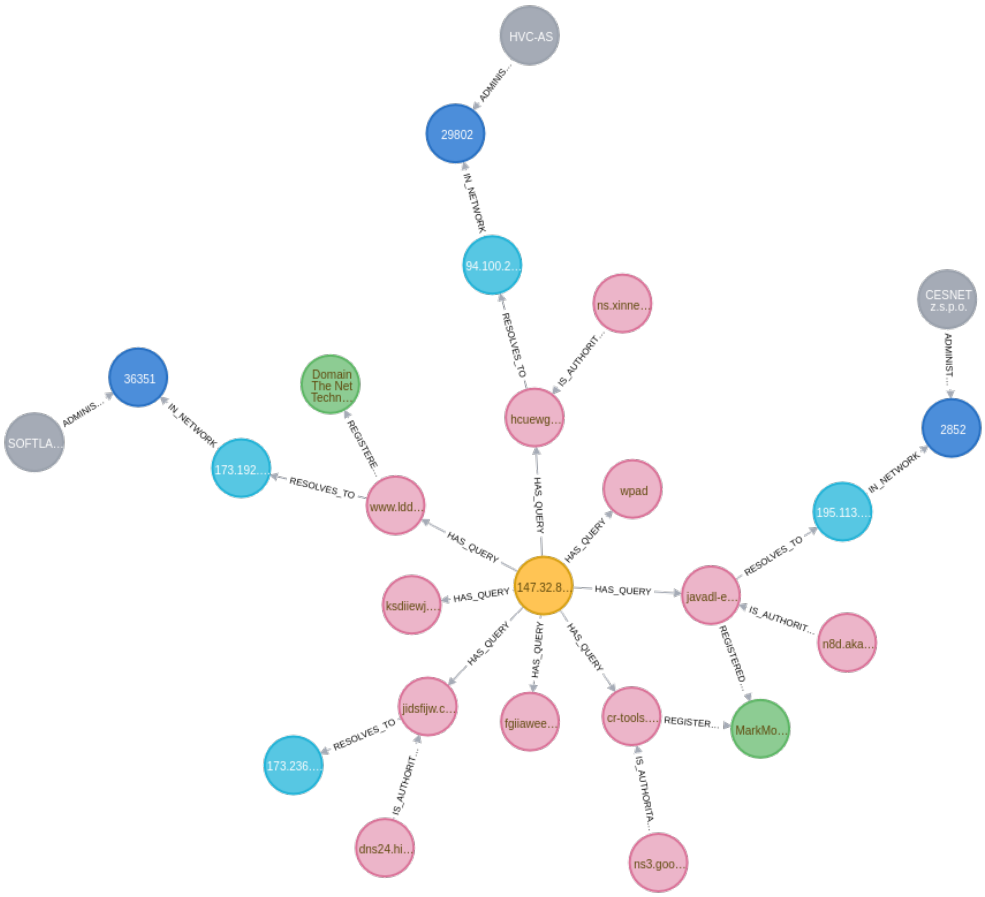


Figure 3.6: Sample graph from the CTU-13 dataset

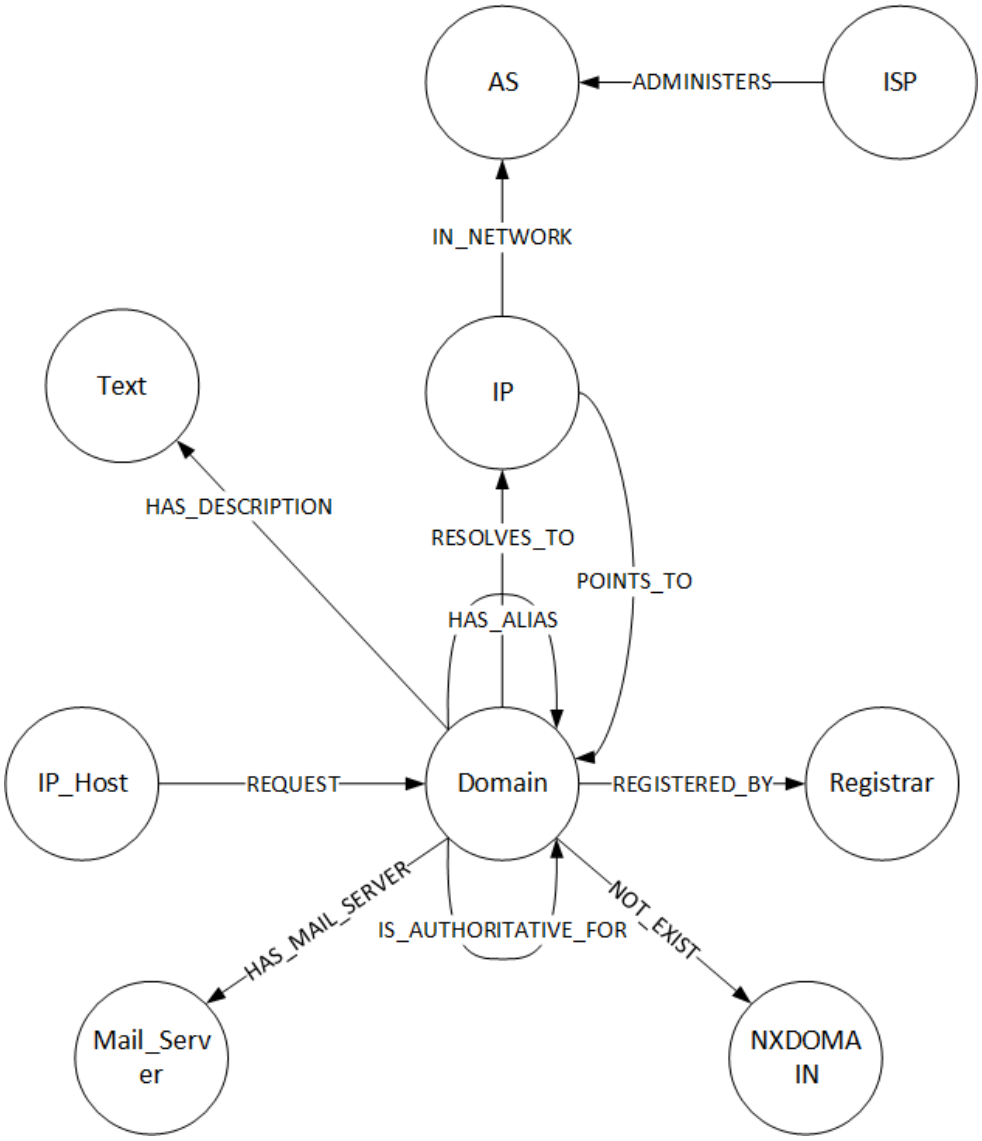


Figure 3.7: The graph data model used in this thesis

Table 3.2: Relationships in the graph data model

Relationship	Property	From ->To
RESOLVES_TO		Domain ->IP
REQUEST	first_seen, last_seen	IP_HOST ->Domain
REGISTERED_BY	creation_date	Domain ->Registrar
IN_NETWORK		IP ->AS
ADMINISTERS		ISP ->AS
POINTS_TO		IP ->DOMAIN
HAS_ALIAS		Domain ->Domain
IS_AUTHORITATIVE_FOR		Domain ->Domain
NOT_EXIST		Domain ->NXDOMAIN
HAS_MAILSERVER		Domain ->Mail_server
HAS_DESCRIPTION		Domain ->Text

IP

The IP node contains the IP address that the requested domain resolves to. The *IP* attribute is the A(IPv4) or AAAA(IPv6) record returned in the DNS query response. Since one domain name can resolve to several IP addresses, one IP node is created for each address. This way, it will be easier to see if several domains resolve to the same IP address, or if they did in the past. Each IP node also contains a *blacklisted* attribute that indicates whether the IP has been found in an IP blacklist.

Domain

The *Domain* node represents the requested domain name. Each node has a *blacklisted* attribute that indicates if the domain name has been found in any of the blacklists.

Registrar

A domain name registrar manages the use of domain names. Each *Domain* node is connected to a *Registrar* node if the information is available. Domain registry information is not always available, so not all *Domain* nodes are connected to a *Registrar* node. A recent report[19] by Awake Security reveals that certain registrars host a large number of malicious domains. It shows that 60% of the reachable domains registered through GalComm are malicious or suspicious. Domain registrar information can therefore be a useful indicator of malicious activity.

[19] `https://awakesecurity.com/blog/the-internets-new-arms-dealers-malicious-domain-registrars/`

AS

Each IP is a part of a routing prefix. An autonomous system is a collection of routing prefixes controlled by a common network administrator on behalf of a single administrative entity. Each *IP* node is therefore connected to an *AS* node. Each node has an AS number which a unique identifier assigned to each AS. By investigating if several *IP* nodes connect to the same *AS* node, we can potentially find relations between malicious domains that are not evident from the DNS request solely.

ISP

Each AS node is connected to an ISP node. An internet service provider (ISP) provides internet access to their customers. Each ISP controls one or more Autonomous Systems (AS).

Text

The Text node represents the DNS TXT record, which contains text that describes the domain it is connected to. The TXT record is frequently exploited for disseminating the commands of the botmaster, therefore the analysis of this record can reveal malicious actions.

IP_HOST

The IP_HOST represents the IP address of the client that send the DNS query for the domain name in the Domain node. If this node has issued many queries for blacklisted domains, there is a a possibility that the host is infected.

Mail_server

MX-Records specify the Mail Exchange servers for a given domain name. Each Mail_server node contains a *name* attribute. Mail servers connected to known malicious domains are often involved in spamming campains, namely they send large amounts of phishing mails. Thus this feature is a useful indicator for malicious actions.

NXDOMAIN

If a queried domain name does not exists, an NXDOMAIN response is returned. All Domain nodes containing non existant domain names are connected to a single NXDOMAIN node. Malicious domains are often taken down and re-hosted on different domain names. A large amount of DNS queries to *Domain* nodes connected to the NXDOMAIN node is considered suspicious behaviour.

3.8.2 Relationships

The nodes themselves and their attributes provide resourceful data, but this only becomes useful after the nodes have been connected based on some of the relationships. These relationships allow us to easily get an overview of the domain names and IP addresses, and how these are connected to each other. Most of the relationships in this data model represents connections between DNS RR types described in section 2.1.3.

REQUEST

The REQUEST relations represents a DNS request for a domain name. It connects the IP_HOST node (client) and the Domain node, and is characterized by the timestamps of the first and last observation of a DNS request for a given domain name.

RESOLVES_TO

If a domain name exists, it resolves to one or more IP addresses. This relationship represents the A (IPv4) or AAAA (IPv6) records returned in the DNS query response, and connects the Domain and IP nodes.

NOT_EXIST

When a queried domain name does not exist, an NXDOMAIN response code is returned. All Domain nodes that represent non-existent domain names have a NOT_EXIST relationship to the same NXDOMAIN node.

REGISTERED_BY

Connects each domain name to its registrar if the WHOIS information is available.

IN_NETWORK

Each IP address belongs to an Autonomous System (AS). All IP nodes are therefore connected to an AS node if this information is available.

ADMINISTERS

Each AS is administered by an Internet Service Provider (ISP), therefore each AS node is connected to an ISP node.

POINTS_TO

The POINTS_TO relationship represents the DNS PTR resource record, which are used to map IP addresses to domain names. The PTR resource record does not

necessarily return the same domain name resolved by the DNS request, but it can reveal useful information about the IP address, for example if it is a DSL host that is possibly infected.

HAS_ALIAS

The HAS_ALIAS relationship represents the CNAME records for each domain name. These records are domain name aliases, and allow several domain names to point to the same domain name. Analyzing CNAME records is useful because domain names are often changed rapidly. If one domain name is found in a blacklist the other CNAME records connected to the same domain should be investigated.

IS_AUTHORITATIVE_FOR

This relationship represents the NS-record, which specifies which DNS server is responsible for a zone. Several malicious Domain records might have the same Authoritative name server, which makes this a useful relationship.

HAS_MAILSERVER

This relationship represents the MX record for a given domain name, and connects Domain and Mail_Server nodes.

HAS_DESCRIPTION

This relationships connects each Domain node with its corresponding TXT node.

Chapter 4

Results

This chapter presents the outcomes of our research after creating the databases of the graph data model, and running queries on the databases to find potentially malicious domain names and IP addresses. We also show how our model can be used to detect potential clusters of malicious activity by examining graph nodes related with blacklisted nodes or other malicious nodes.
The log files were imported into Neo4j using the Python program described in section 3.2. The evaluations were performed on a computer with the following specifications:

- Desktop computer with AMD Ryzen 5 3600X, 16GB RAM and Ubuntu 18.04 as operating system.
- Neo4j version 4.0.1 and Neo4j Desktop version 1.2.9

4.1 Importing Log Data

The values necessary for creating the nodes and relationships were retrieved directly from the Eidsiva log file, which was a plain text file (TXT). When creating the database from the CTU-13 dataset, a different approach was used since the log file was in pcap file format. Each packet from the packet capture file was read using PyShark, where only the DNS traffic was retrieved. The necessary resource record fields from each packet were extracted and stored in a Python dictionary. Registrant data from WHOIS was also added, in addition to the Autonomous System (AS) and ISP related to each IP address. The dictionary was then fed to the official Neo4j Python driver, which executed transactions based on several Cypher queries.

4.2 Eidsiva Dataset Findings

The Eidsiva dataset only contains the DNS traffic between each host and the resolver. Therefore the graph data model looked different than the one created from the CTU-13 dataset. It contains Domain, IP_HOST and Registrar nodes, as well as

the HAS_QUERY and REGISTERED_BY relationships. While this meant that we were unable to see what IP addresses each domain name resolved to, there was still a large amount of useful information to be found in the queries between each host and domain name. An example graph is shown in figure 4.1 with all of the available node types in this database:

- IP_HOST (yellow)
- Domain (red)
- Registrar (green)

Figure 4.1: Example graph from Eidsiva database with all node types

4.2.1 Querying the Eidsiva Database

Figure 4.2 shows a graph generated by the query found in code listing 4.1. It shows 20 Domain nodes that have queried the same domain (red), in this case *www.google.com*. This particular domain name is well known and therefore whitelisted, meaning that this graph cluster is certainly benign.

Code listing 4.1: Finds all clients that have sent DNS queries for 'google.com'

```
MATCH (i:IP_HOST)-[q:HAS_QUERY]->(d:Domain{name:"www.google.com"})
RETURN i,q,d
LIMIT 20
```

Since this dataset contains the time and date of each DNS query, it is possible to use this to search for queries made in specific time intervals. One possible use case is to search for the most queried domains in a given day:

Code listing 4.2: Cypher code used to find most queried domain names on May 4, 2020

```
MATCH(src:IP_Host)-[hq:HAS_QUERY {date: "04-May-2020"}]->(d:Domain)
RETURN DISTINCT d.name as dns_query, COUNT(hq) AS total ORDER BY total DESC
LIMIT 10
```

Code listing 4.3: Cypher query that returns all blacklisted domain names

```
MATCH (n:Domain {blacklisted:true}) return n.name
```

Figure 4.2: Queries from 20 different hosts to the same domain in the Eidsiva dataset

"dns_query"	"total"
"www.google.com"	601
"outlook.office365.com"	526
"www.microsoft.com"	470
"api-global.netflix.com"	407
"gateway.fe.apple-dns.net"	407
"mobile.pipe.aria.microsoft.com"	340
"presence.teams.microsoft.com"	338
"dns.msftncsi.com"	323
"cf-st.sc-cdn.net"	318
"e6858.dsce9.akamaiedge.net"	312

Figure 4.3: Top domains queried on May 4, 2020

Each domain has a "blacklisted" attribute which is generated by checking if the associated domain name is found in any of the blacklists described in 3.7.2. By running the query in code listing 4.3 we can find all domain names that have been blacklisted:

- .
- lan
- t.co
- m.me
- UBNT
- io
- ath6
- dev
- t.me
- null
- com
- no
- g.co
- net
- HPPC
- a.co
- wpad

Code listing 4.4: Cypher query that returns all domain names that are both blacklisted and whitelisted

```
MATCH (n:Domain {blacklisted:true, whitelisted:true}) return n.name
```

Most of the domain names found are clear false positives, and some are popular URL shorteners. By running the query shown in 4.4 we found that all of the following were found in both a blacklist and a whitelist:

- t.co
- m.me
- t.me
- g.co

The fact that the domain names were found in the whitelist means that they are widely used. Therefore, they are most likely not malicious. If we want to find domain names that might be related to the ones that are blacklisted, we can run queries to find relations between them. One possibility is to see if several hosts have sent DNS request for the same domain name, like in figure 4.2. We can also find out what other domains the same hosts have visited. Figure 4.4 was generated by the query in code listing 4.6 and shows two other hosts that have queried the same domain name, in this case *t.co*. Since they all have requested a blacklisted domain name, there is a possibility that all of the host have been infected. Each host in this graph is also the center of a cluster of queried domain names that are potentially malicious.

If one host has queried several malicious domain names, it might have downloaded malware. Another type of malicious activity is cases where blacklisted sites are visited frequently. This can mean that the host has been infected and is part of a botnet or malware distribution network, and is sending and receiving data from an external server such as a C&C server. By running a the query in code listing 4.5 we found that a client with address *48a3db8e7657277aee70a496d0413241f956cbd2* had sent 246 queries to *t.co*. This abnormally high number indicated that this client should be investigated further to see if it has been infected.

Code listing 4.5: Cypher query that returns the 10 IP_HOST nodes that have sent the most queries for *t.co*

```
MATCH (src:IP_HOST)-[:HAS_QUERY]->(:Domain {name: "t.co"})
MATCH (src)-[hq:HAS_QUERY]->(d:Domain)
RETURN DISTINCT src.ip AS dns_query, count(hq) AS total ORDER BY total DESC
LIMIT 10
```

Code listing 4.6: Queried domain names for clients that have queried blacklisted domain name

```
MATCH (src:IP_HOST)-[:HAS_QUERY]->(:Domain {name: "t.co"})
MATCH (src)-[:HAS_QUERY]->(d:Domain) RETURN src, d LIMIT 25
```

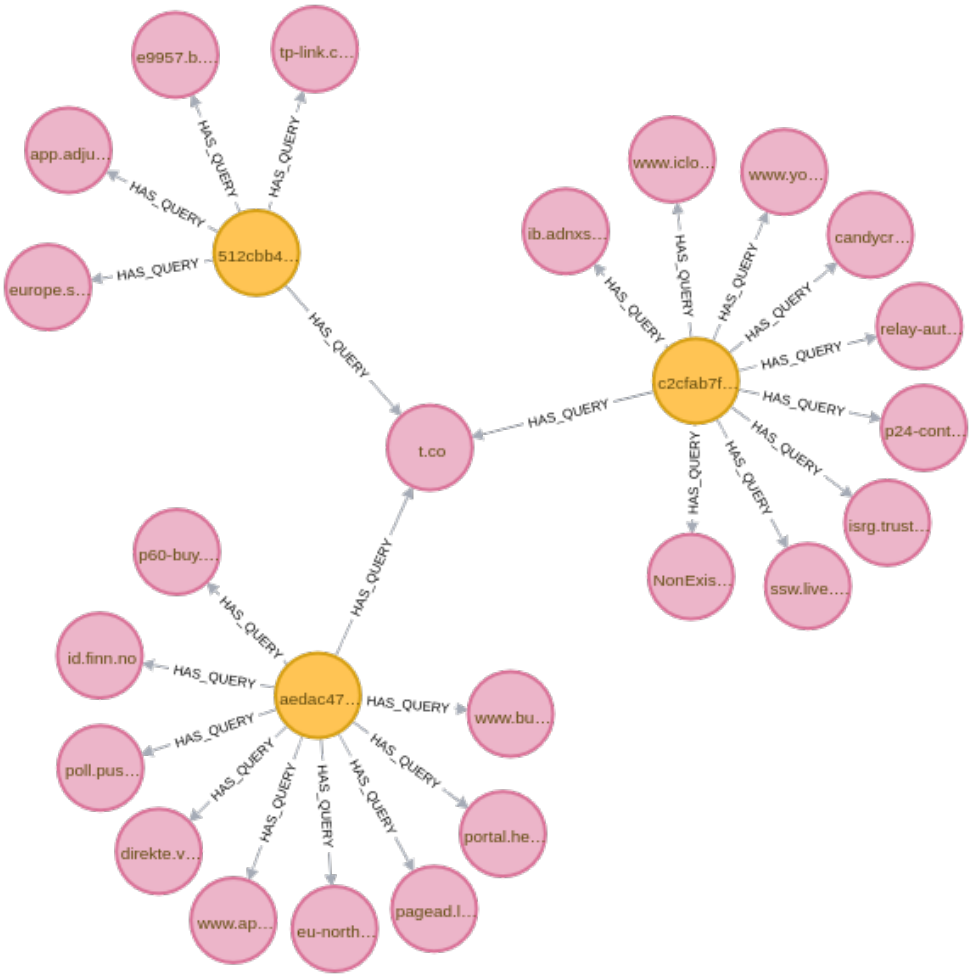


Figure 4.4: Hosts that have sent DNS queries for the same blacklisted domain name

Another useful feature, especially for security analysts, is the ability to find the time stamps for the first and last request to a given domain name. Also, domain

names related to malicious activity are often used only for limited periods. Botnets usually use many different domain names for their C&C servers in order to have redundancy. The large amount means that they are expendable and can be changed frequently. In the case of a computer infected by malware, this is a useful indicator of malicious activity. If the last query was sent recently, the machine is most likely still infected.

4.3 CTU-13 Dataset Findings

The CTU-13 dataset contains data about DNS RR types and content for each captured network packet. This allowed us to fully utilize our graph data model and display packet data combined with data from external sources. Figure 4.5 shows an example of a single DNS query where all of the node types are present. The different node types are color coded to make them easily distinguishable:

- IP_HOST (orange)
- Domain (red)
- IP (light blue)
- AS (dark blue)
- ISP (gray)
- Mail_Server (light brown)
- Registrar (green)

The data model also contains Text nodes, but no DNS TXT RRs were found in this dataset. Also, there are no NXDOMAIN nodes present due to the absence of NXDOMAIN responses in the logs.

Code listing 4.7: Returns all available node types in the CTU-13 dataset

```
MATCH(src:IP_HOST)-[:HAS_QUERY]-(d:Domain)
MATCH(d)-[:RESOLVES_TO]->(i:IP)
MATCH(i)-[:IN_NETWORK]->(as:AS)<-[:ADMINISTERS]-(isp:ISP)
MATCH(d)-[:HAS_MAIL_SERVER]->(mx:Mail_Server)
MATCH(d)-[:REGISTERED_BY]->(r:Registrar)
RETURN src,d,i,as,isp,mx,r LIMIT 2
```

4.3.1 Querying the CTU-13 Database

Same Registrar

Domains names that are planned to be used as C2 servers in botnets are often bulk registered in advance. Normally, all of the domain names are registered by the same registrar.
If one of the Domain nodes in this cluster represents a blacklisted domain name, the other Domain nodes in the cluster might be malicious as well. Each REGISTERED_BY relationships contains a creation_date attribute, which shows when the domain name was first registered. By comparing the different creation dates in the cluster, we can potentially find domain names that are related to the

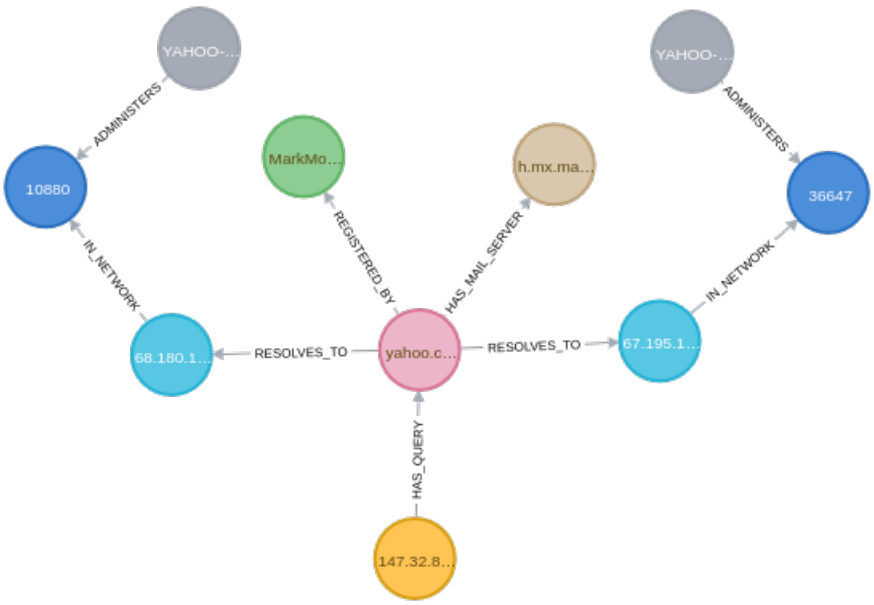


Figure 4.5: Example DNS query that contains all available node types

same type of activity. The Cypher query in code listing 4.8 finds domain nodes that have been queried by a single host, and their registrar. The resulting graph is shown in figure 4.6. As described in section 2.3.1, domain names related to botnets and malware distribution networks are often registered by the same registrars. Registrars are therefore a useful feature when analyzing performing DNS log analysis. If one of the domain names are malicious, there is a chance that others with the same registrar are also malicious.

Code listing 4.8: Queried domain names for clients that have queried blacklisted domain names

```
MATCH (src:IP_HOST)-[hq:HAS_QUERY]->(d:Domain)-[rb:REGISTERED_BY]-(r:Registrar)
RETURN src,hq,d,r LIMIT 10
```

Same IP Addresses

As described, malicious domain names are often changed rapidly. Therefore it is useful to see all of the domain names that resolve to each IP address. The Cypher query in code listing 4.9 returns domain names and the IP addresses they resolve to. Figure 4.7 shows a subset of the resulting graph, where several domain names resolve to the same IP addresses. Figure 4.8 shows another subset where a single domain name (smtp.aol.com) resolves to several IP addresses.

Code listing 4.9: Finds mappings between domain names and IP addresses

```
MATCH (d:Domain)-[r:RESOLVES_TO]->(i:IP) RETURN d, i LIMIT 100
```

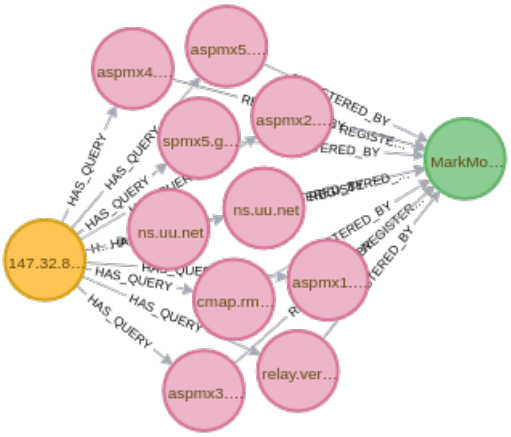


Figure 4.6: Queried domain names with the same registrar

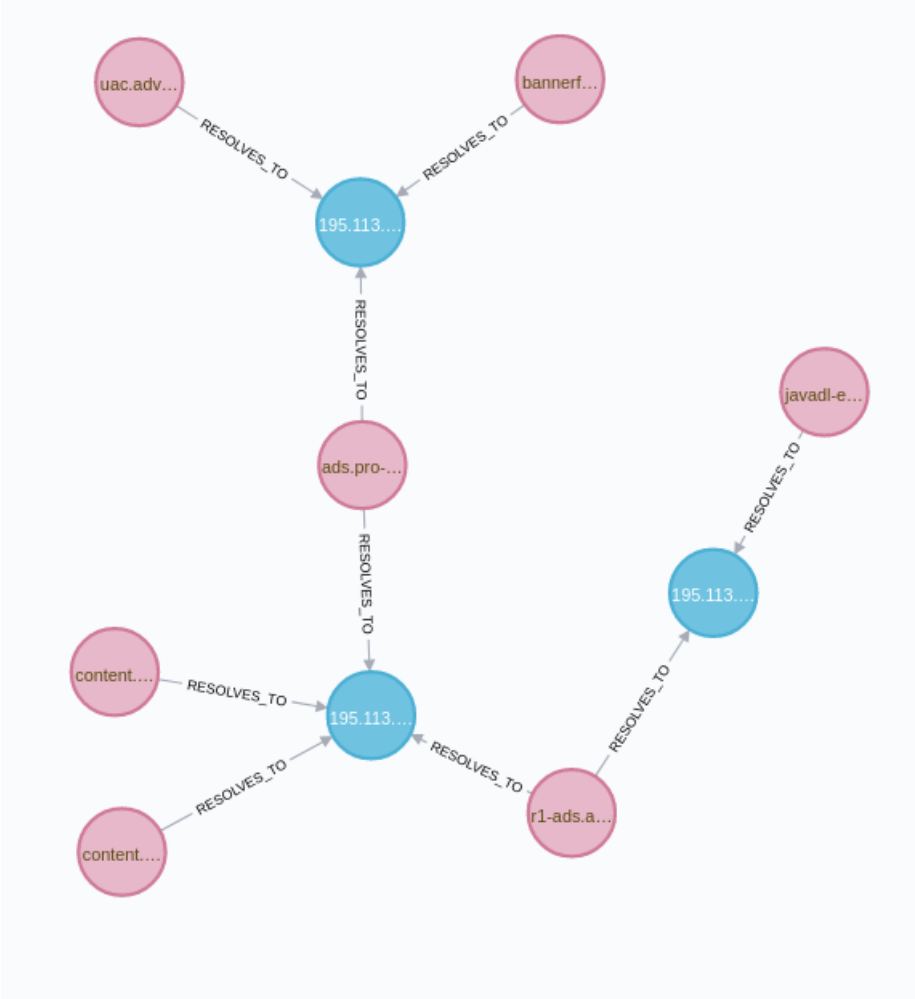


Figure 4.7: Domain names (red) that resolve to the same IP address (blue)

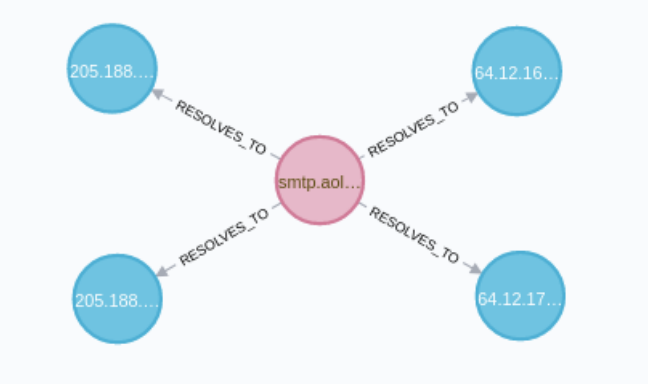


Figure 4.8: Single domain name (red) that resolves to several IP addresses (blue)

Malicious IP Addresses

This dataset contains not only domain names, but also the IP addresses they resolve to. No domain names in this dataset were blacklisted, but the query in code listing 4.10 returned the blacklisted IP addresses 74.220.207.177 and 127.0.0.1. The query in code listing 4.11 returned the graph in figure 4.9, which gave two starting points for an investigation. The left graph shows that the domain name *realto-house.ru* resolved to the IP 74.220.207.177. The two node clusters have significant differences. The right cluster contains a number of domains with a .ru country code top-level domain (ccTLD), which means that they are registered in Russia. Russian domain names are sometimes used for malicious activity, so this is a reason for further investigation. A suspicious detail is that all of the domain names resolve to 127.0.0.1, which is the loopback IP address also known as localhost. This address is used when connecting to the same computer the end user is on. Malware can use this to avoid suspicion, for example by resolving to it before or after malicious activities in order to appear legitimate.
The left cluster shows that the only domain names that has resolved to the IP address 74.220.207.177 is "realto-house.ru". By running the query in code listing 4.12, we found all nodes that were connected to the IP node with address 74.220.207.177. The result is shown in graph 4.10, where we found additional nodes that were potentially malicious. The complete set of information found can be summarized as:

- realto-house.ru resolves to 74.220.207.177. No other domain names resolved to this particular IP.
- realto-house.ru has a mail server with the same name.
- ns1.vega-host.com is the authoritative name server for realto-house.ru.
- A host with IP 147.32.84.165 has sent queries for realto-house.ru

Code listing 4.10: Returns all blacklisted IP addresses

```
MATCH(i:IP) WHERE i.blacklisted = TRUE RETURN i
```

Code listing 4.11: Returns all domain names that resolve to blacklisted IP addresses

```
MATCH(i:IP) WHERE i.blacklisted = TRUE
MATCH (d:Domain)-[:RESOLVES_TO]->(i)
RETURN i,d
```

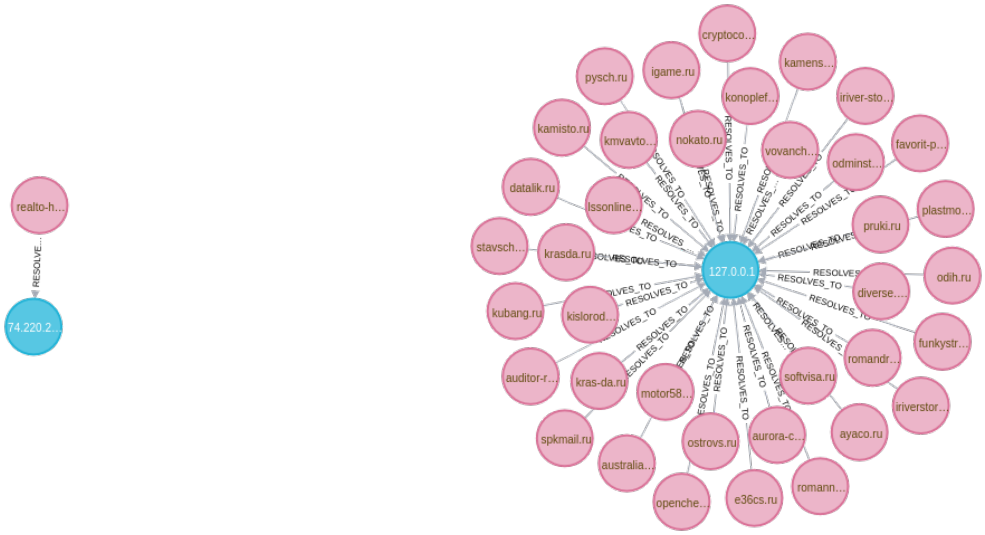


Figure 4.9: All domain names that resolve to the blacklisted IP addresses

Code listing 4.12: Returns all nodes that are connected to the IP node with ip = 74.220.207.177

```
MATCH(i:IP) WHERE i.ip = "74.220.207.177"
MATCH (d:Domain)-[:RESOLVES_TO]->(i)
MATCH (n)-[r]-(d)
RETURN i,d,n
```

Malicious Domain Registrars

Malicious domain names that are used for the same type of activity are often registered by the same domain registrar. Going back to the right cluster in 4.9, we ran the query in code listing 4.13 and found that many of the domain names were registered by the same registrar, as shown in figure 4.11.

Code listing 4.13: Returns all Domain and Registrar nodes connected to the IP node with ip=127.0.0.1

```
MATCH(i:IP) WHERE i.blacklisted = TRUE
MATCH (d:Domain)-[:RESOLVES_TO]-(i)
MATCH (d)-[:REGISTERED_BY]->(r:Registrar)
RETURN i,d,r
```

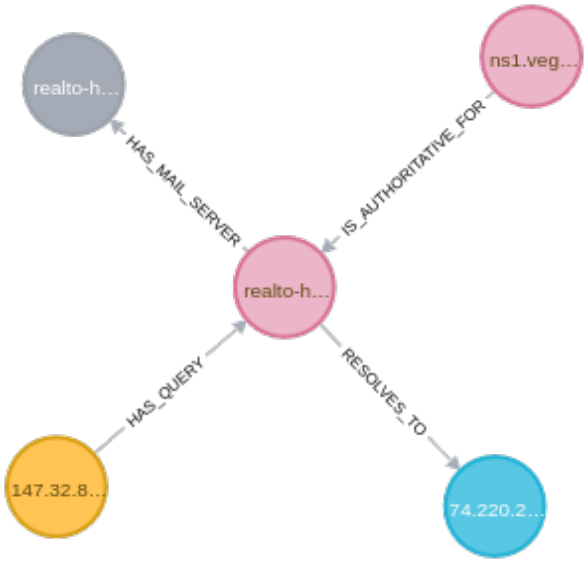


Figure 4.10: All nodes connected to blacklisted IP

AS and ISP Information

In order to find information about a subset of IP, AS and ISP nodes, we ran the query in code listing 4.14. The result was the two clusters shown in figure 4.12. The right cluster shows domain names and IPs related to a subnet controlled by Google. This cluster can therefore be considered benign. The ISP in the right cluster is *CT-HangZhou-IDC*. None of the domain names or IPs in this cluster were blacklisted, but open-source intelligence (OSINT) revealed that this ISP is known to host malware. Therefore, this cluster can be considered malicious.

Code listing 4.14: Returns all Domain and Registrar nodes connected to the IP node with ip=127.0.0.1

```
MATCH(ip:IP)-[:IN_NETWORK]->(as:AS)<-[:ADMINISTERS]-(isp:ISP)
RETURN ip,as,isp
LIMIT 10
```

Figure 4.11: Registrars related to potentially malicious domain names

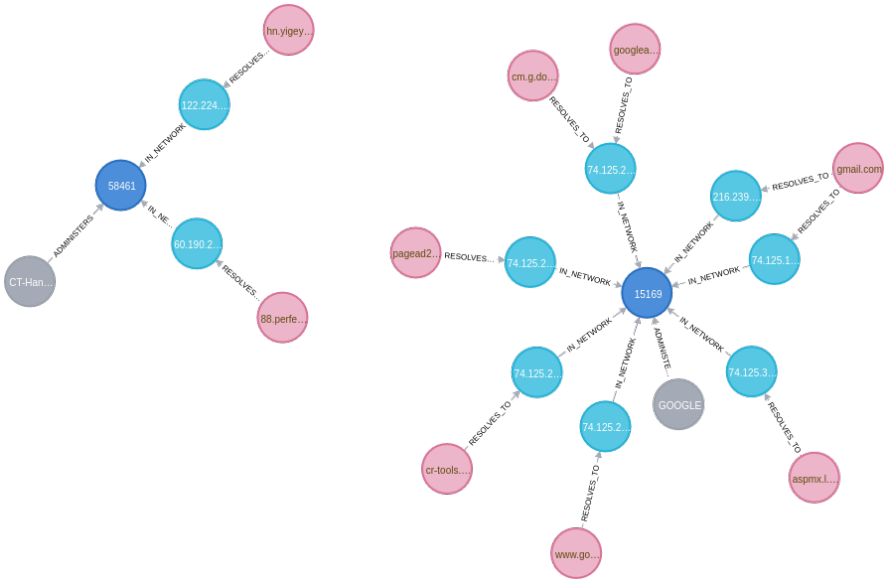


Figure 4.12: IP addresses owned by two ISPs

Chapter 5

Discussion

This chapter includes a discussion of the findings for each of the research questions defined for this thesis project. Potential issues related to the methodology and findings are also discussed. Lastly, ethical and legal considerations are described.

5.1 RQ1: Graph Data Model

The first research question, *How can DNS data be represented in a graph database?*, aimed to create a graph data model which could be used to store both typical network features of malicious domain names and explore the usefulness of new ones. During the problem investigation phase, related works were studied to find out what types of graph models had been developed. We also wanted to investigate the usefulness of the domain features used in each approach. We found that a majority of the existing works in this area of study use only the contents of DNS RRs in their models. Also, only a few approaches used graph databases such as Neo4j to store the data. We saw a potential to use graph database technology to store both contents of RRs and enrichment data from various sources.
A set of DNS RRs were based on their prevalence in normal and malicious traffic patterns. The main challenge was to find DNS datasets that contained all of the necessary RRs and were available for research purposes. This was challenging, largely due to the personal information contained in DNS traffic. Client IP addresses and queried domain can be used to build profiles of end users. Because of this, we used datasets that were anonymized or cleared of personally identifiable information.
Developing and importing data into the graph data model was the main objective of this thesis project. Our final model shows that it is possible to create graph representations of DNS data in a reasonable time frame, mainly due to the developer resources available in Neo4j and easily formatted Cypher queries. The temporal and spatial properties were suitable for use in a graph representation. Through several experiments we found that our model performed well as a tool for detection and threat hunting.

5.2 RQ2: Incorporating External Data

The second research question, *How can data from external sources be incorporated into the graph structure?*, builds upon the first and presented a similar set of challenges and findings. Domain registrar information proved to be a useful feature when determining the maliciousness of a domain. When shown together with domain names and IP addresses, it resulted in relations that were not apparent in the original DNS traffic. Analysis of AS and ISP nodes showed promising results that could be used to find malicious clusters of IPs based on AS or ISP information.

5.3 RQ3: Detecting Malicious Domains

Like many related works described in chapter 2, our detection algorithm relies on enrichment data from blacklists and whitelists. Our experiments show that the model can detect both malicious domain names and IPs, and use this to infer relationships to other nodes in the graph. These relationships often formed clusters of suspicious domains that could reveal evasion techniques used by malware to avoid network security monitoring tools such as IDSs. The set of features in our graph data model could also be suitable for machine learning techniques such as deep learning or clustering. However, this was outside the scope of this thesis project.
One issue with this approach is that the data import process is time consuming. Thus, not all domain names in the datasets could be analyzed. More malicious domain names and IPs might have been discovered if the entire dataset could have been analyzed. Also, a different set of blacklists might have resulted in a different set of suspicious domains.

5.4 Potential issues

DNS over HTTPS (DoH) has become more popular lately, and popular web browsers such as Firefox have started rolling out DoH as the default setting in certain regions[1]. While this improves privacy, it poses several issues for security information and event management (SIEM) systems that monitor DNS traffic [39]. When DoH is enabled, the DNS traffic is sent over port 443 and secured via TLS. This means that important information such as DNS RRs are unavailable to network security monitoring tools.

5.5 Ethical and Legal Considerations

Careful consideration was taken when collecting DNS traffic logs. Personally identifiable information was removed or anonymized. Freely available datasets were

[1] `https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/`

checked to ensure that they contained no such information. Information from each end user, such as IP addresses, was anonymized. This did lead to less representative datasets with fewer features, but it was still possible to test our model.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we explored the use of graph databases to analyze DNS traffic logs and find relations between malicious domain names and IP addresses. A graph data model was built using commonly used DNS resource records and enrichment information from resources such as WHOIS and Maxmind. DNS traffic logs were retrieved from two different sources and used to create two versions of the graph data model, to see how the model would work on different log formats.
Botnets and malware distribution networks are increasing threats in the cyber security domain. The DNS protocol is used to develop evasion techniques such as fast flux, that are designed to make malicious traffic and malware more difficult to detect and stop. Domain names and IP addresses are changed rapidly, which is hard to detect with signature-based intrusion detection systems. Graph databases present the ability to see how the domain names and IP addresses are connected. The data model can be used to both detect malicious activity, and analyze logs. When testing our graph data model in several scenarios, we found that it was possible to discover new relations within the DNS logs. This makes it useful in environments such as Security Operations Centers (SOC) where log file analysis is an important part of detecting malicious activity. The Neo4j database can be queried quickly with queries that are easy to understand and write. In addition, answers to queries are received quicker than in relational database.

6.2 Future Work

While the graph databases created in this work have potential for use in both network security monitoring and malware analysis, there exists possibilities to extend the work. The performance of the implemented log file import program proved to be sufficient for smaller datasets of a limited time span and with few querying hosts. However, when used on a dataset provided by Eidsiva, with a large amount of DNS queries, it proved to be too inefficient for real-time analysis of network

packets. To be able to monitor large networks, the code needs to be optimized. This work used only a single instance of Neo4j when creating the graph databases. One possibility for performance improvement is to cluster several instances to import data from several sources in parallel. Operational security (OPSEC) could also be improved by changing the way information is retrieved from external data sources. This could prevent malware authors from being alerted that their activity has been detected. There is also a possibility to include data from other sources, such as HTTP logs, and correlate them with DNS logs. This has the potential to provide a better understanding of malware behaviour and infection methods.

Bibliography

[1] Cisco, 'Cisco 2016 annual security report', 2016.

[2] M. Anagnostopoulos, G. Kambourakis and S. Gritzalis, 'New facets of mobile botnet: Architecture and evaluation', *International Journal of Information Security volume*, vol. 15, pp. 455–473, 2016.

[3] N. Inc., 'Neo4j graph platform', 2019. [Online]. Available: `https://neo4j.com/`.

[4] P. V. Mockapetris, *Rfc1034: Domain names-concepts and facilities*, 1987.

[5] P. V. Mockapetris, *Rfc1035: Domain names-implementation and specification*, 1987.

[6] M. Anagnostopoulos, G. Kambourakis, E. Konstantinou and S. Gritzalis, 'Dnssec vs. dnscurve: A side-by-side comparison', in. IGI Global, 2012, p. 201.

[7] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose, 'Dns security introduction and requirements', RFC 4033 (Proposed Standard), Tech. Rep., 2005.

[8] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose, 'Resource records for the dns security extensions', RFC 4034 (Proposed Standard), Tech. Rep., 2005.

[9] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose, 'Protocol modifications for the dns security extensions', RFC 4035, March, Tech. Rep., 2005.

[10] P. Hoffman and P. McManus, 'Dns queries over https (doh)', *Internet Requests for Comments, IETF, RFC*, vol. 8484, 2018.

[11] P. Vixie, O. Gudmundsson 3rd and B. Wellington, *Rfc2845: Secret key transaction authentication for dns (tsig)*, 2000.

[12] L. Shafir, Y. Afek and A. Bremler-Barr, 'Nxnsattack: Recursive dns inefficiencies and vulnerabilities', *arXiv preprint arXiv:2005.09107*, 2020.

[13] S. S. Silva, R. M. Silva, R. C. Pinto and R. M. Salles, 'Botnets: A survey', *Computer Networks*, vol. 57, no. 2, pp. 378–403, 2013.

[14] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis and S. Gritzalis, 'DNS Amplification Attack Revisited', *Computers & Security*, vol. 39, Part B, pp. 475–485, 2013.

[15] J. Nazario and T. Holz, 'As the net churns: Fast-flux botnet observations', in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, 2008, pp. 24–31.

[16] G. Wang, J. W. Stokes, C. Herley and D. Felstead, 'Detecting malicious landing pages in malware distribution networks', in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2013, pp. 1–11.

[17] L. Invernizzi, S. Miskovic, R. Torres, C. Kruegel, S. Saha, G. Vigna, S.-J. Lee and M. Mellia, 'Nazca: Detecting malware distribution in large-scale networks.', in *NDSS*, Citeseer, vol. 14, 2014, pp. 23–26.

[18] G. Zhao, K. Xu, L. Xu and B. Wu, 'Detecting apt malware infections based on malicious dns and traffic analysis', *IEEE access*, vol. 3, pp. 1132–1142, 2015.

[19] Y. Zhauniarovich, I. Khalil, T. Yu and M. Dacier, 'A survey on malicious domains detection through dns data analysis', *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 67, 2018.

[20] D. Chiba, T. Yagi, M. Akiyama, T. Shibahara, T. Yada, T. Mori and S. Goto, 'Domainprofiler: Discovering domain names abused in future', in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2016, pp. 491–502.

[21] L. Bilge, E. Kirda, C. Kruegel and M. Balduzzi, 'Exposure: Finding malicious domains using passive dns analysis.', in *Ndss*, 2011, pp. 1–17.

[22] L. Daigle, 'Whois protocol specification', *RFC 3912 (Draft Standard)*, 2004.

[23] M. Stevanovic, J. M. Pedersen, A. D'Alconzo, S. Ruehrup and A. Berger, 'On the ground truth problem of malicious dns traffic analysis', *computers & security*, vol. 55, pp. 142–158, 2015.

[24] R. Sharifnya and M. Abadi, 'Dfbotkiller: Domain-flux botnet detection based on the history of group activities and failures in dns traffic', *Digital Investigation*, vol. 12, pp. 15–26, 2015.

[25] A. Berger, A. D'Alconzo, W. N. Gansterer and A. Pescapé, 'Mining agile dns traffic using graph analysis for cybercrime detection', *Computer Networks*, vol. 100, pp. 28–44, 2016.

[26] A. Berger and W. N. Gansterer, 'Modeling dns agility with dnsmap', in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2013, pp. 387–392.

[27] C. Peng, X. Yun, Y. Zhang, S. Li and J. Xiao, 'Discovering malicious domains through alias-canonical graph', in *2017 IEEE Trustcom/BigDataSE/ICESS*, IEEE, 2017, pp. 225–232.

[28] J. Lee and H. Lee, 'Gmad: Graph-based malware activity detection by dns traffic analysis', *Computer Communications*, vol. 49, pp. 33–47, 2014.

[29] L. Diederichsen, K.-K. R. Choo and N.-A. Le-Khac, 'A graph database-based approach to analyze network log files', in *International Conference on Network and System Security*, Springer, 2019, pp. 53–73.

[30] F. Zou, S. Zhang, W. Rao and P. Yi, 'Detecting malware based on dns graph mining', *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 102 687, 2015.

[31] I. Khalil, T. Yu and B. Guan, 'Discovering malicious domains through passive dns data graph analysis', in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ACM, 2016, pp. 663–674.

[32] S. Yadav and A. N. Reddy, 'Winning with dns failures: Strategies for faster botnet detection', in *International Conference on Security and Privacy in Communication Systems*, Springer, 2011, pp. 446–459.

[33] N. Jiang, J. Cao, Y. Jin, L. E. Li and Z.-L. Zhang, 'Identifying suspicious activities through dns failure graph analysis', in *The 18th IEEE International Conference on Network Protocols*, IEEE, 2010, pp. 144–153.

[34] T. W. Edgar and D. O. Manz, *Research methods for cyber security*. Syngress, 2017.

[35] B. M. Sasaki and J. Chao, *Graph Databases for Beginners*. Neo4j, 2018.

[36] R. V. Bruggen, *Learning Neo4j*. Packt Publishing, 2014.

[37] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer and A. Taylor, 'Cypher: An evolving query language for property graphs', in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1433–1445.

[38] S. Garcia, M. Grill, J. Stiborek and A. Zunino, 'An empirical comparison of botnet detection methods', *computers & security*, vol. 45, pp. 100–123, 2014.

[39] A. Fidler, 'Potential isp challenges with dns over https', 2019. [Online]. Available: `https://indico.uknof.org.uk/event/46/contributions/668/attachments/898/1109/UKNOF43_Potential_ISP_challenges_with_DNS_over_HTTPS_Issue_1A_050419.pdf`.

Appendix A

Source Code

This section contains all Python functions that were used to create the graph databases.

A.1 Python Functions

Code listing A.1: Creates nodes and relationships in Neo4j

```
def create_graph(tx, cap):
    tx.run("MERGE (d:Domain {name: $host, blacklisted: $in_blacklists,
    whitelisted: $whitelisted}) ",
           {"host": cap['host'], "src": cap['src'],
           "in_blacklists": cap['in_blacklists'],
            "whitelisted": cap['whitelisted']})
    if cap['cname'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (a:Domain {name: $cname}) "
               "MERGE (d)-[:HAS_ALIAS]->(a)",
               {"host": cap['host'], "cname": cap['cname']})
    if cap['ns'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (n:Domain {name: $ns}) "
               "MERGE (n)-[:IS_AUTHORITATIVE_FOR]->(d)",
               {"host": cap['host'], "ns": cap['ns']})
    if cap['src'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (i_src:IP_HOST {ip: $src}) "
               "MERGE (i_src)-[p:HAS_QUERY]->(d)",
               {"src": cap['src'], "host": cap['host']})
    if cap['txt'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (t:TXT {content: $txt})"
               "MERGE (d)-[:HAS_DESCRIPTION]->(t)",
               {"host": cap['host'], "txt": cap['txt']})
    if cap['nxdomain'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (n:NXDOMAIN) "
               "MERGE (d)-[:NOT_EXIST]->(n)",
               {"host": cap['host']})
    if cap['dst'] is not None:
```

```
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (i:IP {ip: $dst, blacklisted: $blacklisted}) "
               "MERGE (d)-[:RESOLVES_TO]->(i)",
               {"host": cap['host'], "dst": cap['dst'],
               "blacklisted": check_ip(cap['dst'])})
        if cap['dst'] is not None:
            tx.run("MATCH (d:Domain {name: $host}) "
                   "MATCH (i:IP {ip: $dst}) "
                   "MATCH (d)-[p:RESOLVES_TO]->(i) "
                   "SET p.time = $time",
                   {"host": cap['host'], "dst": cap['dst'], "time": cap['time']})
    if cap['ptr'] is not None:
        tx.run("MATCH (i:IP {ip: $ip}) "
               "MERGE (d:Domain {name: $ptr}) "
               "MERGE (i)-[:POINTS_TO]->(d)",
               {"ip": cap['dst'], "ptr": cap['ptr']})
    if cap['registrar'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (r:Registrar {name: $registrar}) "
               "MERGE (d)-[p:REGISTERED_BY]->(r) "
               "SET p.creation_date = $creation_date",
               {"registrar": cap['registrar'], "host": cap['host'],
               "creation_date": cap['creation_date']})
    if cap['mx'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MERGE (m:Mail_Server {name: $mx}) "
               "MERGE (d)-[:HAS_MAIL_SERVER]->(m)",
               {"host": cap['host'], "mx": cap['mx']})
    if cap['timestamp'] is not None:
        tx.run("MATCH (d:Domain {name: $host}) "
               "MATCH (i:IP_HOST {ip: $src}) "
               "MATCH (i)-[p:HAS_QUERY]->(d) "
               "SET p.last_seen = $time",
               {"host": cap['host'], "src": cap['src'], "time": cap['timestamp']})
        tx.run("MATCH (d:Domain {name: $host}) "
               "MATCH (i:IP_HOST {ip: $src}) "
               "MATCH (i)-[p:HAS_QUERY]->(d) WHERE NOT EXISTS(p.first_seen) "
               "SET p.first_seen = $time",
               {"host": cap['host'], "src": cap['src'], "time": cap['timestamp']})
            if cap['asn'] is not None:
        tx.run("MATCH (i:IP {ip: $dst}) "
               "MERGE (as:AS {number: $asn}) "
               "MERGE (i)-[:IN_NETWORK]->(as) "
               "MERGE (isp:ISP {name: $isp}) "
               "MERGE (isp)-[:ADMINISTERS]->(as)",
               {"dst": cap['dst'], "asn": cap['asn'], "isp": cap['isp']})
```

Code listing A.2: Creates dictionary with values from log file and passes it to create_graph

```
def log_to_dict(filename):
    cap = pyshark.FileCapture(filename)
    filetype = filename.split(".")[1]
    if filetype == 'pcap':
        for packet in cap:
            if 'DNS' in packet:
                src = None
                if packet.dns.flags_response == '0':
                    src = packet.ip.src
```

```
                whois_result = check_whois(packet.dns.qry_name)
                geo_result = None
                packet_dict = {'trans_id': packet.dns.id, 'src': src, 'dst': None,
                               'host': packet.dns.qry_name,
                               'qry_type': packet.dns.qry_type,
                               'qry_class': packet.dns.qry_class,
                               'registrar': None, 'creation_date': None,
                               'in_blacklists':
                               check_blacklist(packet.dns.qry_name),
                               'whitelisted': check_whitelist(packet.dns.qry_name),
                               'ns': None, 'mx': None, 'cname': None, 'txt': None,
                               'time': None, 'ptr': None, 'timestamp': None,
                               'asn': None, 'isp': None, 'nxdomain': None}
                try:
                    geo_result = check_geo(packet.dns.a)
                    packet_dict.update({'dst': packet.dns.a})
                    packet_dict.update({'ns': packet.dns.ns})
                    packet_dict.update({'mx': packet.dns.mx_mail_exchange})
                    packet_dict.update({'cname': packet.dns.cname})
                    packet_dict.update({'txt': packet.dns.txt})
                    packet_dict.update({'ptr': packet.dns.ptr_domain_name})
                    packet_dict.update({'time': packet.dns.time})
                    packet_dict.update({'nxdomain': packet.dns.nxdomain})
                except AttributeError:
                    print("Resource type not found in packet")
                if whois_result:
                    packet_dict.update({'registrar': whois_result['registrar']})
                    packet_dict.update({'creation_date':
                    whois_result['creation_date']})
                if geo_result:
                    packet_dict.update({'asn': geo_result['asn']})
                    packet_dict.update({'isp': geo_result['isp']})
                update_db(create_graph, packet_dict)
    elif filetype == 'txt':
        with open(filename, "r") as logfile:
            i = 0
            for line in logfile:
                fields = line.split(" ")
                domain_name = remove_chars(fields[4])
                whois_result = check_whois(domain_name)
                try:
                    packet_dict = {'timestamp': fields[0] + ' ' + fields[1],
                    'src': fields[3], 'host': domain_name,
                    'in_blacklists': check_blacklist(domain_name),
                    'registrar': None, 'creation_date': None, 'whitelisted':
                    check_whitelist(domain_name), 'ns': None, 'mx': None,
                    'cname': None, 'txt': None, 'time': None, 'ptr': None,
                    'dst': None}
                    if whois_result:
                        packet_dict.update({'registrar': whois_result['registrar']})
                        packet_dict.update({'creation_date':
                        whois_result['creation_date']})
                    update_db(create_graph, packet_dict)
                except AttributeError:
                    print("Resource type not found in packet")
    elif filetype == 'csv':
        load_csv()
    else:
        print("Filetype not supported")
```

Code listing A.3: Checks if domain names in log files are known legitimate domains

```
def check_whitelist(domain_name):
    in_list = False
    whitelist = csv.reader(open("Whitelists/majestic_1000.csv", "r"), delimiter=",")
    for line in whitelist:
        if line[2] == domain_name or ('www.' + line[2]) == domain_name:
            in_list = True
            break
    return in_list
```

Code listing A.4: Checks if domain names are found in any domain blacklists

```
def check_blacklist(domain_name):
    in_list = False
    malwaredomainlist = open("Blacklists/malwaredomainlist_hosts.txt", "r")
    urlhaus = open("Blacklists/urlhaus.txt", "r")
    phishtank = csv.reader(open("Blacklists/verified_online(phishtank).csv", "r"),
    delimiter=",")
    cybercrime_tracker = open("Blacklists/CYBERCRiME-06-03-20.txt", "r")
    blacklists = [malwaredomainlist, urlhaus]
    for bl in blacklists:
        domains = []
        for line in bl:
            line = line.split("  ")
            if line[0] == '127.0.0.1':
                domains.append(line)
        for line in domains:
            if line[1] == domain_name or ('www.' + line[1]) == domain_name:
                in_list = True
                break
    for line in phishtank:
        variations = [str(domain_name) + "/", "www." + str(domain_name),
        domain_name[4:]]
        if line[1][7:] in variations or line[1][8:] in variations:
            in_list = True
            break
    for line in cybercrime_tracker:
        variations = [str(domain_name) + "/", "www." + str(domain_name),
        domain_name[4:]]
        if line in variations:
            in_list = True
            break
    for f in blacklists:
        f.close()
    cybercrime_tracker.close()
    return in_list
```

Code listing A.5: Finds registrar info for a specific domain name

```
def check_whois(domain):
    result = None
    try:
        whois_query = whois.query(domain)
        cached = False
        if whois_query is not None:
            if whois_query.registrar is not '':
                with open('datasets/whois_cache.csv', 'r') as cache:
                    reader = csv.reader(cache, delimiter=',')
```

```
                    for line in reader:
                        if domain == line[0]:
                            cached = True
                            result = {"registrar": line[1],
                                      "creation_date": line[2]}
                            break
                    if not cached:
                        with open('datasets/whois_cache.csv', 'a') as out_file:
                            writer = csv.writer(out_file)
                            writer.writerow((domain, whois_query.registrar,
                            whois_query.creation_date))
                            result = {"registrar": whois_query.registrar,
                                      "creation_date": whois_query.creation_date}
    except whois.exceptions.UnknownTld:
        print("Unknown TLD")
    except whois.exceptions.WhoisCommandFailed:
        print("Command timed out")
    except (whois.exceptions.FailedParsingWhoisOutput, ValueError):
        print("Error in output")
    except KeyError:
        print("Key error")
    except whois.exceptions.UnknownDateFormat:
        print("Unknown date format")
    return result
```

Code listing A.6: Checks if a resolved IP address is found in any IP blacklists

```
def check_ip(ip):
    in_list = False
    firehol = open("Blacklists/firehol_level1.netset", "r")
    malwaredomainlist_ip = open("Blacklists/malwaredomainlist_ip.txt", "r")
    cinsscore = open("Blacklists/ci-badguys.txt", "r")
    blacklists = [firehol, malwaredomainlist_ip, cinsscore]
    for blacklist in blacklists:
        for line in blacklist:
            if line[0] is not '#':
                if line is ip or ipaddress.IPv4Address(ip)
                in ipaddress.IPv4Network(line.strip('\n')):
                    in_list = True
                    break
    for file in blacklists:
        file.close()
    return in_list
```

Code listing A.7: Finds ASN and ISP for each IP

```
def check_geo(ip):
    ip_version = ipaddress.ip_address(str(ip))
    if ip_version.version == 4:
        with open('datasets/GeoLite2-ASN-Blocks-IPv4.csv', newline='') as ipv4_list:
            reader = csv.reader(ipv4_list, delimiter=',')
            next(reader, None)
            for line in reader:
                if ipaddress.IPv4Address(ip) in ipaddress.IPv4Network(line[0]):
                    return {"asn": line[1], "isp": line[2]}
    else:
        with open('datasets/GeoLite2-ASN-Blocks-IPv6.csv', newline='') as ipv6_list:
            reader = csv.reader(ipv6_list, delimiter=',')
            next(reader, None)
```

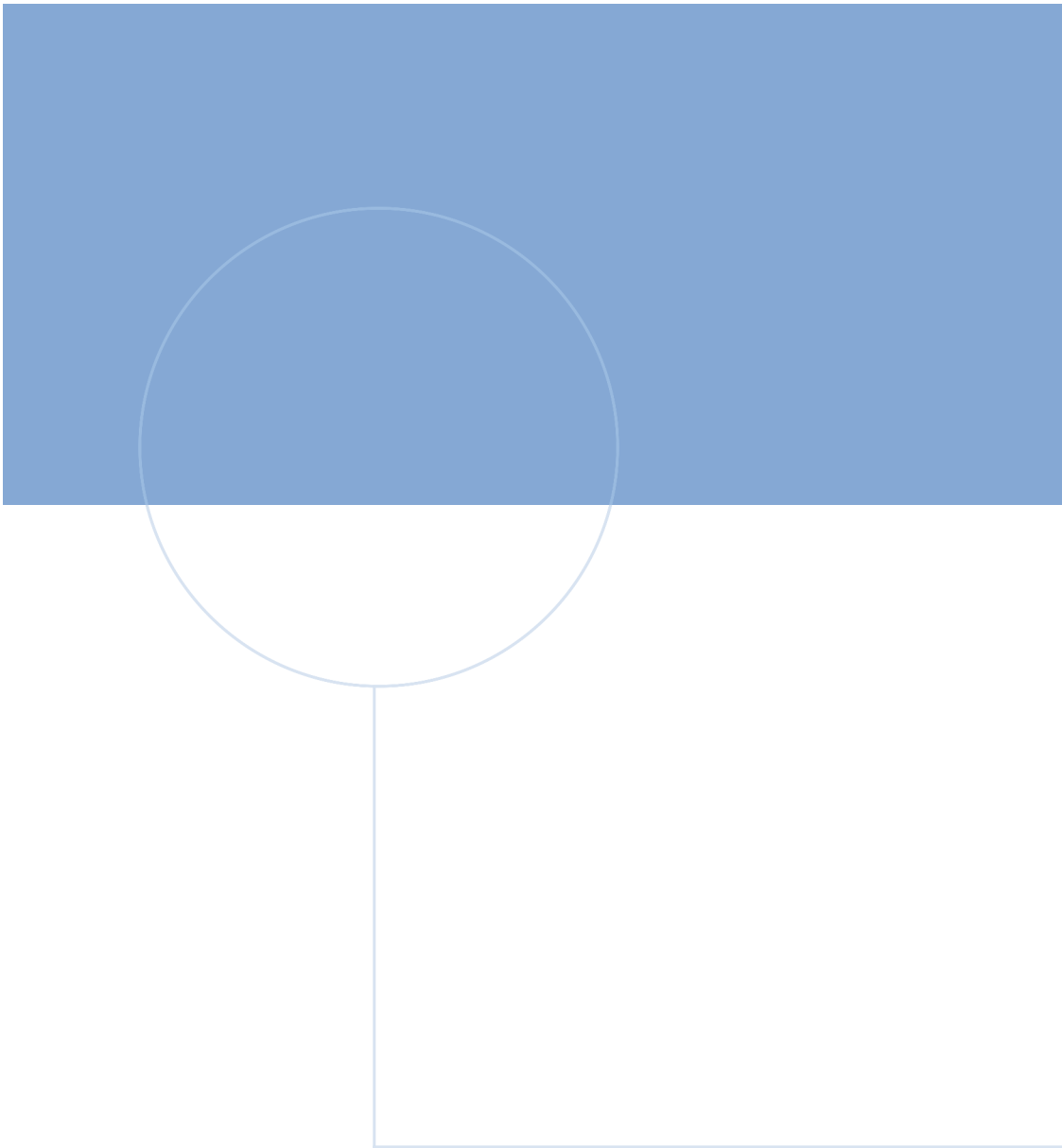
```
            for line in reader:
                if ipaddress.IPv6Address(ip) in ipaddress.IPv6Network(line[0]):
                    return {"asn": line[1], "isp": line[2]}
```

Code listing A.8: Converts dataset from TXT to CSV format

```
import csv

def import_dataset(filename):
    with open(filename, 'r') as in_file:
        stripped = (line.strip() for line in in_file)
        lines = (line.split(" ") for line in stripped if line)
        unwanted = ['"', '""']
        with open('datasets/eidsiva_test.csv', 'w') as out_file:
            writer = csv.writer(out_file)
            writer.writerow(('date', 'time', 'user', 'src', 'domain_par',
            'view', 'ntp-stealth', 'query', 'domain_name',
                            'in', 'rr_type', '+', 'dst'))
            for line in lines:
                write = True
                for field in line:
                    for char in unwanted:
                        if char in field:
                            write = False
                if write:
                    writer.writerow(line)
                else:
                    print(line)

import_dataset("datasets/anon_dns_records.txt")
```



NTNU

Norwegian University of
Science and Technology